Environmental Bisimulations for Higher-Order Languages

Davide Sangiorgi
University of Bologna
Davide.Sangiorgi@cs.unibo.it

Naoki Kobayashi Eijiro Sumii
Tohoku University
{koba,sumii}@ecei.tohoku.ac.jp

Abstract

Developing a theory of bisimulation in higher-order languages can be hard. Particularly challenging can be: (1) the proof of congruence, as well as enhancements of the bisimulation proof method with "up-to context" techniques, and (2) obtaining definitions and results that scale to languages with different features.

To meet these challenges, we present environmental bisimulations*, a form of bisimulation for higher-order languages, and its basic theory. We consider four representative calculi: pure λ-calculi (call-by-name and call-by-value), call-by-value λ-calculus with higher-order store, and then Higher-Order π-calculus. In each case: we present the basic properties of environmental bisimilarity, including congruence; we show that it coincides with contextual equivalence; we develop some up-to techniques, including up-to context, as examples of possible enhancements of the associated bisimulation method.*

Unlike previous approaches (such as applicative bisimulations, logical relations, Sumii-Pierce-Koutavas-Wand), our method does not require induction/indices on evaluation derivation/steps (which may complicate the proofs of congruence, transitivity, and the combination with up-to techniques), or sophisticated methods such as Howe's for proving congruence. It also scales from the pure λ-calculi to the richer calculi with simple congruence proofs.

1 Introduction

Behavioral equivalence and bisimulation in higher-order languages. Proving equivalence of computer programs is an important but challenging problem. Equivalence between two programs means that the programs should behave "in the same manner" under any context [24]; this notion of equality is called *contextual equivalence*. Finding effective methods for equivalence proofs is particularly challenging in *higher-order* languages (i.e., languages where program code can be passed around, as opposed to *first-order* languages).

Bisimulation has emerged as a powerful operational method for proving equivalence of programs in various kinds of languages, due to the associated co-inductive proof method. Further, a number of enhancements of the bisimulation method have been studied, usually called *up-to techniques*. To be useful, the behavioral relation resulting from bisimulation—*bisimilarity*—should be a *congruence*; and preferably it should coincide with contextual equivalence. For first-order languages, there is a common consensus about what bisimulation is and how it should be defined, and the associated proof techniques are well-developed (e.g., techniques for proving congruence, and up-to techniques).

The picture is less clear for higher-order languages, as available definitions and proof techniques are often difficult to adapt to different languages. We informally discuss some key design issues for bisimulation, and why previous proposals of bisimulations are not robust enough, using an abstract form of transitions for higher-order calculi, and occasionally referring to concrete calculi such as λ-calculi and Higher-Order π-calculus (HOπ). (The actual syntax for transitions in the calculi of the paper will be slightly different; for instance in HOπ we will use the "early" style, rather than the "late" one as below.) Transitions can take three forms:

- *First-order* transitions $P \xrightarrow{\ell} P'$ (the label ℓ is first-order, that is, it does not contain terms). Reductions in pure λ-calculi and τ-transitions in HOπ are of this kind (in the former case, ℓ is omitted).
- *Higher-order output* transitions $P \xrightarrow{\ell} \langle P_1 \rangle P_2$, in which a higher-order value P_1 is produced and P_2 is the continuation. In pure λ-calculi this occurs when P is a value (thus P itself is the value emitted and there is no continuation). In HOπ it occurs when an output prefix is consumed, as in $\overline{a}P_1.\, P_2 \xrightarrow{a} \langle P_1 \rangle P_2$, where process P_1 is emitted along channel a.
- *Higher-order input* transitions $P \xrightarrow{\ell} (x)P'$, in which the abstraction $(x)P'$ so revealed then calls for a higher-order value to instantiate the variable x. In λ-calculus, abstractions are produced by terms such as $\lambda x.\, Q$; in HOπ, by input prefixes such as $a(x).\, Q$.

The bisimulation clause for first-order transitions is uncontroversial, and is the standard one for first-order languages. The main difficulty is finding the appropriate clauses for the higher-order transitions. Suppose $\mathcal{R}$ is a bisimulation, with (P, Q) in $\mathcal{R}$. Consider matching output transitions

$$P \xrightarrow{\ell} \langle P_1 \rangle P_2 \quad \text{and} \quad Q \xrightarrow{\ell} \langle Q_1 \rangle Q_2. \qquad (*)$$

How should P_1, P_2, Q_1, Q_2 be related? Imposing P_1 bisimilar to Q_1, and P_2 bisimilar to Q_2, can be too strong a requirement. For instance, in HOπ and in calculi with information hiding (or generative names), it breaks the correspondence with contextual equivalence.

On the other hand, matching input transitions

$$P \xrightarrow{\ell} (x)P' \quad \text{and} \quad Q \xrightarrow{\ell} (x)Q' \qquad (**)$$

raise the question of what should be substituted for x; i.e., for which terms P_1 and Q_1 should we require $P'\{^{P_1}/_x\}\ \mathcal{R}\ Q'\{^{Q_1}/_x\}$? We discuss some possible choices:

- P_1 and Q_1 are all pairs of identical terms, as in applicative bisimulations (the most studied form of bisimulation for higher-order calculi, e.g., [2, 10, 19, 25, 27, 30]). This is unsound under the presence of generative names, data abstraction, or encryption [13, 41, 42]. Moreover, proving that bisimilarity is a congruence can be hard. To see why, consider an application context, and a pair of bisimilar functions M, N plus a pair of bisimilar arguments M', N'. We have to prove that MM' and NN' are bisimilar, but we are unable to apply the bisimulation hypothesis on the functions M and N since their arguments M' and N' are bisimilar but not necessarily identical. Difficulties also arise with up-to context techniques (see [17, 19, 30] for the usefulness of these techniques in higher-order languages and the problems with applicative bisimulations).
- P_1 and Q_1 are related by $\mathcal{R}$. This makes the above congruence argument for MM' and NN' work. However, this definition of bisimulation, which we call a *BA-bisimulation*,[1] breaks the monotonicity of the generating functional (the function from relations to relations that represents the clauses of bisimulation). Indeed, BA-bisimulations in general are unsound. For instance, take the identity function $I = \lambda x.\, x$ and $\Sigma = EE$ where $E = \lambda x.\, \lambda y.\, xx$. Term Σ is a "purely convergent term" because it always reduces to itself when applied to any argument, regardless of the input received. Of course I and Σ should not be regarded as bisimilar, yet $\{(I, \Sigma)\}$ would be a BA-bisimulation (the only related input is the pair (I, Σ) itself, and the result of the application is again the same pair).
- P_1, Q_1 are fresh variables. This bisimulation method [32, 39] is complete (with respect to contextual equivalence) only in certain extensions of the λ-calculus (e.g., call-by-value with *both* state *and* callcc).

[1] BA indicates that the bisimilarity uses "Bisimilar Arguments."

Environmental bisimulations and paper contributions. In this paper we propose *environmental bisimulations* as a bisimulation method for higher-order languages.

A key idea of environmental bisimulations is to make a clear distinction between the tested terms and the environment. An element of an environmental bisimulation has, in addition to the tested terms P and Q, a further component $\mathcal{E}$, the environment, which expresses the observer's current knowledge. (In languages richer than pure λ-calculi, there may be other components, for instance to keep track of generated names.) The bisimulation requirements for higher-order inputs and outputs naturally follow. In the higher-order outputs $(*)$, (P_1, Q_1) are published to the environment, so they should be put into $\mathcal{E}$. Thus roughly the clause becomes:

- if $(\mathcal{E}, P, Q) \in \mathcal{R}$ and $P \xrightarrow{\ell} \langle P_1 \rangle P_2$
  then $Q \xrightarrow{\ell} \langle Q_1 \rangle Q_2$ and $(\mathcal{E} \cup \{(P_1, Q_1)\}, P_2, Q_2) \in \mathcal{R}$.

