Logical Bisimulations and functional languages*

Davide Sangiorgi[1], Naoki Kobayashi[2], and Eijiro Sumii[2]

[1] University of Bologna, Italy
[2] Tohoku University, Japan

Abstract. Developing a theory of bisimulation in higher-order languages can be hard. Particularly challenging can be the proof of congruence and, related to this, enhancements of the bisimulation proof method with "up-to context" techniques.
We present *logical bisimulations*, a form of bisimulation for higher-order languages, in which the bisimulation clause is somehow reminiscent of logical relations. We consider purely functional languages, in particular untyped call-by-name and call-by-value lambda-calculi, and, in each case: we present the basic properties of logical 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.

1 Introduction

Applicative bisimulations and behavioral equivalence in higher-order languages. Equivalence proof 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 [20]. Finding effective methods for equivalence proofs is particularly challenging in higher-order languages (i.e., languages where program code can be passed around like other data).

Bisimulation has emerged as a very 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*. Bisimulation has been transplanted onto (sequential) higher-order languages by Abramsky [1]. This version of bisimulation, called *applicative bisimulations*, and variants of it, have received considerable attention [11, 12, 22, 24, 16]. In short, two functions P and Q are applicatively bisimilar when their applications $P(M)$ and $Q(M)$ are applicatively bisimilar for any argument M.

Applicative bisimulations have two significant limitations. First, they do not scale very well to languages richer than pure λ-calculus. For instance, they are unsound under the presence of generative names [14] or data abstraction [28]

* Sangiorgi's research was partially supported by european FET project SENSORIA and italian MIUR Project n. 2005015785, "Logical Foundations of Distributed Systems and Mobile Code".

because they apply bisimilar functions to an *identical* argument. Secondly, congruence proofs of applicative bisimulations are notoriously hard. Such proofs usually rely on Howe's method [13]. The method appears however rather subtle and fragile, for instance under the presence of generative names [14], nondeterminism [13], or concurrency (e.g., [9, 10]). Also, the method is very syntactical and lacks good intuition about when and why it works. Related to the problems with congruence are also the difficulties of applicative bisimulations with "up-to context" techniques (the usefulness of these techniques in higher-order languages and its problems with applicative bisimulations have been extensively studied by Lassen [16]; see also [24, 15]).

Congruence proofs for bisimulations usually exploit the bisimulation method itself to establish that the closure of the bisimilarity under contexts is again a bisimulation. To see why, intuitively, this proof does not work for applicative bisimulation, consider a pair of bisimilar functions P_1, Q_1 and another pair of bisimilar terms P_2, Q_2. In an application context they yield the terms P_1P_2 and Q_1Q_2 which, if bisimilarity is a congruence, should be bisimilar. However the arguments for the functions P_1 and Q_1 are bisimilar, but not necessarily identical: hence we are unable to apply the bisimulation hypothesis on the functions.

The above congruence argument would work if the bisimulation were required to apply bisimilar functions to *bisimilar* arguments. This definition of bisimulation, that in this discussion we call BA-bisimulation[3], 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 pair) according to the definition above.

Logical bisimulations. In this paper we investigate a different approach to defining bisimilarity on functions. The main feature of our bisimulations, that we call *logical bisimulations*, is to apply related functions (i.e., functions in the bisimulation relation) P and Q to arguments in the context closure of the bisimulation, that is, arguments of the forms $C[V_1, \ldots, V_n]$ and $C[W_1, \ldots, W_n]$ for a context C and related values $(V_1, W_1), \ldots, (V_n, W_n)$. Thus the arguments can be identical terms, as for applicative bisimilarity, or related terms, as in BA-bisimulation, or combinations of these. As in BA-bisimulation, so in logical bisimulations the generating functional is non-monotone. However, as in applicative bisimilarity—and in contrast with BA-bisimulations—logical bisimulations are sound and the corresponding functional has a greatest fixed-point which coincides with contextual equivalence.

The intuition behind the bisimulation requirement of logical bisimulations is the following. Consider an observer that is playing the bisimulation game,

[3] BA indicates that the bisimilarity uses "Bisimilar Arguments"

testing related terms. Values produced by related terms are like outputs towards the observer, who can use them at will: they have become part of the observer's *knowledge*. Thus the observer can check the consistency of such values (for instance, the outermost construct should be same). In addition, however, the observer can use them to build more complex terms (such as $C[V_1, \ldots, V_n]$ and $C[W_1, \ldots, W_n]$ above) and use them as arguments when testing pairs of related functions. Of course this power is useless if the values are first-order, since related values must then be identical. But it is relevant in a higher-order language and yields the bisimulation requirement described above.

A possible drawback of logical bisimulations over applicative bisimulations is that the set of arguments to related functions that have to be considered in the bisimulation clause is larger (since it includes also non-identical arguments). As a remedy to this, we propose the use of up-to techniques, as enhancements to the bisimulation proof method. We consider a number of such enhancements in the paper, including forms of up-to context and up-to expansion.

Another difference of logical bisimulations over applicative bisimulations (as well as most definitions of bisimulation for functions in the literature) is that we use a small-step, rather than big-step, semantics. For this reason, logical bisimulations are defined on arbitrary closed terms, rather than values. The use of small-step semantics may seem cumbersome—in particular for languages without non-determinism—because it seems to require more elements in bisimulations than big-step semantics. However, again, this disadvantage disappears by means of up-to techniques. In fact, the extension to small-step semantics often *simplifies* an equivalence proof, because we can now compare terms in the middle of evaluations without reducing them to values. Further, big-step versions of logical bisimulations will be derived as a corollary of the soundness of certain up-to techniques (precisely "up-to reduction"). Another reason for choosing a small-step semantics is that this is often required for non-determinism or concurrency.

