Higher-Order Languages: Bisimulation and Coinductive Equivalences (Extended Abstract)

Davide Sangiorgi

Higher-order languages: bisimulation and coinductive equivalences (extended abstract)

Davide Sangiorgi[1]

Università di Bologna and INRIA

1 Summary

Higher-order languages have been widely studied in functional programming, following the λ-calculus. In a higher-order calculus, variables may be instantiated with terms of the language. When multiple occurrences of the variable exist, this mechanism results in the possibility of copying the terms of the language.

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 [Mor68]. Finding effective methods for equivalence proofs is particularly challenging in higher-order languages: pure functional languages like the λ-calculus, and richer languages including non-functional features such as non-determinism, information hiding mechanisms (e.g., generative names, store, data abstraction), concurrency, and so on.

Bisimulation [Par81a,Par81b,Mil89,San09,San12] 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 higher-order languages by Abramsky [Abr90]. This version of bisimulation, called *applicative bisimulations*, and variants of it, have received considerable attention [Gor93,GR96,Pit97,San98,Las98]. 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 some serious limitations. For instance, they are unsound under the presence of generative names [JR99] or data abstraction [SP05] 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 [How96]. The method appears however rather subtle and fragile, for instance under the presence of generative names [JR99], non-determinism [How96], or concurrency (e.g., [FHJ98,GH05b]). 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 [Las98]; see also [San98,KW06]).

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 argument 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.

Proposals for improving applicative bisimilarity include *environmental bisimulations* [SKS11,KLS11,PS12] and *logical bisimulations* [SKS07]. 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, a further component, 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. For instance, in higher-order outputs, the values emitted by the tested terms are published to the environment, and are added to it, as part of the updated current knowledge. In contrast, when the tested terms perform a higher-order input (e.g., in λ-calculi the tested terms are functions that require an argument), the arguments supplied are terms that the observer can build using the current knowledge; that is, terms obtained by composing the values currently in the environment using the operators of the calculus.

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 is offered by 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.

The difference between environmental bisimulations and logical bisimulations is that the latter does not make use of an explicit environment: the environment is implicitly taken to be the set of pairs forming the bisimulation. This simplifies the definition, but has the drawback of making the functional of bisimulation non-monotone. In λ-calculi one usually is able to show that the functional has nevertheless a greatest fixed-point which coincides with contextual equivalence. But in richer languages this does not appear to be possible.

For bisimulation and coinductive techniques, a non-trivial extension of higher-order languages concern probabilities. Probabilistic models are more and more pervasive. Not only they are a formidable tool when dealing with uncertainty and incomplete information, but they sometimes are a *necessity* rather than an alternative, like in computational cryptography (where, e.g., secure public key encryption schemes need to be probabilistic [GM84]). A nice way to deal compu-

tationally with probabilistic models is to allow probabilistic choice as a primitive when designing algorithms, this way switching from usual, deterministic computation to a new paradigm, called probabilistic computation. Examples of application areas in which probabilistic computation has proved to be useful include natural language processing [MS99], robotics [Thr02], computer vision [CRM03], and machine learning [Pea88].

This new form of computation, of course, needs to be available to programmers to be accessible. And indeed, various programming languages have been introduced in the last years, spanning from abstract ones [JP89,RP02,PPT08] to more concrete ones [Pfe01,Goo13], being inspired by various programming paradigms like imperative, functional or even object oriented. A quite common scheme consists in endowing any deterministic language with one or more primitives for probabilistic choice, like binary probabilistic choice or primitives for distributions.

One class of languages which cope well with probabilistic computation are functional languages. Indeed, viewing algorithms as functions allows a smooth integration of distributions into the playground, itself nicely reflected at the level of types through monads [GAB+13,RP02]. As a matter of fact, many existing probabilistic programming languages [Pfe01,Goo13] are designed around the λ-calculus or one of its incarnations, like Scheme. All these allows to write higher-order functions (programs can take functions as inputs and produce them as outputs).

Bisimulation and context equivalence in a probabilistic λ-calculus have been considered in [ALS14], where a technique is proposed for proving congruence of *probabilistic applicative bisimilarity*. While the technique follows Howe's method, some of the technicalities are quite different, relying on non-trivial "disentangling" properties for sets of real numbers, these properties themselves proved by tools from linear algebra. The bisimulation is proved to be sound for contextual equivalence. Completeness, however, fails: applicative bisimilarity is strictly finer. A subtle aspect is also the late vs. early formulation of bisimilarity; with a choice operator the two versions are semantically different; the congruence proof of bisimilarity crucially relies on the late style.