In the higher-order inputs $(**)$, the arguments P_1 and Q_1 should be terms that the observer can build using the current knowledge; that is, terms obtained by composing the values in $\mathcal{E}$ using the operators of the calculus. We write $\mathcal{E}^\star$ for pairs of terms of this form. The clause roughly becomes:

- if $(\mathcal{E}, P, Q) \in \mathcal{R}$ and $P \xrightarrow{\ell} (x)P'$ then $Q \xrightarrow{\ell} (x)Q'$ and $(\mathcal{E}, P'\{^{P_1}/_x\}, Q'\{^{Q_1}/_x\}) \in \mathcal{R}$ for any $(P_1, Q_1) \in \mathcal{E}^\star$.

Finally, we need clauses to express the observer's test capabilities on the environment. For instance, in λ-calculi the observer can check the consistency of values related in $\mathcal{E}$ (e.g., the outermost construct should be the same); in HOπ, the observer is allowed to run, at any time, processes that are related in the environment, which yields the clause:

- $(\mathcal{E}, P, Q) \in \mathcal{R}$ implies $(\mathcal{E}, P \mid P_1, Q \mid Q_1) \in \mathcal{R}$, for $(P_1, Q_1) \in \mathcal{E}$.

As for BA-bisimulations, so in environmental bisimulations testing higher-order inputs on *related* arguments facilitates proofs of congruence (and up-to contexts). But unlike BA-bisimulations, the separation between environment and tested terms maintains the monotonicity of the generating functional.

A possible drawback of environmental bisimulations over, say, applicative bisimulations is that the set of arguments to related functions that have to be considered in the bisimulation clause is larger (since it also includes non-identical arguments). As a remedy to this, we propose the use of up-to techniques (in particular techniques involving

up-to contexts), which are easier to establish for environmental bisimulations than for applicative bisimulations, and which allow us to considerably enhance the bisimulation proof method.

We use a small-step, rather than big-step, semantics in environmental bisimulations. This is important in non-confluent languages but may seem cumbersome in, e.g., λ-calculi, because it seems to require more elements in bisimulations than big-step semantics. Again, we remedy this with up-to techniques (such as "up-to reduction"). Further, small-step semantics together with up-to techniques sometimes *simplifies* equivalence proofs, as we can exploit the possibility of comparing terms in the middle of evaluations without having to reduce them to values (e.g., Example 4.6). Big-step versions of environmental bisimulations are anyway derived as a corollary of soundness of certain up-to techniques.

To test the robustness of environmental bisimulations, we transport definitions and proof techniques from pure λ-calculi to λ-calculi with full-fledged store (a language with information hiding by generative names), and to Higher-Order π-calculi (as examples of concurrent, non-deterministic languages). In each case: we present the basic properties of environmental bisimilarity, including its congruence properties; we show that it coincides with contextual equivalence (in concurrency, this is barbed congruence); we develop a few up-to techniques as examples of possible enhancements of the associated bisimulation method. These techniques include *up-to contexts*, *up-to expansion*, and *up-to full contexts* in which the erased contexts can bind free variables of their arguments.

In the languages without references, *logical bisimulation*—a form of environmental bisimulations without explicit environment (the environment is taken to be the bisimulation itself)—is also a viable technique. In logical bisimulation the generating functional is non-monotone. We show in [34] (this is further developped in [35]) that the functional has nevertheless a greatest fixed-point that coincides with contextual equivalence.

Environmental bisimulations have been inspired by bisimulations for higher-order calculi with information hiding mechanisms (encryption [41], data abstraction [42], and store [17]), where an environment was necessary because of the information hiding. In this respect, our contribution is to isolate this idea, simplify and strengthen the method, develop its basic theory, so to propose it as a general method for higher-order languages. See Section 6 for more details.

We sometimes refer to [34], which contains further details on definitions and results that, due to a lack of space, are not included in the present paper.

2 Preliminaries

We introduce general notations and terminologies for the paper. Familiarity with standard terminologies (such as free/bound variables, and α-conversion) is assumed.

We use meta-variables $M, N, P, Q, \ldots$ for terms, and $V, W, \ldots$ for values (where the notion of terms and values varies depending on the calculus being considered). We identify α-convertible terms. We write $M\{^N/_x\}$ for the capture-avoiding substitution of N for x in M. A term is *closed* if it contains no free variables. The set of free variables of a term M is $\mathsf{fv}(M)$. A *context* C is an expression obtained from a term by replacing some sub-terms with *holes* of the form $[\cdot]_i$. We write $C[M_1, \ldots, M_n]$ for the term obtained by replacing each occurrence of $[\cdot]_i$ in C with M_i.

We use meta-variables $\mathcal{R}, \mathcal{S}, \mathcal{E}, \mathcal{F}, \ldots$ for binary relations; $\mathcal{R}\,\mathcal{S}$ is the composition of $\mathcal{R}$ and $\mathcal{S}$, whereas $\mathcal{R}^\star$ is the *closure of relation $\mathcal{R}$ under contexts*, i.e.

$$\{(C[M_1, \ldots, M_n], C[N_1, \ldots, N_n]) \text{ s. t. } M_i \mathcal{R} N_i, \forall i\}$$

and contains both R and the identity relation. By default, we restrict $\mathcal{R}^\star$ to closed terms unless noted otherwise.

Sequences $M_1, \ldots, M_n$ are often abbreviated to $\widetilde{M}$, and notations are extended to tuples componentwise. Hence, we write $C[\widetilde{M}]$ for $C[M_1, \ldots, M_n]$, and $\widetilde{M}\mathcal{R}\widetilde{N}$ for $(M_1 \mathcal{R} N_1) \wedge \cdots \wedge (M_n \mathcal{R} N_n)$. We have some remarks on the results of this paper:

- Although the results are often stated for closed values only, they can be generalized to open terms in a common way. In λ-calculi, this can be done by defining an ad hoc relation—the least congruence containing $(M, (\lambda x.\, M)x)$ for every M—and proving its preservation under evaluation, as in Sumii-Pierce [41] and Koutavas-Wand [17]. (Alternatively, we may also consider a bisimulation between M and $(\lambda x.\, M)x$. The proof is straightforward in either case.) Thus properties between open terms M and N can be derived from the corresponding properties between the closed terms $\lambda\widetilde{x}.\, M$ and $\lambda\widetilde{x}.\, N$, for $\{\widetilde{x}\} \supseteq \mathsf{fv}(M) \cup \mathsf{fv}(N)$. In HO$\pi$, it is similar; for instance, if $\mathsf{fv}(M) \subseteq \{x\}$ then one relates processes M and $\boldsymbol{\nu} a(a(x).\, M \mid \overline{a}x.\, \mathbf{0})$, where a is a fresh name.
- The results are stated for untyped languages. Adapting them to languages with a simply-typed discipline is straightforward. (We use a simply-typed calculus in an example.)

3 Call-by-name λ-calculus

The set Λ of pure λ-terms is defined by:

$$M, N ::= x \mid \lambda x.\, M \mid MN$$

We write $\Lambda^\bullet$ for the subset of closed terms. The *call-by-name reduction relation* $\longrightarrow$ is the least relation over $\Lambda^\bullet$ closed under the following rules.

$$\beta: \ (\lambda x.\, M)N \longrightarrow M\{^N/_x\} \qquad \mu: \ \frac{M \longrightarrow M'}{MN \longrightarrow M'N}$$

We write $\Longrightarrow$ for the reflexive and transitive closure of $\longrightarrow$. The values are the terms of the form $\lambda x.\, M$.