In summary, with logical 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.

The bisimulation clause on functions of logical bisimulations is somehow reminiscent of logical relations, see, e.g., [19, Chapter 8] and [23]. (The analogy is stronger for the BA-bisimulations discussed earlier; 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 [5, 8, 4, 2], existential types [23, 8, 3, 2], store [3], or encryption [27], to give just a few examples.

The idea of logical bisimulations stems from bisimulations for higher-order calculi with information hiding mechanisms (such as encryption [29], data abstraction [28], and store [15]), where the use of context closures of function arguments was *necessary* because of the information hiding. In this respect, our contribution in this paper is to isolate this idea and propose it as a general method for higher-order languages. Moreover, we simplify and strengthen the method and develop its basic theory.

In this paper we consider purely functional languages, in particular untyped call-by-name and call-by-value lambda-calculi. It seems difficult to adapt logical bisimulation, at least in the form presented here, to non-functional languages; for instance, languages with information hiding constructs (e.g., for store, encryption, data abstraction) or with parallelism. To treat these languages we have added an explicit notion of environment to the bisimulations. The technical details become rather different, and can be found in [26].

2 Preliminaries

In this section, we introduce general notations and terminologies used throughout the paper. Familiarity with standard terminologies (such as free/bound variables, and α-conversion) for the λ-calculus is assumed.

We use meta-variables $M, N, P, Q, \ldots$ for terms, and $V, W, \ldots$ for values (in untyped λ-calculus the only closed values are the abstractions). 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. Note that a context may contain no holes, and therefore any term is a context. A context may bind variables in $M_1, \ldots, M_n$; for example, if $C = \lambda x.\, [\cdot]_1$ and $M = x$, then $C[M]$ is $\lambda x.\, x$, not $\lambda y.\, x$. The set Λ of λ-terms is defined by:

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

We write $\Lambda^\bullet$ for the subset of closed terms.

We use meta-variables $\mathcal{R}, \mathcal{S}, \ldots$ 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.

$$\{(C[M_1, \ldots, M_n], C[N_1, \ldots, N_n]) \;\mid\; M_i \mathcal{R} N_i \text{ for each } i\}$$

By definition $\mathcal{R}^\star$ 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 often 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 in the remainder of this paper:

– Although the results are often stated for closed values only, they can be generalized to open terms in a common way. 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 [29] and Koutavas-Wand [15]. (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 $\{\widetilde{x}\} \supseteq \mathsf{fv}(M) \cup \mathsf{fv}(N)$.
– The results in this paper are stated for untyped languages. Adapting them to languages with a simply-typed discipline is straightforward. (We will use a simply-typed calculus in an example.)

3 Call-by-name λ-calculus

The *call-by-name reduction relation* $\longrightarrow$ is the least relation over $\Lambda^\bullet$ closed under the following rules.

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

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

3.1 Logical bisimulations

If $\mathcal{R}$ is a relation on closed terms, then we extend it to open terms thus: if $\mathsf{fv}(M, N) = \{\widetilde{x}\}$, then $M\ \mathcal{R}^o\ N$ holds if for all $\widetilde{M}, \widetilde{N} \in \Lambda^\bullet$ with $\widetilde{M}\ \mathcal{R}^\star\ \widetilde{N}$ we have $M\{\widetilde{M}/\widetilde{x}\}\ \mathcal{R}\ N\{\widetilde{N}/\widetilde{x}\}$.

Definition 1 (logical bisimulation). *A relation $\mathcal{R} \subseteq \Lambda^\bullet \times \Lambda^\bullet$ is a* logical bisimulation *if whenever $M\ \mathcal{R}\ N$,*

1. *if $M \longrightarrow M'$ then $N \Longrightarrow N'$ and $M'\ \mathcal{R}\ N'$;*
2. *if $M = \lambda x.\, M'$ then $N \Longrightarrow \lambda x.\, N'$ and $M'\ \mathcal{R}^o\ N'$;*
3. *the converse of (1) and (2) above, on N.*

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

As $\mathcal{R}$ occurs in negative position in the definition of logical bisimulation, the existence of the *largest* bisimulation is unclear. Indeed the union of two logical bisimulations is not necessarily a logical bisimulation. We however prove below that $\approx$ itself is a bisimulation, so that it is also the largest bisimulation. We often omit "logical" in the remainder of the paper.

Remark 1. The negative occurrence of $\mathcal{R}$ in the definition of logical bisimulation breaks the monotonicity of the generating functional (the function from relations to relations that represents the clauses of bisimulation). Therefore we cannot appeal to the Knaster-Tarski's fixed point theorem for the existence of a largest bisimulation. (Such a theorem guarantees the existence of the greatest fixed point for a monotone function on a complete lattice; moreover this point coincides with the greatest post-fixed point of the function; see [25] for discussions on the theorem and on coinduction). Thus, if we take Knaster-Tarski as the justification of coinduction, then we could not call *coinductive* the proof method for logical bisimulations. However we can show that the largest logical bisimulation exists, and therefore the proof method given by logical bisimulations is sound and complete. We call the method coinductive because it has the form of standard coinductive proof methods. We thus take coinduction with a meaning broader than that given by Knaster-Tarski's theorem, namely as a notion for reasoning about functions on complete lattices that have a greatest post-fixed point.

First we prove that $\approx$ is an equivalence relation; the only non-trivial case is transitivity.

Lemma 1. *Suppose $\mathcal{R}$ is a bisimulation, $M\ \mathcal{R}\ N$, and $M \Longrightarrow M'$. Then there is N' such that $N \Longrightarrow N'$ and $M'\ \mathcal{R}\ N'$.*

Proof. Induction on the length of $M \Longrightarrow M'$.