Context equivalence and bisimilarity, however, coincidence on pure λ-terms. The resulting equality is that induced by *Levy-Longo trees* (LLT), generally accepted as the finest extensional equivalence on pure λ-terms under a lazy regime. The proof follows Böhm-out techniques along the lines of [San94,SW01]. The result is in sharp contrast with what happens under a nondeterministic interpretation of choice (or in the absence of choice), where context equivalence is coarser than LLT equality.

A coinductive characterisation of context equivalence on the whole probabilistic language is possible via an extension in which weighted formal sums — terms akin to distributions — may appear in redex position. Thinking of distributions as sets of terms, the construction reminds us of the reduction of nondeterministic to deterministic automata. The technical details are however quite different, because we are in a higher-order language and therefore — once more

— we are faced with the congruence problem for bisimulation, and because formal sums may contain an *infinite* number of terms. The proof of congruence of bisimulation in this extended language uses the technique of logical bisimulation, therefore allowing bisimilar functions to be tested with *bisimilar* (rather than identical) arguments (more precisely, the arguments should be in the context closure of the bisimulation). In the probabilistic setting, however, the ordinary logical bisimulation game has to be modified substantially. For instance, formal sums represent possible evolutions of running terms, hence they should appear in redex position only (allowing them anywhere would complicate matters considerably). The obligation of redex position for certain terms is in contrast with the basic schema of logical bisimulation, in which related terms can be used as arguments to bisimilar functions and can therefore end up in arbitrary positions. This problem is solved by moving to *coupled logical bisimulations*, where a bisimulation is formed by a pair of relations, one on ordinary terms, the other on terms extended with formal sums. The bisimulation game is played on both relations, but only the first relation is used to assemble input arguments for functions.

In higher-order languages coinductive equivalences and techniques appear to be more fundamental than in first-order languages. Evidence of this are the above-mentioned results of correspondence between forms of bisimilarity and contextual equivalence in various λ-calculi. Contextual equivalence is a '*may*' of form of testing that, in first-order languages (e.g., CCS) is quite different from bisimilarity or even simulation equivalence. Indeed, in general, higher-order languages have a stronger discriminating power than first-order languages [BSV14]. For instance, if we use higher-order languages to test first-order languages, using (may-like) contextual equivalence, then the equivalences induced is often finer than the equivalences induced by first-order languages (usually trace equivalence); moreover, the natural definition of the former equivalences is coinductive, whereas that for the latter equivalences is inductive. In *distributed* higher-order languages, a construct that may strongly enhance the discriminating power is *passivation* [SS03,GH05a,LSS09a,LSS09b,LSS11,LPSS11,PS12,KH13]. Passivation offers the capability of capturing the content of a certain location into a variable, possibly copying it, and then restarting the execution in different contexts. The same discriminating power can also be obtained in call-by-value λ-calculi (that is, without concurrency or nondeterminism) extended with a location-like construct akin to a store of imperative λ-calculi, and operators for reading the content of this location, overriding it, and, if the location contains a process, for consuming such process (i.e., performing observations on the process actions). When the tested first-order processes are probabilistic, the difference in discriminating power between first-order and higher-order languages increases further: in higher-order languages equipped with passivation, or in a call-by-value λ-calculus, bisimilarity may be recovered [BSV14].

References

[Abr90] Samson Abramsky. The lazy lambda calculus. In David A. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison-Wesley,

1990.

[ALS14] Michele Alberti, Ugo Dal Lago, and Davide Sangiorgi. On coinductive equivalences for higher-order probabilistic functional programs. In *Proc. POPL'14*. ACM, 2014.

[BSV14] Marco Bernardo, Davide Sangiorgi, and Valeria Vignudelli. On the discriminating power of passivation and higher-order interaction. Submitted, 2014.

[CRM03] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Kernel-based object tracking. *IEEE Trans. on Pattern Analysis and Machine Intelligence,*, 25(5):564–577, 2003.

[FHJ98] William Ferreira, Matthew Hennessy, and Alan Jeffrey. A theory of weak bisimulation for core CML. *Journal of Functional Programming*, 8(5):447–491, 1998.

[GAB+13] Andrew D. Gordon, Mihhail Aizatulin, Johannes Borgström, Guillaume Claret, Thore Graepel, Aditya V. Nori, Sriram K. Rajamani, and Claudio V. Russo. A model-learner pattern for bayesian reasoning. In *POPL*, pages 403–416, 2013.