Environmental bisimulation. An *environmental relation* is a set of elements each of which is of the form $(\mathcal{E}, M, N)$ or $\mathcal{E}$, and where M, N are closed terms and $\mathcal{E}$ is a relation on closed values. We use $\mathcal{X}, \mathcal{Y}$ to range over environmental relations. In a triple $(\mathcal{E}, M, N)$ the relation component $\mathcal{E}$ is the *environment*, and M, N are the *tested terms*. We write $M\ \mathcal{X}_\mathcal{E}\ N$ for $(\mathcal{E}, M, N) \in \mathcal{X}$.

Definition 3.1 *An environmental relation $\mathcal{X}$ is an* environmental bisimulation *if*

1. *$M\ \mathcal{X}_\mathcal{E}\ N$ implies:*
   - *(a) if $M \longrightarrow M'$ then $N \Longrightarrow N'$ and $M'\ \mathcal{X}_\mathcal{E}\ N'$*
   - *(b) if $M = V$ then $N \Longrightarrow W$ and $\mathcal{E} \cup \{(V, W)\} \in \mathcal{X}$*
   - *(c) the converse of the above two conditions, on N*
2. *if $\mathcal{E} \in \mathcal{X}$ then for all $(\lambda x.\, P, \lambda x.\, Q) \in \mathcal{E}$ and for all $(M_1, N_1) \in \mathcal{E}^\star$ it holds that $P\{^{M_1}/_x\}\ \mathcal{X}_\mathcal{E}\ Q\{^{N_1}/_x\}$.*

We write $\approx$ for the union of all environmental bisimulations, and call it environmental bisimilarity.

Relations $\approx_\mathcal{E}$, as well as the other bisimulation equivalences in the paper, are extended to open terms using closing abstractions. Thus if $\widetilde{x} = \mathsf{fv}(M, N)$ then $M \approx_\mathcal{E} N$ if $\lambda\widetilde{x}.\, M \approx_\mathcal{E} \lambda\widetilde{x}.\, N$.

For examples and applications of $\approx_\mathcal{E}$, the most important case is when $\mathcal{E} = \emptyset$: this states the equivalence between two terms without any predefined knowledge from the observer. We therefore introduce a special symbol for it, and write $M \simeq N$ as an abbreviation for $M \approx_\emptyset N$.

Example 3.2 *We have $I_1 \simeq I_2$ for $I_1 \stackrel{def}{=} \lambda x.\, x$ and $I_2 \stackrel{def}{=} (\lambda x.\, x)(\lambda x.\, x)$, by taking $\mathcal{X} = \{(\mathcal{E}, I_1, I_2) \mid \mathcal{E} \subseteq Id\} \cup \{(\mathcal{E}, M, M) \mid \mathcal{E} \subseteq Id \wedge M \in \Lambda^\bullet\} \cup \{\mathcal{E} \mid \mathcal{E} \subseteq Id\}$ for $Id = \{(V, V) \mid V \in \Lambda^\bullet\}$. Note that the singleton set $\{(\emptyset, I_1, I_2)\}$ by itself is not an environmental bisimulation because of the $\mathcal{E}^\star$ in clause (2). Burdens like this (or more) will be removed by the up-to techniques described later in this section. Specifically, the finite set $\{(\emptyset, I_1, I_2), \emptyset\}$ will be an environmental bisimulation "up to contexts."*

Example 3.3 *We have $I_1 \simeq I_3$ for $I_1 \stackrel{def}{=} \lambda x.\, x$ and $I_3 \stackrel{def}{=} \lambda x.\, (\lambda y.\, y)x$, by taking $\mathcal{X} = \{(\mathcal{E}, M, N), (\mathcal{E}, M, (\lambda y.\, y)N) \mid \mathcal{E} \subseteq S^\star \wedge M\ S^\star\ N\} \cup \{\mathcal{E} \mid \mathcal{E} \subseteq S^\star\}$ for $S = \{(I_1, I_3)\}$. Proving this to be an environmental bisimulation only using the definition above is even harder: we need an induction on contexts in clause (1.a). Again, with up-to techniques, the finite set $\{(\mathcal{S}, I_1, I_3), \mathcal{S}\}$ suffices (it is an environmental bisimulation up to contexts and reduction).*

Basic properties. It is immediate to check that the union of bisimulations is a bisimulation itself; hence we derive:

Lemma 3.4 *$\approx$ is the largest environmental bisimulation.*

Lemma 3.5 *$M \approx_\mathcal{E} N$ and $\mathcal{E}' \subseteq \mathcal{E}$ imply $M \approx_{\mathcal{E}'} N$.*

Lemma 3.6 *$M \approx_{\mathcal{E}_1} N$ and $N \approx_{\mathcal{E}_2} L$ imply $M \approx_{\mathcal{E}_1\,\mathcal{E}_2} L$.*

The last lemma can be proved by showing $\{\mathcal{E}' \mid \mathcal{E}' \subseteq \mathcal{E}_1\,\mathcal{E}_2 \text{ for } \mathcal{E}_1, \mathcal{E}_2 \in \approx\} \cup \{(\mathcal{E}', P, R) \mid \mathcal{E}' \subseteq \mathcal{E}_1 \circ \mathcal{E}_2,\ P \approx_{\mathcal{E}_1} Q \approx_{\mathcal{E}_2} R\}$ to be an environmental bisimulation. A corollary of it is the transitivity of $\simeq$.

We now consider the congruence properties of the bisimilarity. In bisimilarities for higher-order languages, these are usually the most delicate basic properties to establish. With the exception of Lemma 3.10, the properties below will hold in all the λ-calculi we consider in the paper (with occasional minor adjustments). We write $\widehat{\approx}_\mathcal{E}$ for the restriction of $\approx_\mathcal{E}$ to values, and similarly for $\widehat{\simeq}$.

Lemma 3.7 (congruence for values) *For all $\mathcal{E}$, relation $\widehat{\approx}_\mathcal{E}$ is a congruence. In particular, $\widehat{\simeq}$ is a congruence.*

In call-by-name *evaluation contexts* are described by the following grammar: $C := CM \mid [\cdot]$

Lemma 3.8 (arbitrary terms under evaluation contexts) *For all $\mathcal{E}$, relation $\approx_\mathcal{E}$ is preserved by evaluation contexts (i.e., if $M \approx_\mathcal{E} N$ and C is an evaluation context, then also $C[M] \approx_\mathcal{E} C[N]$).*

Lemmas 3.7 and 3.8 are proved simultaneously, defining appropriate bisimulations and reasoning by induction on contexts.

We write $\lambda.\, M$ for the thunk obtained from M (i.e., a term $\lambda x.\, M$ with $x \notin \mathsf{fv}(M)$).

Lemma 3.9 (congruence for arbitrary terms, thunked) *For all $\mathcal{E}$, if $\lambda.\, M \approx_\mathcal{E} \lambda.\, N$, then also $C[M] \approx_\mathcal{E} C[N]$, for all contexts C.*

The proof uses Lemma 3.7 and validity of β-conversion (the fact that $(\lambda x.\, M)N \simeq M\{^N/_x\}$, for any $\lambda x.\, M$ and N closed, is preserved by any context).

In the case of call-by-name and other pure λ-calculi, the above result can be simplified and strengthened: for all $\mathcal{E} \in \approx$, relation $\approx_\mathcal{E}$ is a congruence. In particular we have:

Lemma 3.10 *Relation $\simeq$ is a congruence relation.*

Again, the proof is by induction on contexts. (In richer λ-calculi the statement of this lemma may need to be refined, see e.g., [34, Remark D.2].)

Up-to techniques. We prove Lemma 3.10 using a few "up-to" techniques, as enhancements of the bisimulation proof method. We introduce these techniques, and a few others, below. Such techniques allow us to prove bisimulation results using relations that in general are not themselves bisimulations, but are contained in a bisimulation. The full definitions are given in [34, Appendix A]; here, we simply indicate the modifications to the bisimulation clauses (i.e., the clauses of Definition 3.1). We omit the statements of soundness of the techniques.