Lemma 2. *Suppose $\mathcal{R}_1$ and $\mathcal{R}_2$ are bisimulations. Then also $\mathcal{R}_1\,\mathcal{R}_2$ (the relational composition between them) is a bisimulation.*

Proof. We prove that $\mathcal{R}_1\,\mathcal{R}_2$ is a bisimulation. As an example, consider clause (2) of the bisimulation. Thus, suppose $M\ \mathcal{R}_1\,\mathcal{R}_2\ N$ because $M\ \mathcal{R}_1\ L\ \mathcal{R}_2\ N$, and $M = \lambda x.\, M'$.

Since $\mathcal{R}_1$ is a bisimulation, there is L' such that $L \Longrightarrow \lambda x.\, L'$ and $M'\ \mathcal{R}_1^{\circ}\ L'$. Using Lemma 1, since also $\mathcal{R}_2$ is a bisimulation, there is N' such that $N \Longrightarrow \lambda x.\, N'$ and $L'\ \mathcal{R}_2^{\circ}\ N'$.

We have to prove that for all $(M_1, N_1) \in (\mathcal{R}_1\,\mathcal{R}_2)^{\star}$, we have $M'\{^{M_1}/_x\}\ \mathcal{R}_1\,\mathcal{R}_2\ N'\{^{N_1}/_x\}$. If $(M_1, N_1) \in (\mathcal{R}_1\,\mathcal{R}_2)^{\star}$, then there is a context C and terms $\widetilde{M_1'}, \widetilde{N_1'}$ with $\widetilde{M_1'}\ \mathcal{R}_1\,\mathcal{R}_2\ \widetilde{N_1'}$ such that $M_1 = C[\widetilde{M_1'}]$ and $N_1 = C[\widetilde{N_1'}]$. By definition of relational composition, there are $\widetilde{L_1'}$ such that $\widetilde{M_1'}\mathcal{R}_1\widetilde{L_1'}\mathcal{R}_2\widetilde{N_1'}$. Hence, since $\mathcal{R}_1$ and $\mathcal{R}_2$ are bisimulations, we have

$$M'\{^{C[\widetilde{M_1'}]}/_x\}\ \mathcal{R}_1\ L'\{^{C[\widetilde{L_1'}]}/_x\} \text{ and } L'\{^{C[\widetilde{L_1'}]}/_x\}\ \mathcal{R}_2\ N'\{^{C[\widetilde{N_1'}]}/_x\}.$$

We can therefore conclude $M'\{^{C[\widetilde{M_1'}]}/_x\}\ \mathcal{R}_1\,\mathcal{R}_2\ N'\{^{C[\widetilde{N_1'}]}/_x\}$.

Next we prove that $\approx$ is preserved by contexts, which allows us to conclude that $\approx$ is a congruence relation. In bisimilarities for higher-order languages, the

congruence properties are usually the most delicate basic properties to establish. In contrast with proofs for applicative bisimilarity, which usually involve sophisticated techniques such as Howe's, for logical bisimilarity simple inductive reasoning on contexts suffices.

Lemma 3. *If $\mathcal{R}$ is a bisimulation, then also $\mathcal{R}^\star$ is a bisimulation.*

Proof. We prove that $\mathcal{R}^\star$ is a bisimulation. Suppose $(C[\widetilde{M}], C[\widetilde{N}]) \in \mathcal{R}^\star$ with $\widetilde{M}\ \mathcal{R}\ \widetilde{N}$. We prove clauses (1) and (2) of the bisimulation by induction on the size of C. There are three cases to consider.

The case $C = [\cdot]_i$ is immediate, using the fact that $(\mathcal{R}^\star)^\star = \mathcal{R}^\star$.

In the case $C = \lambda x.\, C'$, only clause (2) of bisimulation applies: let M_1, N_1 be the arguments of the functions, with $M_1\ \mathcal{R}^\star\ N_1$; we have also $C'[\widetilde{M}]\{^{M_1}/_x\}\ \mathcal{R}^\star\ C'[\widetilde{N}]\{^{N_1}/_x\}$, and we are done.

It remains the case $C = C_1 C_2$, where only clause (1) of bisimulation applies. There are two possibilities of reduction for $C_1[\widetilde{M}]C_2[\widetilde{M}]$: the left-hand side $C_1[\widetilde{M}]$ reduces alone; the left-hand side is a function, say $\lambda x.\, P$, and the final derivative is $P\{^{C_2[\widetilde{M}]}/_y\}$. The first possibility is dealt with using induction. In the second one, by the induction hypothesis, we infer: $C_1[\widetilde{N}] \Longrightarrow \lambda y.\, Q$ and $P\ (\mathcal{R}^\star)^{\circ}\ Q$. Hence $P\{^{C_2[\widetilde{M}]}/_y\}\mathcal{R}^\star Q\{^{C_2[\widetilde{N}]}/_y\}$, and we are done.

Corollary 1. *$\approx$ is a congruence relation.*

Finally, we prove that $\approx$ itself is a bisimulation, exploiting the previous results.

Lemma 4. *$\approx$ is a bisimulation.*

Proof. In the proof that $\approx$ is a bisimulation, clause (1) of Definition 1 is straightforward to handle.

We consider clause (2). Thus, suppose $\lambda x.\, M \approx N$. By definition of $\approx$, there is a bisimulation $\mathcal{R}$ such that $\lambda x.\, M\ \mathcal{R}\ N$; hence there is N' such that $N \Longrightarrow \lambda x.\, N'$ and $M\ \mathcal{R}^{\circ}\ N'$. We have to prove that also $M \approx^{\circ} N'$ holds.

