Beyond Bisimulation: The “up-to” Techniques

Davide Sangiorgi

Università di Bologna, Italy
http://www.cs.unibo.it/~sangio/

Abstract. We consider the bisimulation proof method – an instance of the co-induction proof method – that is at the heart of the success of bisimulation. We discuss a number of enhancements of the method and some open problems.

1 Bisimulation

Bisimulation (and, more generally, co-induction) can be regarded as one of the most important contributions of Concurrency Theory to Computer Science. 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 rich and deep perspective on the meaning of co-induction and its duality with induction, has been largely motivated by the interest in bisimulation.

In this paper we consider the bisimulation proof method – an instance of the co-induction proof method – that is at the heart of the success of bisimulation. More precisely, we discuss enhancements of the method, motivate them, and hint at some related open problems. This is not supposed to be a comprehensive paper, but rather a quick guide to the state of the art in the topic, which the interested reader could also use to search for more details.

We consider bisimilarity on standard labelled transition systems. Their transitions are of the form \( P \xrightarrow{\mu} Q \), where \( P \) and \( Q \) are processes, and label \( \mu \) is drawn from some alphabet of actions.

Definition 1. A relation \( R \) on processes is an bisimulation if whenever \((P, Q) \in R\),

1. \( P \xrightarrow{\alpha} P' \) implies \( Q \xrightarrow{\alpha} Q' \) and \((P', Q') \in R\), for some \( Q' \)
2. the converse, on the actions from \( Q \).

\( P \) and \( Q \) are bisimilar, written \( P \sim Q \), if \((P, Q) \in R\) for some bisimulation \( R \).
(\sim \text{ can also be viewed as the greatest fixed-point of a certain monotone function on relations, whose definition closely follows the bisimulation clauses above.) By definition of \sim, a bisimulation relation is contained in \sim, and hence it consists of only pairs of bisimilar processes. This immediately suggests a proof method for \sim, by far the most popular one: to demonstrate that \((P,Q) \in \sim\) holds, find a bisimulation relation containing the pair \((P,Q)\).

2 An Example of Redundancy

In the clauses of definition (1) the same relation \(S\) is mentioned in the hypothesis and in the thesis. In other words, when we check the bisimilarity clause on a pair \((P,Q)\), all needed pairs of derivatives, like \((P',Q')\), must be present in \(S\). We cannot discard any such pair of derivatives from \(S\), 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 bisimulation clauses 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 recursively defined thus: \(!P \overset{\text{def}}{=} P \parallel !P\). 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., \(!P \parallel !P \sim !P\). To prove this, we would like to use the singleton relation

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

But \(S\) is easily seen not to be a bisimulation relation. For instance, if \(P \xrightarrow{\mu} P'\), then we have

\[ !P \parallel !P \xrightarrow{\mu} !P \parallel P' \parallel !P \]

The derivative process does not appear in the processes of \(S\), hence \(!P\) cannot possibly match the transition from \(!P \parallel !P\) in a way that would close the bisimulation diagram.

If we add pairs of processes to \(S\) so as to make it into a bisimulation relation, then we might find that the simplest solution is to take the infinite relation

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