Up-to environment. This technique introduces some flexibility in the choice of the environment for two tested terms, by allowing environments that are larger than those requested by Definition 3.1 (by Lemma 3.5, a larger environment gives a stronger requirement). For instance, the technique can allow us to avoid environments that incrementally grow during the bisimulation game, retaining instead only their limit (i.e., the largest environment). In clause (1.a), we replace "$M' \ \mathcal{X}_{\mathcal{E}} \ N'$" with "$M' \ \mathcal{X}_{\mathcal{E}'} \ N'$ for some $\mathcal{E}'$ with $\mathcal{E} \subseteq \mathcal{E}'$," and similarly in (2); in (1.b), we replace "$\mathcal{E} \cup \{(V,W)\} \in \mathcal{X}$" with "$\mathcal{E}' \in \mathcal{X}$ for some $\mathcal{E}'$ with $\mathcal{E} \cup \{(V,W)\} \subseteq \mathcal{E}'$."

Up-to bisimilarity. This technique introduces a (limited) use of $\simeq$ on tested terms. This can allows us to avoid bisimulations with elements that, behaviorally, are the same. In clause (1.a), we replace "$M' \ \mathcal{X}_{\mathcal{E}} \ N'$" with "$M' \ \mathcal{X}_{\mathcal{E}} \simeq N'$"; in (1.b), we replace "$\mathcal{E} \cup \{(V,W)\} \in \mathcal{X}$" with "$\mathcal{E} \cup \{(V,W')\} \in \mathcal{X}$ for some W' with $W \simeq W'$"; in (2), we replace "$P\{^{M_1}/_x\} \ \mathcal{X}_{\mathcal{E}} \ Q\{^{N_1}/_x\}$" with "$P\{^{M_1}/_x\} \simeq \mathcal{X}_{\mathcal{E}} \simeq \ Q\{^{N_1}/_x\}$." We cannot strengthen up-to bisimilarity by using $\simeq$ also on the left-hand side of $\mathcal{X}$ in clauses (1.a) and (1.b), for the technique would be unsound; this is similar to the problems of up-to bisimilarity in standard small-step bisimilarity [22].

Up-to reduction and up-to expansion. This technique exploits the confluent property of reduction so to replace tested terms with derivatives of them. When reduction is confluent this technique avoids the main disadvantage of small-step bisimulations over the big-step ones, namely the need of considering each single derivative of a tested term.

In clause (1.a), we replace "$M' \ \mathcal{X}_{\mathcal{E}} \ N'$" with "there are M'', N'' with $M' \Longrightarrow M''$ and $N' \Longrightarrow N''$ such that $M'' \ \mathcal{X}_{\mathcal{E}} \ N''$"; similarly, in (2) replace "$P\{^{M_1}/_x\} \ \mathcal{X}_{\mathcal{E}} \ Q\{^{N_1}/_x\}$" with "there are M', N' with $P\{^{M_1}/_x\} \Longrightarrow M'$ and $Q\{^{N_1}/_x\} \Longrightarrow N'$ such that $M' \ \mathcal{X}_{\mathcal{E}} \ N'$."

The technique allows us to derive the soundness of the "big-step" version of environmental bisimulation, in which clauses (1.a) and (1.b) are unified by requiring that

- if $M \Longrightarrow V$ then $N \Longrightarrow W$ and $\mathcal{E} \cup \{(V,W)\} \in \mathcal{X}$.

Up-to contexts. This technique allows us to cancel a common context in tested terms, requiring instead that only the arguments of such context be pairwise related. Thus in clauses (1.a) and (2) the final occurrence of $\mathcal{X}_{\mathcal{E}}$ is replaced by $\mathcal{X}_{\mathcal{E}}{}^{\star}$, and in clause (1.b), "$\mathcal{E} \cup \{(V,W)\} \in \mathcal{X}$" is replaced by "$\mathcal{E} \cup \{(V,W)\} \subseteq \mathcal{E}'^{\star}$, for some $\mathcal{E}' \in \mathcal{X}$."

Up-to expansion and up-to full contexts. More powerful than up-to reduction and up-to contexts are the *up-to expansion* and *up-to full contexts* techniques. We discuss them in [34, Appendix B], in connection with logical bisimulations. The versions for environmental bisimulation are similar. Up-to reduction and up-to contexts remain however interesting because they are simpler, and easier to combine with other techniques and to adapt to richer languages.

Combinations of up-to. The previous techniques can be combined together, in the expected manner. We shall see an example in Section 4 (up-to environment, reduction and contexts).

Contextual equivalence.

Definition 3.11 (contextual equivalence) *Terms M and N are* contextually equivalent, *written $M \equiv N$, if, for any context C such that $C[M]$ and $C[N]$ are closed, $C[M] \Downarrow$ iff $C[N] \Downarrow$.*

Theorem 3.12 (soundness and completeness of bisimulation) *Relations $\equiv$ and $\simeq$ coincide.*

Logical bisimulations. A special case of environmental bisimulation is when all environments in the triples are the same. In this case we can drop all environments in the bisimulation, if we require that each pair in the environment is also a pair of tested terms. For lack of space we refer to [34] for the details.

Call-by-value λ-calculus. Environmental bisimulations can be adapted to the call-by-value λ-calculus with minor modifications that take into account the different notion of value. We discuss call-by-value λ-calculus in [34, Appendix C], together with an example.

4 Imperative call-by-value λ-calculus

In this section, we study the addition of imperative features (higher-order references, that we call locations). For

this, we use Koutavas and Wand's language [17] (a call-by-value λ-calculus, with locations and a few other auxiliary operators).

The syntax and semantics of the language are given in [34, Appendix D]. It is a standard call-by-value λ-calculus with constants, tuples, creation of new locations, deferencing, and assignment. We use s, t to range over stores, i.e., finite mappings from locations to closed values and l, k over locations. Then $s[l = V]$ is the update of s (possibly an extension of s if l is not in the domain of s). We write $\emptyset$ for the empty store, and $\mathsf{dom}(s)$ for the domain of s. We write $s \uplus s'$ for the union of the two stores when $\mathsf{dom}(s)$ and $\mathsf{dom}(s')$ are disjoint. $\star$ is the unit value (i.e., nullary tuple). We often write $M_1; M_2$ for $(\lambda x.\, M_2)M_1$ when $x \notin \mathsf{fv}(M_2)$. We write $\langle s\,;\, M\rangle \longrightarrow \langle s'\,;\, M'\rangle$ for a small-step reduction relation (where in $\langle s\,;\, M\rangle$, M is a term and s its store), and $\Longrightarrow$ for the reflexive and transitive closure of $\longrightarrow$. We write $\langle s\,;\, M\rangle \Downarrow$ if $\langle s\,;\, M\rangle \Longrightarrow \langle s'\,;\, V\rangle$ for some s' and V. A configuration $\langle s\,;\, M\rangle$ is called *well-formed* if M is closed and all the locations in M and s are in $\mathsf{dom}(s)$.

In this section, $\mathcal{E}^\star$ is the closure of a relation under *location-free* contexts (i.e., contexts without free locations). So, for example, (l, l) is in $\mathcal{E}^\star$ only if $(l, l) \in \mathcal{E}$.

The bisimulation. Let $\mathcal{E}, \mathcal{F}$ range over relations on values. The notion of environmental relation is modified to accommodate stores, which are needed to run terms. Thus the elements of an environmental relation are now of the form $(\mathcal{E}, \langle s\,;\, M\rangle, \langle t\,;\, N\rangle)$ or $(\mathcal{E}, s, t)$. Further, the relation must be *well-formed*, in the sense that in a tuple $(\mathcal{E}, \langle s\,;\, M\rangle, \langle t\,;\, N\rangle)$ the free locations in M and s and all left components of pairs in $\mathcal{E}$ (resp. N and t and all right components of pairs in $\mathcal{E}$) must appear in the domain of s (resp. t). Similarly for an element $(\mathcal{E}, s, t)$. We write $\langle s\,;\, M\rangle\ \mathcal{X}_\mathcal{E}\ \langle t\,;\, N\rangle$ when $(\mathcal{E}, \langle s\,;\, M\rangle, \langle t\,;\, N\rangle) \in \mathcal{X}$.