Take $M_1 \approx^\star N_1$; we want to show $M\{^{M_1}/_x\} \approx N'\{^{N_1}/_x\}$. If $M_1 \approx^\star N_1$, then there is a context C and terms $M'_1, \ldots, M'_n, N'_1, \ldots, N'_n$ with $M'_i\ \mathcal{S}_i\ N'_i$ for some bisimulation S_i such that $M_1 = C[M'_1, \ldots, M'_n]$ and $N_1 = C[N'_1, \ldots, N'_n]$. We have:

$$
\begin{array}{lll}
M\{^{C[M'_1,\ldots,M'_n]}/_x\} & \mathcal{R}\ N'\{^{C[M'_1,\ldots,M'_n]}/_x\} & \text{(since } M\ \mathcal{R}^{\circ}\ N'\text{)} \\
& \mathcal{S}_1^\star\ N'\{^{C[N'_1,M'_2,\ldots,M'_n]}/_x\} & \\
& \ldots & \\
& \mathcal{S}_n^\star\ N'\{^{C[N'_1,,\ldots,N'_{n-1}N'_n]}/_x\} &
\end{array}
$$

This closes the proof, because each $\mathcal{S}_i^\star$ is a bisimulation (Lemma 3) and because bisimulations are closed under composition (Lemma 2).

Example 1. We have $I_1 \approx I_2$ for $I_1 \stackrel{\text{def}}{=} \lambda x.\, x$ and $I_2 \stackrel{\text{def}}{=} \lambda x.\, (\lambda y.\, y)x$, by taking $\mathcal{R}$ to be the least pre-congruence such that $I_1 \;\mathcal{R}\; I_2$ and, if $M \;\mathcal{R}\; N$, then $M \;\mathcal{R}\; (\lambda y.\, y)N$. Note that the singleton relation $\{(I_1, I_2)\}$ by itself is not a logical bisimulation because of the implicit use of $\mathcal{R}^\star$ in clause (2) of bisimulation. Burdens like this are frequent in bisimulation proofs, and will be removed by the up-to techniques described later in this section. Specifically, the singleton relation $\{(I_1, I_2)\}$ will be a logical bisimulation "up to reduction and contexts".

3.2 Up-to techniques

We show a few "up-to" techniques, as enhancements of the bisimulation proof method. They allow us to prove bisimulation results using relations that in general are not themselves bisimulations, but are contained in a bisimulation. Rather than presenting complete definitions, we indicate the modifications to the bisimulation clauses (Definition 1). For this, it is however convenient to expand the abbreviation $M' \;\mathcal{R}^{\circ}\; N'$ in clause (2) of the definition, which thus becomes "for all $(M_1, N_1) \in \mathcal{R}^\star$ it holds that $M'\{^{M_1}/_x\} \;\mathcal{R}\; N'\{^{N_1}/_x\}$", and to describe the modifications with respect to this expanded clause.

We also omit the statements of soundness of the techniques.

Up-to bisimilarity. This technique introduces a (limited) use of $\approx$ on tested terms. This can allow us to avoid bisimulations with elements that, behaviorally, are the same. In clause (1), we replace "$M' \;\mathcal{R}\; N'$" with "$M' \;\mathcal{R} \approx\; N'$"; in (2), we replace "$M'\{^{M_1}/_x\} \;\mathcal{R}\; N'\{^{N_1}/_x\}$" with "$M'\{^{M_1}/_x\} \;\approx\; \mathcal{R} \;\approx\; N'\{^{N_1}/_x\}$". We cannot strengthen up-to bisimilarity by using $\approx$ also on the left-hand side of $\mathcal{R}$ in clause (1), for the technique would be unsound; this is reminiscent of the problems of up-to bisimilarity in standard small-step bisimilarity for concurrency. [18].

Up-to reduction. 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), we replace "$M' \;\mathcal{R}\; N'$" with "there are M'', N'' with $M' \Longrightarrow M''$ and $N' \Longrightarrow N''$ such that $M'' \;\mathcal{R}\; N''$"; similarly, in (2) we replace "$M'\{^{M_1}/_x\} \;\mathcal{R}\; N'\{^{N_1}/_x\}$" with "there are M'', N'' with $M'\{^{M_1}/_x\} \Longrightarrow M''$ and $N'\{^{N_1}/_x\} \Longrightarrow N''$ such that $M'' \;\mathcal{R}\; N''$".

The technique allows us to derive the soundness of the "big-step" version of logical bisimulation, in which clauses (1) and (2) are unified by requiring that

- if $M \Longrightarrow \lambda x.\, M'$ then $N \Longrightarrow \lambda x.\, N'$ and $M' \;\mathcal{R}^{\circ}\; N'$.

Up-to expansion. In concurrency, a useful auxiliary relation for up-to techniques is the *expansion relation.* (A similar relation is Sands' improvement for functional languages [24]). We adapt here the concept of expansion to the λ-calculus. We

write $M \Longrightarrow_n M'$ if M reduces to M' in n steps. We present the big-step version of expansion, since we will use it in examples. As for bisimilarity, so for expansion the small-step version is equally possible. Similarly, the up-to techniques described for bisimilarity can also be used to enhance expansion proofs, and then the big-step version of expansion below can be derived from the small-step version plus a "weighted" version of up-to reduction.

Definition 2. *A relation $\mathcal{R}$ is an* expansion relation *if whenever $M\ \mathcal{R}\ N$,*

1. *$M \Longrightarrow_m \lambda x.\, M'$ implies $N \Longrightarrow_n \lambda x.\, N'$ with $m \leq n$, and $M'\ \mathcal{R}^{\circ}\ N'$;*
2. *The converse, i.e., $N \Longrightarrow_n \lambda x.\, N'$ implies $M \Longrightarrow_m \lambda x.\, M'$ with $m \leq n$ and $M'\ \mathcal{R}^{\circ}\ N'$;*

Expansion*, written $\preceq$, is the union of all expansion relations.*

Thus if $M \longrightarrow M'$ then $M \succeq M'$ holds, but not necessarily $M' \succeq M$.

