INTRODUCTION

The notion of bisimulation (due to Park [P] and Milner [M2]) has become a classical tool for characterizing a basic semantics of concurrent systems, whose behaviour (either directly or indirectly) is operationally defined by labelled transition systems. Indeed, that notion has first of all an appealing intuitive meaning and, moreover, is associated with an elegant and powerful proof technique (see especially [M2]).

The problem we tackle in this paper is to extend and generalize that notion and the associated proof technique to handle, for example, concurrent systems algebraically specified as abstract data types, as it has been done, eg, in [BW1, AMRW, AR1]. The motivation is that algebraic specifications of concurrent systems are a framework where we can handle together concurrency, data types and functions (see [AR]) and, as shown in [AR1], define powerful families of concurrent calculi, where, for example, processes can exchange messages which include processes as data and systems can be composed into multilevel systems. Hence, it is important to have at hand a notion of bisimulation which allows us to prove semantic identities on those calculi as it was done for CCS and SCCS.

As it has been shown in [ARW], algebraic transition systems are a special case of relational specifications, a general framework for specifying data types with observations. One of the novelties of the paper is to show that a natural generalization of Park-Milner's bisimulation can be given within that framework and used to provide semantics for data type specifications.

Moreover, even if we restrict ourselves to consider only algebraic transition systems, our notion of bisimulation generalizes the Park-Milner's one in an essential point, apart for being compatible with an algebraic setting: the labels of a process can have processes as subterms. More generally, in the language of relational specifications, the observations can have the same nature of the observed objects. Hence bisimulation semantics can handle situations which are not covered by a terminal algebra semantics, as eg in [ARW].

We emphasize the fact that here we stick closely to the Park-Milner's point of view. Hence the basic idea is to use the observations (eg, the transitions in an algebraic transition system) in order to define a semantic equivalence on the observed objects (eg, the states in an algebraic transition system), without requiring by construction a closure by congruence as it is done in [AW]. Like for CCS and SCCS, the fact that the bisimulation equivalence results to be a congruence on the observed objects is taken as a criterion for considering sensible the given specification and indeed the present definition is applied in [AR2] to show that the bisimulation equivalence is a congruence for the parameterized higher-order calculi defined in [AR1]. Moreover, since the bisimulation is a greatest fixed point, we can keep the nice associated proof technique. Finally, we can extend to relational specifications the Milner's inductive definition of strong equivalence in [M1] and show that it is equivalent to the bisimulation equivalence under a condition that in the case of labelled transition systems is less restrictive then the usual finitely branching condition.

The paper is organized as follows. In section 1, with the help of two simple examples, we explain the problem and motivate a proposal for a solution. In section 2 we formally define the concept of generalized bisimulation, give the
relevant properties justifying an associated proof technique and showing sufficient conditions for a bisimulation congruence to be a model of the non-relational part of a specification. Then section 3 is devoted to illustrate, still by very simple examples, the wide range of application of the concept. Finally, section 4 generalizes the notion of strong equivalence and shows the coincidence with bisimulation equivalence under a generalized finitely branching condition. The nontrivial proofs are in Appendix 2.

As it has been already mentioned the main application of this work is to algebraic higher-order concurrent calculi where processes are just data types and hence one can have functions with processes as arguments and values as in [AR1, AR2]. The present setting can be extended to the case of higher-order partial algebraic specifications. We have preferred to stick here to first-order specifications both for simplicity and lack of room for introducing some machinery, similarly to [M6]; a higher-order version is in preparation.

Another related interesting investigation is to fully compare the present definition with the hierarchy of simulation congruences built in [AW], in the framework of hierarchical algebraic specifications.

1 THE PROBLEM

1.1 Example 1. The Case Of Algebraic Transition Systems

A labelled transition system is a triple \((S,F,T)\), where \(S\) is a set of objects called states, \(F\) a set of objects called flags (or labels) and \(T\) is a subset of \(S \times F \times S\); every triple \((s,f,s') \in T\) is called a transition and written usually \(s \xrightarrow{f} s'\).

A specification of an algebraic transition system (see [AMRW, BW1]) is a specification \(TS\) of the form

\[
\text{enrich } \text{STATE} + \text{FLAG} + \text{RELBOOL} \text{ by }
\]

\[
\text{opns } \square \longrightarrow \square : \text{state} \times \text{flag} \times \text{state} \rightarrow \text{relbool}
\]

\[
\text{axioms } \text{Ax}
\]

where \(\text{STATE}\) and \(\text{FLAG}\) are algebraic specifications of elements of sort state and flag respectively and \(\text{Ax}\) is a set of axioms of the form "\(\text{cond} \Rightarrow s \xrightarrow{f} s' = \text{True}\)".

We have here a specification \(\text{RELBOOL} = \text{sorts relbool opns True: relbool (total)},\) which plays the same role as \(\text{BOOL}\) but, for technical reasons, it must be distinguished.

Here and in the following we assume that the sorts are in positive conditional form, so that the initial model always exists. The algebraic notations and results which we use in the paper can be found in [BW2] and a short summary is given in Appendix 1.

We give a very simple toy example of an algebraic transition system representing processes which evolve in parallel. This is not an interesting example of a transition system, but we have preferred a very simple example to a significant one in order to emphasize techniques and concepts. This example can be extended to a more significant one (for example, with handshaking communication; see [AR1]). We introduce here only the following combinators for expressing processes: \(\text{skip}\) (the null process), \(\text{error}\) (corresponding to some erroneous process state), \(\text{p1} ; \text{p2}\) (sequential composition), \(\text{p1} \text{or} \text{p2}\) (binary nondeterministic choice) and \(\text{send}(\text{ch},v)\) (output of a value through a channel). The values can be either integer numbers or processes, so that processes can output processes as values. Processes can be composed in parallel (\(\text{p1} \| \text{p2}\)), and the policy for handling parallel executions is pure interleaving.

We define processes by means of an algebraic specification \(\text{PROCESS}\); we suppose that the specifications \(\text{INT}\) of integers and \(\text{CHAN}\) of channel identifiers are given elsewhere:

\[
\text{VAL} = \text{enrich } \text{INT by }
\]

\[
\text{sorts val, process opns }
\]

\[
i_{\text{int}}: \text{int} \rightarrow \text{val} \quad (\text{injection, total})
\]

\[
i_{\text{process}}: \text{process} \rightarrow \text{val} \quad (\text{injection, total})
\]

Usually \(i_{\text{int}}(x)\) and \(i_{\text{process}}(x)\) are written simply \(x\).