Definition 4.1 *An environmental relation $\mathcal{X}$ is an* environmental bisimulation *if*

1. $\langle s\,;\, M\rangle\ \mathcal{X}_\mathcal{E}\ \langle t\,;\, N\rangle$ *implies:*

(a) *if* $\langle s\,;\, M\rangle \longrightarrow \langle s'\,;\, M'\rangle$ *then* $\langle t\,;\, N\rangle \Longrightarrow \langle t'\,;\, N'\rangle$ *and* $\langle s'\,;\, M'\rangle\ \mathcal{X}_\mathcal{E}\ \langle t'\,;\, N'\rangle$

(b) *if* $M = V$ *then* $\langle t\,;\, N\rangle \Longrightarrow \langle t'\,;\, W\rangle$ *and* $(\mathcal{E} \cup \{(V, W)\}, s, t') \in \mathcal{X}$

(c) *the converse of the above two conditions, on* N

2. *if* $(\mathcal{E}, s, t) \in \mathcal{X}$ *then for all* $(V, W) \in \mathcal{E}$ *we have:*

(a) $V = c$ *implies* $W = c$

(b) $V = (V_1, \ldots, V_n)$ *implies* $W = (W_1, \ldots, W_n)$ *and* $(\mathcal{E} \cup \{(V_1, W_1), \ldots, (V_n, W_n)\}, s, t) \in \mathcal{X}$

(c) *for all fresh* l, l'*, we have* $(\mathcal{E} \cup \{(l, l')\}, s[l = 0], t[l' = 0]) \in \mathcal{X}$

(d) *if* $V = l$ *then* $W = l'$*, for some* l'*, and moreover,*

i. $(\mathcal{E} \cup \{(s(l), t(l'))\}, s, t) \in \mathcal{X}$

ii. *for all* $(V_1, W_1) \in \mathcal{E}^\star$*, we have* $(\mathcal{E}, s[l = V_1], t[l' = W_1]) \in \mathcal{X}$

(e) *if* $V = \lambda x.\, P$ *then* $W = \lambda x.\, Q$ *and for all* $(V_1, W_1) \in \mathcal{E}^\star$ *it holds that* $\langle s\,;\, P\{V_1/x\}\rangle\ \mathcal{X}_\mathcal{E}\ \langle t\,;\, Q\{W_1/x\}\rangle$

(f) *the converse of the above five conditions, on* W

We write $\approx$ for environmental bisimilarity, the union of all bisimulations. We write $\langle s\,;\, M\rangle\ \approx_\mathcal{E}\ \langle t\,;\, N\rangle$ if $(\mathcal{E}, \langle s\,;\, M\rangle, \langle t\,;\, N\rangle) \in \approx$.

The definition of the environmental bisimulation above may seem much more complex than the definition for pure λ-calculi discussed so far. That is due to the presence of stores and additional language constructs. The essential structure of the definition is the same as the case for pure λ-calculi: The conditions (1.a)–(1.c) are the same except the existence of stores. The conditions (2.a)–(2.f) state that the top-level language constructors of related values are the same (in (2.a) they are constants, in (2.b) tuples, in (2.d) locations, in (2.e) functions; (2.c) accounts for the possibility of extending the stores) and that for every operation on related values, the results are also related.

All the basic properties for environmental bisimulations in Section 3 remain valid with due adjustments. We present congruence results, however, as the presence of locations introduce some subtleties.

Lemma 4.2 (congruence for values)
If $\langle s\,;\, V\rangle \approx_\mathcal{E} \langle t\,;\, W\rangle$*, then* $\langle s\,;\, C[V]\rangle \approx_\mathcal{E} \langle t\,;\, C[W]\rangle$ *for any location-free context* C*.*

The restriction that C is location-free is important (especially concerning those locations in the domain of s). For example, $\langle [l = 0]\,;\, \lambda x.\, l := 0\rangle \approx_\mathcal{E} \langle [l = 0]\,;\, \lambda x.\, l := 1\rangle$ holds for $\mathcal{E} = \{(\lambda x.\, l := 0, \lambda x.\, l := 1)\}$ (intuitively because the effect on updates on l is invisible if the observer can access locations only via $\lambda x.\, l := 0$ and $\lambda x.\, l := 1$). Let $C = ([\cdot]_1 \star; \mathtt{if}\ !l = 0\ \mathtt{then}\ 0\ \mathtt{else}\ \Omega)$ (where Ω is a divergent term). Obviously, $\langle [l = 0]\,;\, C[\lambda x.\, l := 0]\rangle \approx_\mathcal{E} \langle [l = 0]\,;\, C[\lambda x.\, l := 1]\rangle$ does not hold.

To extend Lemma 4.2 to arbitrary contexts (that may contain free locations), we have to make sure that the locations that appear free in the context are related in the bisimulation. One way of expressing this is as follows:

Corollary 4.3 *If* $\langle s\,;\, (V, \widetilde{l})\rangle \approx_\mathcal{E} \langle t\,;\, (W, \widetilde{l'})\rangle$ *where* $\mathsf{dom}(s) = \{\widetilde{l}\}$ *and* $\mathsf{dom}(t) = \{\widetilde{l'}\}$*, then* $\langle s \uplus s'\,;\, C[V]\rangle \approx_\mathcal{E} \langle t \uplus s'\,;\, C[W]\rangle$ *for any context* C *and store* s' *such that* $\langle s \uplus s'\,;\, C[V]\rangle$ *and* $\langle t \uplus s'\,;\, C[W]\rangle$ *are well-formed.*

We establish the correspondent of Lemma 3.8:

Lemma 4.4 [congruence for all terms in evaluation contexts] *$\langle s\,;\,M\rangle \approx_{\mathcal{E}} \langle t\,;\,N\rangle$ implies $\langle s\,;\,C[M]\rangle \approx_{\mathcal{E}} \langle t\,;\,C[N]\rangle$ for any location-free evaluation context C.*

Again, the result can be extended to contexts with free locations as we discussed for Lemma 4.2.

Contextual equivalence and up-to techniques. The bisimulation is sound and complete with respect to contextual equivalence (see [34, Appendix D] for the definition of contextual equivalence in the λ-calculus with store):

Theorem 4.5 (soundness and completeness) *Let V and W be closed and $\mathsf{Loc}(V) \cup \mathsf{Loc}(W) = \{\widetilde{l}\}$. Then, $V \equiv W$ if and only if $\langle [\widetilde{l} = \widetilde{0}]\,;\,(V, \widetilde{l})\rangle \approx_{\emptyset} \langle [\widetilde{l} = \widetilde{0}]\,;\,(W, \widetilde{l})\rangle$.*

The up-to techniques developed for environmental bisimulation in pure λ-calculi are also valid for the imperative calculus, again with the expected modifications due to the enriched language. As an example, in [34, Appendix D] we report the full definition of "up to environment, reduction and contexts" which we use in Example 4.6.

Example 4.6 *(This example is a minor modification of Koutavas and Wand's example [17], Section 6.2). Take:*

$$\begin{array}{rcl} M & \stackrel{def}{=} & \lambda g.\,\boldsymbol{\nu} x\,(g\;\lambda z.\,(x := !x + 2));\\ & & \qquad \lambda z.\,\mathtt{if}\;!x\;\mathtt{mod}\;2 = 0\;\mathtt{then}\;\star\;\mathtt{else}\;\Omega \\ N & \stackrel{def}{=} & \lambda g.\,(g\;\lambda z.\,\star); \lambda z.\,\star \end{array}$$

Intuitively, the two terms are equivalent because the location bound by x will be initialized to 0 and then incremented by 2, and therefore maintains an even number as its content.