[GH05a] J. C. Godskesen and T. T. Hildebrandt. Extending howe's method to early bisimulations for typed mobile embedded resources with local names. In *Proc. FSTTCS*, volume 3821 of *Lecture Notes in Computer Science*, pages 140–151. Springer, 2005.

[GH05b] 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.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[Goo13] Noah D. Goodman. The principles and practice of probabilistic programming. In *POPL*, pages 399–402, 2013.

[Gor93] Andrew D. Gordon. *Functional Programming and Input/Output*. PhD thesis, University of Cambridge, 1993.

[GR96] 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.

[How96] Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.

[JP89] C. Jones and Gordon D. Plotkin. A probabilistic powerdomain of evaluations. In *LICS*, pages 186–195, 1989.

[JR99] 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.

[KH13] V. Koutavas and M. Hennessy. Symbolic bisimulation for a higher-order distributed language with passivation. In *Proc. CONCUR'13*, volume 8052 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2013.

[KLS11] Vasileios Koutavas, Paul Blain Levy, and Eijiro Sumii. From applicative to environmental bisimulation. *Electr. Notes Theor. Comput. Sci.*, 276:215–235, 2011.

[KW06] 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.

[Las98] S. B. Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, Department of Computer Science, University of Aarhus, 1998.

[LPSS11] Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. *Inf. Comput.*, 209(2):198–226, 2011.

[LSS09a] S. Lenglet, A. Schmitt, and J.-B. Stefani. Howe's method for calculi with passivation. In *Proc. CONCUR'09*, volume 5710 of *Lecture Notes in Computer Science*, pages 448–462. Springer, 2009.

[LSS09b] S. Lenglet, A. Schmitt, and J.-B. Stefani. Normal bisimulations in calculi with passivation. In *Proc. FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 257–271. Springer, 2009.

[LSS11] S. Lenglet, A. Schmitt, and J.-B. Stefani. Characterizing contextual equivalence in calculi with passivation. *Inf. Comput.*, 209(11):1390–1433, 2011.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mor68] James H. Morris, Jr. *Lambda-Calculus Models of Programming Languages*. PhD thesis, Massachusetts Institute of Technology, 1968.

[MS99] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*, volume 999. MIT Press, 1999.

[Par81a] D. Park. A new equivalence notion for communicating systems. In G. Maurer, editor, *Bulletin EATCS*, volume 14, pages 78–80, 1981. Abstract of the talk presented at the Second Workshop on the Semantics of Programming Languages, Bad Honnef, March 16–20 1981. Abstracts collected in the Bulletin by B. Mayoh.

[Par81b] D.M. Park. Concurrency on automata and infinite sequences. In P. Deussen, editor, *Conf. on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.

[Pea88] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[Pfe01] Avi Pfeffer. IBAL: A probabilistic rational programming language. In *IJCAI*, pages 733–740. Morgan Kaufmann, 2001.

[Pit97] 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.

[PPT08] Sungwoo Park, Frank Pfenning, and Sebastian Thrun. A probabilistic language based on sampling functions. *ACM Trans. Program. Lang. Syst.*, 31(1), 2008.

[PS12] Adrien Piérard and Eijiro Sumii. A higher-order distributed calculus with name creation. In *LICS*, pages 531–540. IEEE, 2012.

[RP02] Norman Ramsey and Avi Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *POPL*, pages 154–165, 2002.

[San94] D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Inf. and Comp.*, 111(1):120–153, 1994.

[San98] 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.

[San09] Davide Sangiorgi. On the origins of bisimulation and coinduction. *ACM Trans. Program. Lang. Syst.*, 31(4), 2009.

[San12] Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012.

[SKS07] Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Logical bisimulations and functional languages. In *Proc. Fundamentals of Software Engineering (FSEN)*, volume 4767 of *LNCS*, pages 364–379. Springer-Verlag, 2007.

[SKS11] Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. *ACM Trans. Program. Lang. Syst.*, 33(1):5, 2011.

[SP05] 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.

[SS03] A. Schmitt and J.-B. Stefani. The m-calculus: a higher-order distributed process calculus. In *Proc. POPL'03*, pages 50–61. ACM, 2003.

[SW01] Davide Sangiorgi and David Walker. *The pi-Calculus – a theory of mobile processes*. Cambridge University Press, 2001.

[Thr02] Sebastian Thrun. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, pages 1–35, 2002.