Bisimulation and Co-induction:
Some Problems

Davide Sangiorgi

University of Bologna, Italy

Abstract

Bisimulation and co-induction are one of the most important contributions to Computer Science that stem from the work on algebraic process calculi. In this note, we review a few outstanding problems that concern bisimulation and co-induction.

Keywords: Bisimulation, higher-order languages, up-to techniques, congruence

Bisimulation and, more generally, co-induction, can be regarded as one of the most important contributions to Computer Science that stem from the work on algebraic process calculi. Nowadays, bisimulation and the co-inductive techniques developed from the idea of bisimulation are widely used, not only in Concurrency, but, more broadly, in Computer Science, in a number of areas: functional languages, object-oriented languages, type theory, data types, domains, databases, compiler optimisations, program analysis, verification tools, etc.. For instance, in type theory bisimulation and co-inductive techniques have been used: to prove soundness of type systems; to define the meaning of equality between (recursive) types and then to axiomatise and prove such equalities; to define co-inductive types and manipulate infinite proofs in theorem provers. Also, the development of Final Semantics, an area of Mathematics based on co-algebras and category theory and that gives us a riche and deep perspective on the meaning of co-induction and its duality with induction, has been largely motivated by the interest in bisimulation.

The classical notion of bisimulation is defined on a Labelled Transition System
(LTS) thus, where $\Sigma$ is the set of all states of the LTS:

A relation $R \subseteq \Sigma \times \Sigma$ is a bisimulation if

$$\quad (P_1, P_2) \in R \text{ and } P_1 \xrightarrow{\mu} P'_1 \text{ imply:}$$

there is $P'_2$ such that $P_2 \xrightarrow{\mu} P'_2$ and $(P'_1, P'_2) \in R$, and the converse, on the actions from $P_2$.

Bisimilarity is then defined as the union of all bisimulations. When the states of the LTS are processes, bisimilarity can be taken as the definition of behavioural equality for them.

The definition of bisimilarity is an example of co-inductive definition; the bisimulation proof method is an example of co-inductive proof method.

Below I briefly discuss three directions for future work related to the notion of bisimulation.

Bisimulation is continuously applied to new formalisms. Often these formalisms bring in new requirements that make the classical definition (1) inappropriate. An example are higher-order process languages, that is, languages in which processes, or terms including processes, can move or be communicated. Consider for instance processes $A \overset{\text{def}}{=} \overline{a}(P|Q) . 0$ and $B \overset{\text{def}}{=} \overline{a}(Q|P) . 0$. These processes can only perform one action, at $a$. In this action, $A$ emits $P|Q$ (the parallel composition of the processes $P$ and $Q$), $B$ emits $Q|P$. Thus, if $P$ and $Q$ are syntactically different, the two processes are distinguished according to definition (1). Hence, an important algebraic law such as the commutativity of parallel composition is broken. (The problem in this specific example can be overcome by requiring bisimilarity rather than identity on the processes emitted in a higher-order output action. This form of bisimulation, called higher-order bisimulation, [16,3], is however troublesome in other situations, see [12] for discussions.)

Other examples of languages in which definition (1) is over-discriminating are typed languages of mobile processes such as the pi-calculus [10,15], and calculi for security such as the spi-calculus [2]. In these cases, as in the case of higher-order processes, matching transitions of bisimilar processes should not necessarily be identical. Further, the knowledge (on the type of values, on secrecy keys, etc.) that the external observer has acquired is significant and, for instance, implies that not all actions of the processes are observable. (See [5,1,4] for more details.)

But if definition (1) cannot be used, what is, and how can we find, the “right” definition? A method that has been extensively used is based on barbed bisimulation [11,15], or variants of it (such as [7], sometimes called reduction-closed barbed congruence). Barbed bisimulation can be uniformly applied to different formalisms because we equip a global observer with a minimal ability to observe actions and/or process states. We then obtain an equivalence, namely inindistinguishability under global observations. This in turn induces a congruence over agents, namely equivalence in all contexts, called barbed congruence. In barbed bisimulation, the bisimulation game is only played on the interactions of processes, as opposed to visible
actions such as input and output. The only checks performed on visible actions are represented by an observability predicate that gives the external observer visibility of the channel at which an action occurs.

Context-based behavioural equalities like barbed congruence suffer from the universal quantification on contexts, that makes it very hard to prove process equalities following the definition, and makes mechanical checking impossible. However, barbed congruence can guide us to find direct characterisations, as forms of labelled bisimilarity without quantification on contexts. For instance, definition (1) is a direct characterisation of barbed congruence in CCS.

Unfortunately, deriving a labelled bisimilarity from barbed bisimulation may require a lot of ingenuity. Further, proofs tend to be very sensitive to the language adopted – a small modification to the language can have dramatic consequences. A general methodology for deriving labelled bisimilarity starting from the syntax and the operational semantics of the language is missing here. The value of this methodology would depend on whether it is robust (applicable to a broad range of language), and algorithmic (based on a number of steps each of which as elementary as possible). Sewell’s contextual labelled transitions [14] can be seen as a progress in this direction.