We define, as abbreviations

$$\begin{array}{l} P \stackrel{def}{=} \lambda z.\,(l := !l + 2) \qquad\qquad Q \stackrel{def}{=} \lambda z.\,\star \\ R \stackrel{def}{=} \lambda z.\,\mathtt{if}\;!l\;\mathtt{mod}\;2 = 0\;\mathtt{then}\;\star\;\mathtt{else}\;\Omega \\ \mathcal{E}_{\widetilde{l_1};\widetilde{l_2};\widetilde{l'}} \stackrel{def}{=} \{(M,N),(\widetilde{l_1},\widetilde{l_2})\} \cup (\cup_{l\in\widetilde{l'}}\{(P,Q),(R,Q)\}) \end{array}$$

where $\widetilde{l_1} \cap \widetilde{l'} = \emptyset$. Recall that $s \uplus r$ indicates the store composed by s and r, with the implicit assumption that s and r have disjoint domains. We now take $\mathcal{X}$ to be the set consisting of $(\emptyset, \langle\emptyset\,;\,M\rangle, \langle\emptyset\,;\,N\rangle)$, and all triples of the form $(\mathcal{E}_{\widetilde{l_1};\widetilde{l_2};\widetilde{l'}}, s \uplus r, t)$ where $\mathsf{dom}(s) = \widetilde{l_1}$, $\mathsf{dom}(t) = \widetilde{l_2}$, $s(\widetilde{l_1})\;\mathcal{E}^{\star}_{\widetilde{l_1};\widetilde{l_2};\widetilde{l'}}\;t(\widetilde{l_2})$, $\mathsf{dom}(r) = \widetilde{l'}$ and for all $l \in \widetilde{l'}$ we have $r(l) = 2n$, for some n. Note that $\widetilde{l'}$ can also be empty. We show that $\mathcal{X}$ is a bisimulation up to environment, reduction, and contexts. The clauses (2.c) and (2.d) for store extension and locations in the definition are easy. We check the conditions for the pairs (M,N) and (P,Q) (the pair (R,Q) is handled similarly). First, (M,N). Take $(V,W) \in \mathcal{E}^{\star}_{\widetilde{l_1};\widetilde{l_2};\widetilde{l'}}$. Using the abbreviations $M_1 \stackrel{def}{=} (V\;\;P); R$ and $N_1 \stackrel{def}{=} (W\;\;Q); Q$, the terms obtained from MV and NW are $\boldsymbol{\nu} x\,(M_1\{x/l\})$ and N_1. Assuming l fresh, we have

$$\langle s \uplus r\,;\,\boldsymbol{\nu} x\,M_1\{x/l\}\rangle \longrightarrow \langle s \uplus r[l=0]\,;\,M_1\rangle$$

$$\langle t\,;\,N_1\rangle \Longrightarrow \langle t\,;\,N_1\rangle$$

Define the context $C \stackrel{def}{=} ([\cdot]_1\;\;[\cdot]_2)[\cdot]_3$. We have $M_1 = C[V,P,R]$ and $N_1 = C[W,Q,Q]$. This is sufficient, up to contexts, because the arguments of the context are pairwise related in $\mathcal{E}_{\widetilde{l_1};\widetilde{l_2};\widetilde{l'},l}$ and $(\mathcal{E}_{\widetilde{l_1};\widetilde{l_2};\widetilde{l'},l}, s \uplus r[l=0], t) \in \mathcal{X}$.

For the pair (P,Q), the results of applications to related values are $l := !l + 2$ and $\star$, respectively. Further,

$$\langle s \uplus r\,;\,l := !l + 2\rangle \Longrightarrow \langle s \uplus r[l = r(l) + 2]\,;\,\star\rangle$$

and this is sufficient, up to reduction, since

$$(\mathcal{E}_{\widetilde{l_1};\widetilde{l_2};\widetilde{l'}}, s \uplus r[l = r(l) + 2], t) \in \mathcal{X}.$$

5 Higher-order π-calculus

In this section we discuss environmental bisimulations in concurrency. We consider the Higher-Order π-calculus [31, 33] in its simplest form, where only processes can be communicated (thus the calculus is similar to Plain CHOCS [43]). The syntax and the LTS are standard [34, Appendix E]. Restriction and input are binders for names and variables, in the usual way; $\mathsf{fv}(P)$ and $\mathsf{fn}(P)$ indicate the free variables and the free names of P, respectively.

In HOπ, an *environmental relation* is a set of elements of the form $(r\,;\,\mathcal{E}\,;\,M\,;\,N)$, where M, N are closed processes, $\mathcal{E}$ is a relation on closed processes, and r is a finite set of names. Intuitively: M and N are the tested processes; $\mathcal{E}$ is the set of values that the tested processes have produced earlier (by means of outputs towards the observer); r are the names that are known, and freely available, to the observer; these names may occur free in processes of $\mathcal{E}$ and in M, N. To simplify notations, we do not keep track of the other free names in $\mathcal{E}$, M, and N: these represent private names that the tested processes have extruded earlier, and that the observer cannot directly access.

We first present the definition of environmental bisimulation and then we comment it. The definition is given in the "early" style, which makes the comparison with contextual equivalence easier. We write $(r\,;\,\mathcal{E})^{\star}$ as an abbreviation for the subset of $\mathcal{E}^{\star}$ (the context closure of $\mathcal{E}$) in which the free names of the contexts are in r and the bound names are fresh. We write $M\;\;\mathcal{X}_{\mathcal{E}\,;\,r}\;\;N$ if $(r\,;\,\mathcal{E}\,;\,M\,;\,N) \in \mathcal{X}$.

Definition 5.1 *An environmental relation $\mathcal{X}$ is an* environmental bisimulation *if $M\;\mathcal{X}_{\mathcal{E}\,;\,r}\;N$ implies:*

1. *if $M \stackrel{\tau}{\longrightarrow} M'$ then $N \stackrel{\tau}{\Longrightarrow} N'$ and $M'\;\;\mathcal{X}_{\mathcal{E}\,;\,r}\;\;N'$*

2. *if* $M \xrightarrow{aP} M'$ *with* $a \in r$*, and* $(P, Q) \in (r\,;\,\mathcal{E})^\star$*, then* $N \stackrel{aQ}{\Longrightarrow} N'$ *and* $M' \;\; \mathcal{X}_{\mathcal{E}\,;\,r} \;\; N'$

3. *if* $M \xrightarrow{(\boldsymbol{\nu}\widetilde{b})\overline{a}P} M'$ *with* $a \in r$ *and* $\widetilde{b}$ *fresh (i.e., not in* r *and not free in the lhs of* $\mathcal{E}$*), then* $N \stackrel{(\boldsymbol{\nu}\widetilde{c})\overline{a}Q}{\Longrightarrow} N'$ *with* $\widetilde{c}$ *fresh and* $M' \;\; \mathcal{X}_{\mathcal{E}\cup\{(P,Q)\}\,;\,r} \;\; N'$

4. *if* $(P, Q) \in \mathcal{E}$ *then* $P \mid M \; \mathcal{X}_{\mathcal{E}\,;\,r} \; Q \mid N$

5. *for all* r' *fresh (i.e., not in* $\mathsf{fn}(\mathcal{E}, M, N)$*) we have* $M \; \mathcal{X}_{\mathcal{E}\,;\,r,r'} \; N$

6. *the converse of (1-3), on the actions from* N

We write $\approx$ *for the union of all environmental bisimulations, thus* $M \;\; \approx_{\mathcal{E}\,;\,r} \;\; N$ *holds if* $(r\,;\,\mathcal{E}\,;\,M\,;\,N) \in \mathcal{X}$*, for some bisimulation* $\mathcal{X}$*. We sometimes write* $M \simeq N$ *if* $M \;\; \approx_{\emptyset\,;\,\mathsf{fn}(M,N)} \;\; N$.