Lemma 5. *$\preceq$ is a pre-congruence and is an expansion relation itself.*

Proof. Similar to the proofs for $\approx$.

In the *bisimulation up-to expansion* technique, in Definition 1, we replace the occurrence of $\mathcal{R}$ in clause (1), and that in clause (2), with $\succeq \mathcal{R} \preceq$.

Since $\longrightarrow \subseteq \preceq$, the up-to expansion technique subsumes, and is more powerful than, up-to reduction. Still, up-to reduction is interesting because it can be simpler to combine with other techniques and to adapt to richer languages.

Up-to values. Using up-to expansion, and exploiting the basic properties of expansion (notably pre-congruence, and the fact that any pair of closed divergent terms is in the expansion relation) we can prove that the quantification over $\mathcal{R}^{\star}$ in clause (2) can be restricted to $\widehat{\mathcal{R}}^{\star}$, where $\widehat{\mathcal{R}}$ indicates the subset of $\mathcal{R}$ with only pairs of values.

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) and (2) the final occurrence of $\mathcal{R}$ is replaced by $R^{\star}$.

Up-to full contexts. The difference between "up-to contexts" and "up-to full contexts" is that in the latter the contexts that are cancelled can also bind variables of the arguments. As a consequence, however, a relation for the "up-to full contexts" is on open terms. Clauses (1) and (2) of Definition 1 are used only on closed terms, but with the last occurrence of $\mathcal{R}$ in each clause replaced by $\mathcal{R}^{\star}$. We add a new clause for open terms:

- If $M\ \mathcal{R}\ N$ then also $M\ \mathcal{R}^{\circ}\ N$ (i.e., if $\widetilde{x} = \mathsf{fv}(M, N)$, then for all $(\widetilde{M_1}, \widetilde{N_1}) \in \mathcal{R}^{\star}$, it holds that $M\{\widetilde{M_1}/\widetilde{x}\}\ \mathcal{R}^{\star}\ N\{\widetilde{N_1}/\widetilde{x}\}$).

Again, the up-to full contexts subsumes, and is more powerful than, up-to contexts, but the latter is simpler to establish and use.

Remark 2. An up-to-full-contexts technique similar to the one above has been proposed by Lassen [17, Lemma 7] and proved sound with respect to applicative bisimilarity. (Lassen was actually hoping to prove the soundness of the up-to-full-contexts technique for applicative bisimilarity itself, but failed; indeed forms of up-to contexts for applicative bisimilarities are notoriously hard). Further, Lassen's paper contains a number of interesting examples, such as least-fixed point properties of recursion and a syntactic minimal invariance property, that are proved for applicative bisimilarity by making use of up-to techniques. Similar proofs can be given for logical bisimilarity.

Big-step versions and combinations of up-to. The previous techniques can be combined together, in the expected manner. Further, for each technique both the small-step and the big-step versions are possible. We give two examples. The (small-step) "*up-to expansion and full contexts*" is defined as "up-to full contexts", but expansion appears in the conclusions. Thus clause (1) becomes:

- if $M, N \in \Lambda^\bullet$ and $M \longrightarrow M'$ then $N \Longrightarrow N'$ and $M' \succeq \mathcal{R}^\star \preceq N'$

Clause (2) is modified similarly; and in the clause for open terms, "$M\{\widetilde{M_1}/\widetilde{x}\}\ \mathcal{R}^\star\ N\{\widetilde{N_1}/\widetilde{x}\}$" is replaced by "$M\{\widetilde{M_1}/\widetilde{x}\} \succeq \mathcal{R}^\star \preceq N\{\widetilde{N_1}/\widetilde{x}\}$".

In the *big-step up-to expansion and context*, the bisimulation clause becomes:

- if $M \Longrightarrow \lambda x.\, M'$ then $N \Longrightarrow \lambda x.\, N'$ and for all $(M_1, N_1) \in \mathcal{R}^\star$ it holds that $M'\{M_1/x\} \succeq \mathcal{R}^\star \preceq N'\{N_1/x\}$. (*)

Of course, in general the more powerful the up-to is, the more work is required in its proof of soundness.

3.3 Contextual equivalence

Definition 3 (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 1 (soundness and completeness of bisimulation). *Relations $\equiv$ and $\approx$ coincide.*

Proof. For closed terms, we prove that $M \equiv N$ implies $M \approx N$ by showing that $\equiv$ is a bisimulation; the proof is simple, proving first that $\equiv$ is an equivalence, that $\equiv^\star\ = \equiv$ and that reduction is included in $\equiv$. The converse implication ($M \approx N$ implies $M \equiv N$) immediately follows from the congruence of $\approx$. The result for open terms is obtained as discussed in Section 2.