Another important and related issue is that when bisimilarity departs from the classical definition (1) it may be hard to establish its properties. For instance, in higher-order process calculi it may be hard to prove that a labelled bisimilarity is a congruence relation. In sequential higher-order languages, congruence properties of bisimilarity are usually established using Howe’s technique [6]. However, in Concurrency such a technique appears to work only in a limited number of cases. Some progress in this direction has been made [12,9,8], but doubts remain on how general and powerful these techniques are.

A third challenging direction for future work that I would like to mention is the enhancement of the bisimulation (and more generally, the co-induction) proof method. I discuss this below.

In the clauses of definition (1) the same relation $R$ is mentioned in the hypothesis and in the thesis. In other words, when we check the bisimilarity clause on a pair $(P_1, P_2)$, all needed pairs of derivatives, like $(P'_1, P'_2)$, must be present in $R$. We cannot discard any such pair of derivatives from $R$, or even “manipulate” its process components. In this way, a bisimulation relation often contains many pairs strongly related with each other, in the sense that, at least, the bisimilarity between the processes in some of these pairs implies that between the processes in other pairs. (For instance, in a process algebra a bisimulation relation might contain pairs of processes obtainable from other pairs through application of algebraic laws for bisimilarity, or obtainable as combinations of other pairs and of the operators of the language.) These redundancies can make both the definition and the verification of a bisimulation relation annoyingly heavy and tedious: It is difficult at the beginning to guess all pairs which are needed; and the clause of (1) must be checked on all pairs introduced.

As an example, let $P$ be a non-deadlocked process from a CCS-like language, and
!P the process defined thus: \( !P \overset{\text{def}}{=} P | !P \). Process \( !P \) represents the replication of \( P \), i.e., a countable number of copies of \( P \) in parallel. (In certain process algebras, e.g., the pi-calculus, replication is the only form of recursion allowed, since it gives enough expressive power and enjoys interesting algebraic properties.) A property that we naturally expect to hold is that duplication of replication has no behavioural effect, i.e., \( !P | !P \sim !P \) (where \( \sim \) is the bisimilarity relation). To prove this, we would like to use the singleton relation

\[
S \overset{\text{def}}{=} \{ ( !P | !P, !P ) \}.
\]

But \( S \) is easily seen not to be a bisimulation relation. If we add pairs of processes to \( S \) so to make it into a bisimulation relation, then we might find that the simplest solution is to take the infinite relation

\[
R \overset{\text{def}}{=} \{ (Q_1, Q_2) : \text{for some } R, Q_1 \sim R | !P | !P \text{ and } Q_2 \sim R | !P \}.
\]

The size augmentation in passing from \( S \) to \( R \) is rather discouraging. But it does somehow seems unnecessary, for the bisimilarity between the two processes in \( S \) already implies that between the processes of all pairs of \( R \).

Some techniques have been proposed that do allow us to relieve the work involved with the bisimulation proof method. For instance, on the previous example, the “bisimulation up to context and up to bisimilarity” technique indeed allows us to prove the property \( !P | !P \sim !P \) simply using the singleton \( S \). In this technique, the pair of derivatives \( P'_1 \) and \( P'_2 \) in (1) need not be in \( R \). It is sufficient to find processes \( P''_1, P''_2, \) and a context \( C[·] \) such that, for \( i = 1, 2 \),

\[
P'_i \sim C[P''_i], \quad (2)
\]

and then only the pair \( (P''_1, P''_2) \) has to be in \( R \). Intuitively, the reason why this technique is sound is that bisimilarity is a congruence, in particular it is preserved by all contexts and it is transitive. Hence, from \( P''_1 \sim P''_2 \) we can infer \( C[P''_1] \sim C[P''_2] \), and then from this and (2) we can conclude \( P'_1 \sim P'_2 \) by transitivity.

In summary, by enhancements of the bisimilarity proof method I refer to methods that allow us to prove bisimilarity results using relations that are strictly included in a bisimulation. Such relations should be as small as possible; precisely, they should have no “redundant” pairs, in the sense discussed above. However, the precise meaning of “redundant” is not clear. Intuitions can be deceptive here. For instance, one might reasonably think that the “bisimulation up to context and up to bisimilarity” technique is always sound if bisimilarity is a congruence. But this is not true; see [13] for counterexamples.

“Bisimilarity up-to” techniques are heavily used in languages for mobility and in concurrent higher-order languages. The proofs of several basic results of the theory of these languages seem infeasible without them. However, most of these techniques
have been introduced in a rather ad-hoc fashion, to solve specific problems on specific languages.

We need to understand better what is an enhancement of the bisimulation proof method: what makes an enhancement sound and why, and how it can be used. Here again, it would be highly desirable to have general results, applicable to different languages.

References