Definition 5.1 has five clauses. The first is the usual one for τ-actions. The second is analogous to the clause for abstractions in λ-calculi of earlier sections. The inputs P and Q are constructed using the environment $\mathcal{E}$ and the names r available to the observer. The third clause is for outputs: the environment is updated with the processes emitted. Both in the input and in the output clause, the action from the processes is at a name that is known to the observer (condition $a \in r$). The fourth clause is new—it does not appear in the sequential calculi of the previous sections. It intuitively shows that the observer can run the values in the environment at any time. The fifth clause allows creation of fresh names by the observer. Relations $\approx_{\mathcal{E}\,;\,r}$ are extended to *open terms* using closing input abstractions. For instance if $\{x\} = \mathsf{fv}(M, N)$ then $M \;\; \approx_{\mathcal{E}\,;\,r} \;\; N$ if $a(x).\, M \;\; \approx_{\mathcal{E}\,;\,r,a} \;\; a(x).\, N$, for a fresh.

With reasoning similar to that for previous languages we prove that environmental bisimulation is an equivalence, and the soundness of a few basic up-to techniques: up-to environment, and up-to bisimilarity, which are similar to previous techniques, and up-to restriction, where in the conclusion a larger set of free names is allowed (i.e., $\mathcal{X}_{\mathcal{E}\,;\,r}$ is replaced by $\mathcal{X}_{\mathcal{E}\,;\,r,r'}$).

To establish congruence, we first prove that environmental bisimilarity is preserved by parallel composition (this is similar, but technically much more complex, to the results of congruence with respect to evaluation contexts in the λ-calculi of previous sections). We then use this result, together with a combination of the up-to techniques mentioned above, to derive congruence results for arbitrary contexts. We report the main result:

Theorem 5.2 *Relation* $\simeq$ *is a congruence relation.*

The congruence result can be used to establish the correspondence between environmental bisimilarity and contextual equivalence. In concurrency, confluence usually fails, and the branching structure in the reduction tree of a term becomes important. The definition of contextual equivalence has to be refined, adding a bisimulation clause on reductions. The resulting relation is called barbed congruence. We consider here the reduction-closed version of barbed congruence [11, 37]. *Reduction-closed barbed congruence*, $\equiv$, is the largest relation that is symmetric, reduction-closed (i.e., if $M \equiv N$, for M, N closed, and $M \xrightarrow{\tau} M'$, then $N \Longrightarrow N'$ and $M' \equiv N'$), context-closed (i.e., $M \equiv N$ implies $C[M] \equiv C[N]$, for all contexts), and barb preserving (i.e., if $M \equiv N$, for M, N closed, and $M \Downarrow_a$, then also $N \Downarrow_a$).

Theorem 5.3 *Relations* $\equiv$ *and* $\simeq$ *coincide.*

We have established the soundness of a few more sophisticated up-to techniques, notably techniques involving up-to contexts of the kind discussed in earlier sections. We report an example of such a technique in [34, Appendix E] (*environmental bisimulation up-to contexts, bisimilarity, and environment*).

6 Related Work

λ-calculi. Forms of bisimulation with an environment have been used (under no explicit name) by Sumii and Pierce for λ-calculi with perfect encryption [41] and data abstraction [42], inspired by bisimulations for typed π-calculus [7], polymorphic π-calculus [26] and spi-calculus [1]. However, their bisimulations were not able to handle higher-order functions. To address this issue, Sumii and Pierce [42, Secion 7] proposed a rather complex variant of their bisimulations with induction on the tree height of evaluation derivation, and an up-to-context technique built into the definition of bisimulations. Koutavas and Wand (KW in short) later gave a clearer account of this approach [16, 17]. KW has an induction on the evaluation of terms to values and an up-to context technique which are nicely hardwired into the definition of the bisimulation. However, KW relies on big-step semantics, which does not scale to languages with non-determinism or concurrency. Further KW breaks the monotonicity of the generating functional of the bisimulation (they have not only $\mathcal{E}$ but also $\mathcal{X}$ in a negative position because of the induction), and therefore requires extra proofs to guarantee that bisimilarity exists (and is transitive). Sumii and Pierce, as well as Koutavas and Wand, only gave indirect proofs of these properties, via the correspondence with contextual equivalence. For similar reasons, it could be difficult to enhance KW bisimulation method via up-to techniques (for instance, in [34, Example B.7] we make a non-trivial use of up-to expansion, which is not available in KW. Hence a proof of this equivalence with KW

requires an infinite relation, rather than a singleton relation as in our proof).

The bisimulation clause on functions of environmental bisimulation is reminiscent of logical relations; see, e.g., [28]. (The analogy is stronger for logical bisimulations, or for the BA-bisimulations discussed in the Introduction; we recall that in logical relations two functions are related if they map related arguments to related results.) However, logical relations represent a type-directed technique and as such remain quite different from bisimulations, which can be untyped. Logical relations work well in pure simply-typed or polymorphic λ-calculus, but they tend to become incomplete and/or require more advanced meta theory in languages with recursive types, existential types [28], encryption [40], store, or concurrency; see e.g. [3] for more references.

Concurrent languages. There are only a few concurrent higher-order languages for which bisimulation techniques have been given; usually the bisimilarity is either a form of higher-order bisimulation and Howe's technique [12] is used to prove congruence (e.g., [5, 8, 9]), or it is a form of context bisimulation or normal bisimulation (e.g., [14, 15, 20, 21, 31, 33]). Howe's technique appears to have limitations in concurrency. It seems be sensitive to the choice of the bisimilarity; in particular it gives problems if the bisimilarity is not both in the "delay" and in the "late" style; either style generally breaks the correspondence with contextual equivalence. (However recently Godskesen and Hildebrandt [9] have been able to handle a delayed input-early version of context bisimulation.) The technique also seems to be sensitive to the structure of terms that are allowed to interact (see [14] for a discussion).

In *context bisimulation* [9, 20, 21, 33], whenever an interaction is produced between the tested processes and the observer, all possible contexts that have originated the action from the observer are taken into account. For instance, clause (3) for output actions would become:

- if $M \xrightarrow{(\boldsymbol{\nu}\widetilde{b})\overline{a}P} M'$ then $N \xRightarrow{(\boldsymbol{\nu}\widetilde{c})\overline{a}Q} N'$,
  and for all G with $\mathsf{fv}(G) \subseteq X$,
  $\boldsymbol{\nu}\widetilde{b}\,(G\{^{P}/_{X}\} \mid M')\ \mathcal{R}\ \boldsymbol{\nu}\widetilde{c}\,(G\{^{Q}/_{X}\} \mid N')$

The quantification over all recipients G is a heavy demand, close to explicitly requiring that the bisimulation is preserved by parallel composition.

To simplify context bisimulation, *normal bisimulation* has been proposed ([31]; Jeffrey and Rathke [15] have then improved it). Normal bisimulation replaces the exchange of processes with the exchange of special processes called *triggers* and essentially implements a dynamic translation of higher-order communication into first-order, for a trigger that is communicated acts like a name-pointer to a process. Normal bisimulation does not explicitly use environments; these are hardwired inside the tested processes, by means of triggers. The main difference with environmental bisimulation is in the input clause: normal bisimulation does not use universal quantifications, and is therefore simpler. Possible advantages of environmental bisimulation are the following. First, it is straightforward to adapt environmental bisimulation to the strong case (where all transitions, including internal steps, are treated equally). Adapting normal bisimulation to the strong case is delicate because the transformation of process communications into trigger communications is valid for weak, but not for strong, equivalences. Second, the proof of congruence of environmental bisimulation is more direct. Related to this is the development of up-to techniques, in particular up-to contexts. Finally, normal bisimulation may rely on the possibility of encoding higher-order communications using triggers; this might not be possible, or might be difficult to achieve, in extensions of HOπ or in calculi different from HOπ.