3.4 Example 1

This example gives the proof of the equivalence between the two fixed-point combinators:

$$Y \stackrel{\text{def}}{=} \lambda y.\, y(Dy(Dy))$$
$$\Theta \stackrel{\text{def}}{=} \Delta\Delta$$

where

$$\Delta \stackrel{\text{def}}{=} \lambda x.\, \lambda y.\, (y(xxy))$$
$$D \stackrel{\text{def}}{=} \lambda y.\, \lambda x.\, y(xx)$$

We establish $Y \approx \Theta$ using a relation $\mathcal{R}$ that has just one pair, namely (Y, Θ), and proving that $\mathcal{R}$ is a big-step logical bisimulation up to expansion and context. First, we note that, for any term M,

$$DM(DM) \succeq YM \tag{1}$$

This holds because $DM(DM) \Longrightarrow_2 M(DM(DM))$ and $YM \Longrightarrow_1 M(DM(DM))$. We now check the bisimilarity clause (*) on the pair (Y, Θ). Term Y is a function; the other term, Θ, becomes a function as follows:

$$\Theta \longrightarrow \lambda y.\, (y(\Delta\Delta y)) \stackrel{\text{def}}{=} \Theta_1$$

Consider now any argument $M\ \mathcal{R}^\star\ N$ for Y and Θ_1. The results are $M(DM(DM))$ and $N(\Delta\Delta N)$, respectively. Now, by (1), it holds that

$$M(DM(DM)) \succeq M(YM)$$

and we are done, since $M(YM)\ \mathcal{R}^\star\ N(\Delta\Delta N) = N(\Theta N)$.

4 Call-by-value λ-calculus

The *one-step call-by-value reduction relation* $\longrightarrow \subseteq \Lambda^\bullet \times \Lambda^\bullet$ is defined by these rules:

$$\beta_{\mathrm{v}}: \quad (\lambda x.\, M)V \longrightarrow M\{^V/_x\}$$

$$\mu: \quad \frac{M \longrightarrow M'}{MN \longrightarrow M'N} \qquad \nu_{\mathrm{v}}: \quad \frac{N \longrightarrow N'}{VN \longrightarrow VN'}$$

We highlight what changes in the theory for call-by-name of the previous sections. For a relation $\mathcal{R}$ we write $\mathcal{R}^{\widehat{\star}}$ for the subset of $\mathcal{R}^\star$ that only relate pairs of values.

- The input for two functions must be values. Therefore, in the definition of bisimulation, the input terms M_1 and N_1 should be in $\mathcal{R}^{\widehat{\star}}$ (rather than $\mathcal{R}^\star$). A similar modification on the quantification over inputs of functions is needed in all definitions of bisimulations and up-to techniques.
- In clause (2) of bisimilarity we add the requirement that the two functions themselves are related, i.e., $\lambda x.\, M'\ \mathcal{R}\ \lambda x.\, N'$. Roughly, this is needed because, in call-by-value, by definition, function arguments are evaluated before applications. The proof of congruence itself for bisimilarity requires this addition. We will nevertheless be able to remove the requirement later, exploiting appropriate up-to techniques.

Remark 3. To make the definition of logical bisimulation uniform for call-by-name and call-by-value, the requirement "$\lambda x.\, M' \;\mathcal{R}\; \lambda x.\, N'$" could also be added in call-by-name. This would not affect the proofs of the result presented. As in call-by-value, the requirement could then be removed by means of appropriate up-to techniques.

For ease of reference, we report the complete definition of bisimulation. If $\mathcal{R}$ is a relation on closed terms, and $\mathsf{fv}(M, N) = \widetilde{x}$, then $M\ \mathcal{R}^{\widehat{\circ}}\ N$ holds if for all $\widetilde{V}, \widetilde{W}$ with $\widetilde{V}\ \mathcal{R}^{\widehat{\star}}\ \widetilde{W}$ it holds that $M\{\widetilde{V}/\widetilde{x}\}\ \mathcal{R}\ N\{\widetilde{W}/\widetilde{x}\}$.

Definition 4. *A relation $\mathcal{R} \subseteq \Lambda^{\bullet} \times \Lambda^{\bullet}$ is a* logical bisimulation *if whenever $M\ \mathcal{R}\ N$,*

1. *if $M \longrightarrow M'$ then $N \Longrightarrow N'$ and $M'\ \mathcal{R}\ N'$*
2. *if $M = \lambda x.\, M'$ then $N \Longrightarrow \lambda x.\, N'$ and*
   (a) $\lambda x.\, M'\ \mathcal{R}\ \lambda x.\, N'$
   (b) $M'\ \mathcal{R}^{\widehat{\circ}}\ N'$
3. *the converse of (1) and (2) above.*

With these modifications, all definitions and results in Section 3 are valid for call-by-value. The structure of the proof also remains the same, with the expected differences in technical details due to the change in reduction strategy. It is however worth revisiting the proof of Lemma 3; although the structure of the proof is the same, the few differences are important, in particular to understand the requirement (2.a) in Definition 4.

Lemma 6. *If $\mathcal{R}$ is a bisimulation, then also $\mathcal{R}^{\star}$ is a bisimulation.*

Proof. As before we prove that $\mathcal{R}^{\star}$ is a bisimulation reasoning by induction on the size of the common contexts of terms $(C[\widetilde{M}], C[\widetilde{N}]) \in \mathcal{R}^{\star}$ with $\widetilde{M}\ \mathcal{R}\ \widetilde{N}$. In the case $C = [\cdot]_i$ we use the fact that $(\mathcal{R}^{\star})^{\widehat{\star}} = \mathcal{R}^{\widehat{\star}}$.

The interesting case is $C = C_1 C_2$ when both $C_1[\widetilde{M}]$ and $C_2[\widetilde{M}]$ are values, say $\lambda x.\, P$ and V, respectively. By the induction hypothesis, we infer:

$$C_2[\widetilde{N}] \Longrightarrow W,$$

for some W with $V\ \mathcal{R}^{\widehat{\star}}\ W$. (Note that here we exploit the requirement (2.a) of Definition 4.) Similarly we infer $C_1[\widetilde{N}] \Longrightarrow \lambda x.\, Q$, for some Q with $P\ (\mathcal{R}^{\star})^{\widehat{\circ}}\ Q$. This implies, since $V\ \mathcal{R}^{\widehat{\star}}\ W$, that $P\{V/x\}\ \mathcal{R}^{\star}\ Q\{W/x\}$.