To the best of our knowledge, the only higher-order concurrent calculus for which up-to-context techniques had been derived is the Ambient calculus [21], for a form of context bisimulation. Ambients however represent a special case of higher-order calculus, for processes can move but cannot be communicated. Moving a process is quite different from communicating it as in HOπ: in the former case the process will always be run, immediately and exactly once; in the latter case, in contrast, the process may be copied, and it is the recipient of the process that decides when and where to run each copy. Thus the problems of congruence of bisimulation, and the related problems for up-to contexts, only show up in a limited form in Ambients.

7 Conclusions and future work

In this paper we have developed the basic theory of environmental bisimulations. In summary, with environmental bisimulations we aim at (1) maintaining the definition of the bisimulation as simple as possible, so to facilitate proofs of its basic properties (in particular congruence and up-to-context techniques, which are notoriously hard in higher-order languages); and (2) separately developing enhancements of the bisimulation method, so as to have simple bisimilarity proofs between terms.

Although the basic idea of environmental bisimulation is the same on all calculi in the paper, the bisimilarity clauses can differ depending on the features of the calculus. It would be interesting to formulate environmental bisimulation in an abstract manner and to derive the concrete definitions in this paper as special instances of it. For this, Milner's bigraphs [23] would be a candidate framework.

It is encouraging that environmental bisimulation work on a variety of calculi (pure λ-calculi, λ-calculus with full-fledged store, Higher-Order π-calculus), and that the proof

of its basic properties, and associated up-to techniques, are easy to transport. To the best of our knowledge, none of the previous bisimilarities can handle such a variety of languages. In the future we plan to consider more sophisticated concurrent languages. For instance, the passivation construct of the Kell Calculus [38] appears challenging.

Acknowledgments. We are grateful to Vassileios Koutavas: discussions with him in the initial development of this work were helpful in clarifying concepts. We would like also to thank Kohei Suenaga for comments.

References

[1] M. Abadi and A. D. Gordon. A bisimulation method for cryptographic protocols. *ESOP*, 1998.

[2] S. Abramsky. The lazy lambda calculus. *Research Topics in Functional Programming*, Addison-Wesley, 1990.

[3] A. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. *ESOP*, 2006.

[4] A. W. Appel and D. McAllester. An indexed model of recursive types for foundational proof-carrying code. *TOPLAS*, 23(5):657–683, 2001.

[5] M. Baldamus and T. Frauenstein. Congruence proofs for weak bisimulation equivalences on higher–order process calculi, 1995. Rep. 95-21, Berlin University of Technology.

[6] L. Birkedal and R. Harper. Relational interpretations of recursive types in an operational setting. *Inf. and Comput.*, 155(1–2):3–63, 1999.

[7] M. Boreale and D. Sangiorgi. Bisimulation in name-passing calculi without matching. *LICS*, 1998.

[8] W. Ferreira, M. Hennessy, and A. Jeffrey. A theory of weak bisimulation for core CML. *J. Functional Programming*, 8(5):447–491, 1998.

[9] J. Godskesen and T. Hildebrandt. Extending Howe's method to early bisimulations for typed mobile embedded resources with local names. *FSTTCS*, LNCS 3821, Springer, 2005.

[10] A. D. Gordon. *Functional Programming and Input/Output*. PhD thesis, University of Cambridge, 1993.

[11] K. Honda and N. Yoshida. On reduction-based process semantics. *Theor. Comp. Sci.*, 152(2):437–486, 1995.

[12] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Inf. and Comput.*, 124(2):103–112, 1996.

[13] A. Jeffrey and J. Rathke. Towards a theory of bisimulation for local names. *LICS*, 1999.

[14] A. Jeffrey and J. Rathke. A theory of bisimulation for a fragment of concurrent ML with local names. *Theor. Comput. Sci.*, 323(1-3):1–48, 2004.

[15] A. Jeffrey and J. Rathke. Contextual equivalence for higher-order pi-calculus revisited. *Logical Methods in Computer Science*, 1(1:4):1–22, 2005.

[16] V. Koutavas and M. Wand. Bisimulations for untyped imperative objects. *ESOP*, 2006.

[17] V. Koutavas and M. Wand. Small bisimulations for reasoning about higher-order imperative programs. *POPL*, 2006.

[18] S. B. Lassen. Relational reasoning about contexts. *Higher Order Operational Techniques in Semantics*, Cambridge University Press, 1998.

[19] S. B. Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, Department of Computer Science, University of Aarhus, 1998.

[20] M. Merro and M. Hennessy. Bisimulation congruences in Safe Ambients. *POPL*, 2002.

[21] M. Merro and F. Z. Nardelli. Behavioral theory for mobile ambients. *J. ACM*, 52(6):961–1023, 2005.

[22] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[23] R. Milner. Pure bigraphs: Structure and dynamics. *Inf. and Comput.*, 204(1):60–122, 2006.

[24] J. H. Morris, Jr. *Lambda-Calculus Models of Programming Languages*. PhD thesis, MIT, 1968.

[25] L. Ong. *The Lazy Lambda Calculus: an Investigation into the Foundations of Functional Programming*. PhD thesis, University of London, 1988.

[26] B. C. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. *J. ACM*, 47(3):531–586, 2000.

[27] A. Pitts. Operationally-based theories of program equivalence. *Semantics and Logics of Computation*, Cambridge University Press, 1997.

[28] A. Pitts. Typed operational reasoning. *Advanced Topics in Types and Programming Languages*, MIT Press, 2005.

[29] J. C. Reynolds. Types, abstraction and parametric polymorphism. *Information Processing*, 1983.

[30] D. Sands. Improvement theory and its applications. *Higher Order Operational Techniques in Semantics*, Cambridge University Press, 1998.

[31] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigm*. PhD thesis, University of Edinburgh, 1992.

[32] D. Sangiorgi. The Lazy Lambda Calculus in a Concurrency Scenario. *Inf. and Comput.*, 111(1):120–153, 1994.

[33] D. Sangiorgi. Bisimulation for Higher-Order Process Calculi. *Inf. and Comput.*, 131(2):141–178, 1996.

[34] D. Sangiorgi, N. Kobayashi, and E. Sumii Appendixes to the paper "Environmental Bisimulations for Higher-Order Languages." `http://www.cs.unibo.it/~sangio/DOC_public/appLICS07.pdf`, 2007.

[35] D. Sangiorgi, N. Kobayashi, and E. Sumii. Logical Bisimulations for Higher-Order Languages, Forthcoming, 2007.

[36] D. Sangiorgi and R. Milner. The problem of "Weak Bisimulation up to". *Proc. CONCUR*, LNCS 630, Springer, 1992.

[37] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[38] A. Schmitt and J.-B. Stefani. The Kell Calculus: A family of higher-order distributed process calculi. *Global Computing*, LNCS 3267, Springer, 2004.

[39] K. Støvring and S. Lassen. A complete, co-inductive syntactic theory of sequential control and state. *POPL*, 2007.

[40] E. Sumii and B. C. Pierce. Logical relations for encryption. *J. Computer Security*, 11(4):521–554, 2003.

[41] E. Sumii and B. C. Pierce. A bisimulation for dynamic sealing. *Theor. Comput. Sci.*, 375(1–3):169–192, 2007.

[42] E. Sumii and B. C. Pierce. A bisimulation for type abstraction and recursion. *POPL*, 2005.

[43] B. Thomsen. Plain CHOCS, a second generation calculus for higher-order processes. *Acta Inf.*, 30:1–59, 1993.