4.1 Up-to techniques

All up-to techniques described for call-by-name are valid also for call-by-value, modulo the technical differences in definitions that we have discussed in the previous subsection. In addition, however, we can also derive the soundness (and completeness) of a form of logical bisimulation with big-step restricted to values (in call-by-value, applicative bisimulation is normally defined this way) and that we call *value big-step logical bisimulation.*

Definition 5. *A relation $\mathcal{E}$ on closed values is a* value big-step logical bisimulation *if for all $V\ \mathcal{E}\ W$ and $V_1\ \hat{\mathcal{E}^\star}\ W_1$, if $VV_1 \Longrightarrow V'$ then there is W' such that $WW_1 \Longrightarrow W'$ and $V'\mathcal{E}W'$; and the converse, on the reductions from W.*

We also provide a further up-to technique, that we call *up-to environment* whereby clause (2.a) of bisimilarity (the requirement $\lambda x.\, M'\ \mathcal{R}\ \lambda x.\, N'$) is removed. Its soundness is proved as follows. If $\mathcal{R}$ is a bisimulation up-to environment, define

$$\mathcal{R}_1 \stackrel{\text{def}}{=} \{(\lambda x.\, M, \lambda x.\, N) \mid \exists M', N'.\, M'\mathcal{R}N' \text{ and } M' \Longrightarrow \lambda x.\, M, N' \Longrightarrow \lambda x.\, N\}$$

and then take

$$\mathcal{R}_2 \stackrel{\text{def}}{=} \mathcal{R} \cup \mathcal{R}_1$$

We then show that $\mathcal{R}_2$ is a bisimulation up-to bisimilarity.

4.2 Example 2

This example uses a simply-typed call-by-value extended with integers, an operator for subtraction ($\hat{-}$), a conditional, and a fixed-point operator Y. The reduction rule for Y is $YV \longrightarrow V(\lambda x.\, YVx)$. As mentioned in Section 2, it is straightforward to accommodate such additions in the theory developed. (We could also encode arithmetic into the untyped calculus and adapt the example, but it would become harder to read.) Let P, Q be the terms

$$P \stackrel{\text{def}}{=} \lambda f.\, \lambda g.\, \lambda x.\, \lambda y.\, \texttt{if}\ x = 0\ \texttt{then}\ y\ \texttt{else}\ g(f\, g\, (x \hat{-} 1)\, y)$$
$$Q \stackrel{\text{def}}{=} \lambda f.\, \lambda g.\, \lambda x.\, \lambda y.\, \texttt{if}\ x = 0\ \texttt{then}\ y\ \texttt{else}\ f\, g\, (x \hat{-} 1)\, (g\, y)$$

Let $F_1 \stackrel{\text{def}}{=} \lambda z.\, Y\, P\, z$ and $F_2 \stackrel{\text{def}}{=} \lambda z.\, Y\, Q\, z$.

The terms $F_1\, g\, n\, m$ and $F_2\, g\, n\, m$ (where g is a function value from integers to integers and n, m are integers) computes $g^n(m)$ if $n \geq 0$, diverge otherwise. In both cases, however, the computations made are different. We show $F_1\, g\, n\, m \approx F_2\, g\, n\, m$ using an up-to technique for logical bisimulations. For this, we use the following relation $\mathcal{R}$:

$$\{(g^r(F_1\, g\, n\, m), F_2\, g\, n\, (g^r(m))) \mid$$
$$r, m, n \in \mathbf{Z}, r \geq 0, \text{ and } g \text{ is a closed value of type } \mathsf{int} \to \mathsf{int}\}.$$

We show that $\mathcal{R}$ is a bisimulation up-to expansion and context.

Let us consider the pair $(g^r(F_1\, g\, n\, m), F_2\, g\, n\, (g^r(m)))$. If $n = 0$, then we have:

$$\begin{array}{rcl} g^r(F_1\, g\, 0\, m) & \longrightarrow\Longrightarrow & g^r(m) \\ & & \mathcal{R}^\star \\ F_2\, g\, 0\, (g^r(m)) & \longrightarrow\succeq & g^r(m) \end{array}$$

So, the required condition holds. If $n \neq 0$, then we have

$$\begin{array}{rcl} g^r(F_1\, g\, n\, m) & \longrightarrow\Longrightarrow & g^r(g(F_1\, g\, (n \hat{-} 1)\, m)) \\ & \succeq & g^{r+1}(F_1\, g\, (n-1)\, m). \end{array}$$

and

$$\begin{array}{rl} F_2\, g\, n\, (g^r(m)) \longrightarrow\succeq & F_2\, g\, (n\hat{-}1)\, (g(g^r(m))) \\ \succeq & F_2\, g\, (n-1)\, (g^{r+1}(m)). \end{array}$$

Here, the first $\succeq$ comes from the fact that y is not copied inside the function F_2. We are done, since

$$(g^{r+1}(F_1\, g\, (n-1)\, m), F_2\, g\, (n-1)\, (g^{r+1}(m))) \in \mathcal{R}.$$

The example above makes use of key features of logical bisimulations: the ability to compare terms in the middle of evaluations, and (some of) its up-to techniques.

5 Data hiding and concurrency

To handle higher-order calculi with information hiding mechanisms, such as store, encryption, data abstraction, we have to enrich logical bisimulations with environments, which roughly collect the partial knowledge on the transmitted values, acquired by an observer interacting with the terms. The same happens in concurrency, where bisimulations with forms of environment have been first proposed, for instance to handle information hiding due to types [6, 21] and encryption [7] (this in π-calculus-like languages; information hiding in higher-order concurrency remains largely unexplored). Bisimulations with environments have also been used in λ-calculi with information hiding mechanisms (such as encryption [29], data abstraction [28], and store [15]); as pointed out in the introductions, these works have motivated and inspired ours. The resulting form of bisimulation, that we have called *environmental bisimulation*, seems robust. The technical details—which are non-trivial—are presented in [26].

6 Conclusions

In this paper we have developed the basic theory of logical bisimulations and tested it on a few representative higher-order calculi.

Bisimulation and co-inductive techniques are known to represent a hard problem in higher-order languages. While we certainly would not claim that logical bisimulations are definitely better than applicative bisimulations or other co-inductive techniques in the literature (indeed, probably a single *best* bisimulation for this does not exist), we believe it is important to explore different approaches and understand their relative merits. This paper reports our initial experiments with logical bisimulations. More experiments, both with concrete examples and with a broader spectrum of languages, are needed.

Acknowledgment

We would like to thank Søren B. Lassen for comments.

References

1. Samson Abramsky. The lazy lambda calculus. In David A. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison-Wesley, 1990.
2. Amal Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In *15th European Symposium on Programming*, pages 69–83, 2006.
3. Amal Ahmed, Andrew W. Appel, and Roberto Virga. An indexed model of impredicative polymorphism and mutable references. `http://www.cs.princeton.edu/~amal/papers/impred.pdf`, 2003.
4. Andrew W. Appel and David McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM Transactions on Programming Languages and Systems*, 23(5):657–683, 2001.
5. Lars Birkedal and Robert Harper. Relational interpretations of recursive types in an operational setting. *Information and Computation*, 155(1–2):3–63, 1999. Summary appeared in *Theoretical Aspects of Computer Software*, *Lecture Notes in Computer Science*, Springer-Verlag, vol. 1281, pp. 458–490, 1997.
6. M. Boreale and D. Sangiorgi. Bisimulation in name-passing calculi without matching. In *13th Annual IEEE Symposium on Logic in Computer Science*, pages 165–175. IEEE Computer Society Press, 1998.
7. Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Proof techniques for cryptographic processes. *SIAM Journal on Computing*, 31(3):947–986, 2002. Preliminary version appeared in *14th Annual IEEE Symposium on Logic in Computer Science*, pp. 157–166, 1999.
8. Karl Crary and Robert Harper. Syntactic logical relations for polymorphic and recursive types. In *Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin*, volume 172 of *Electronic Notes in Theoretical Computer Science*, pages 259–299. Elsevier Science, 2007.
9. William Ferreira, Matthew Hennessy, and Alan Jeffrey. A theory of weak bisimulation for core CML. *Journal of Functional Programming*, 8(5):447–491, 1998.
10. Jens Chr. Godskesen and Thomas Hildebrandt. Extending Howe's method to early bisimulations for typed mobile embedded resources with local names. In *Foundations of Software Technology and Theoretical Computer Science*, volume 3821 of *Lecture Notes in Computer Science*, pages 140–151. Springer-Verlag, 2005.
11. Andrew D. Gordon. *Functional Programming and Input/Output.* PhD thesis, University of Cambridge, 1993.
12. Andrew D. Gordon and Gareth D. Rees. Bisimilarity for a first-order calculus of objects with subtyping. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 386–395, 1996.
13. Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.
14. Alan Jeffrey and Julian Rathke. Towards a theory of bisimulation for local names. In *14th Annual IEEE Symposium on Logic in Computer Science*, pages 56–66, 1999.
15. Vassileios Koutavas and Mitchell Wand. Small bisimulations for reasoning about higher-order imperative programs. In *Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 141–152, 2006.
16. S. B. Lassen. *Relational Reasoning about Functions and Nondeterminism.* PhD thesis, Department of Computer Science, University of Aarhus, 1998.

17. Søren B. Lassen. Relational reasoning about contexts. In Andrew D. Gordon and Andrew M. Pitts, editors, *Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute, pages 91–135. Cambridge University Press, 1998.
18. Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.
19. John C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.
20. James H. Morris, Jr. *Lambda-Calculus Models of Programming Languages*. PhD thesis, Massachusetts Institute of Technology, 1968.
21. Benjamin C. Pierce and Davide Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. *Journal of the ACM*, 47(3):531–586, 2000. Extended abstract appeared in *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1997, pp. 531–584.
22. Andrew Pitts. Operationally-based theories of program equivalence. In Andrew M. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, Publications of the Newton Institute, pages 241–298. Cambridge University Press, 1997.
23. Andrew Pitts. Typed operational reasoning. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 7, pages 245–289. MIT Press, 2005.
24. David Sands. Improvement theory and its applications. In Andrew D. Gordon and Andrew M. Pitts, editors, *Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute, pages 275–306. Cambridge University Press, 1998.
25. Davide Sangiorgi. On the origins of bisimulation, coinduction, and fixed points. Draft, May, 2007.
26. Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. In *22nd Annual IEEE Symposium on Logic in Computer Science*, 2007.
27. Eijiro Sumii and Benjamin C. Pierce. Logical relations for encryption. *Journal of Computer Security*, 11(4):521–554, 2003. Extended abstract appeared in *14th IEEE Computer Security Foundations Workshop*, pp. 256–269, 2001.
28. Eijiro Sumii and Benjamin C. Pierce. A bisimulation for type abstraction and recursion. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 63–74, 2005.
29. Eijiro Sumii and Benjamin C. Pierce. A bisimulation for dynamic sealing. *Theoretical Computer Science*, 375(1–3):169–192, 2007. Extended abstract appeared in *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 161–172, 2004.