Verification Methods for Finite Systems

(Extended Abstract)

Ahmed Bouajjani Joseph Sifakis

Laboratoire de Génie Informatique, IMAG, BP53X, 38041 Grenoble-cedex, France

e-mail : bouajjan@imag.fr, sifakis@imag.fr

The increasing complexity of concurrent systems requires the development of rigorous design methods. This need has motivated research on specification formalisms and the associated verification methods and tools.
By verification, we mean the comparison of a system against its specifications. We consider that a system is described by a program. Specifications describe the service provided by the system and may concern both qualitative and quantitative aspects of its behaviour. In this paper, we do not address the verification problem of quantitative requirements such as reliability or performance.

Verification methods strongly depend on the features of the specification language.
Early results on verification concern sequential programs (systems) which can be specified in terms of initial/final state relations. The specification languages used in this case express two kinds of properties : strong correctness and weak correctness. The latter means that if the program terminates then the initial/final state relation is satisfied. The former requires in addition to that, termination of the program.
The study of specification and verification methods for concurrent systems has shown that verification theories for sequential programs can be extended to concurrent programs. However, these extensions are not sufficient as they concern a very restricted subset of the significant properties of concurrent systems.
In fact, concurrent systems or their components are reactive in nature. Their rôle is to maintain an interaction with their environment and consequently they must be specified in terms of their current behaviour. Of course, some interactions may produce a final result but in general termination is not a desirable property. For this reason, specification formalisms of concurrent systems describe behaviours or global properties of behaviours, rather than relations between initial and final states. A behaviour characterises a family of sequences of *observables*, the latter being either properties of the states or properties of the transitions (state changes) or a combination of them. The specification problem becomes in this context non trivial : choice and definition of the observables, soundness and exhaustivity of the descriptions. Consequently, the verification problem changes and it is important to say that contrary to the sequential case, compositional verification methods are not available for all (most of) the types of properties occuring in specifications.

According to the type of the specification language, two main approaches to the specification and thus to the verification problem, can be distinguished : the *transition-based* and the *logic-based* approach.

By transition-based languages we mean languages whose semantics is defined in terms of transition systems, for example programming languages with operational semantics, process algebras, automata, Petri nets. Specifications in such languages describe a behaviour observed at a certain abstraction level.
In this case, both the system (program) and its specifications can be considered as transition systems. The comparison is carried out by means of an appropriate equivalence relation or an implementation preorder defined on transition systems. Any decision procedure for these relations is a verification method.

Formulas of a logic are used to express specifications as a set of global system properties such as, deadlockfreeness, mutual exclusion, fairness, etc. In this case, a satisfaction relation relating programs to formulas is defined; a property is an assertion expressing the fact that a program satisfies a formula. Any decision procedure for the satisfaction relation is a verification method.
It is generally admitted that the two approaches are complementary and combining them could simplify both specification and verification tasks.

One can distinguish two classes of verification methods.
The first, contains methods known as *axiomatic* or *deductive* which consist in defining a deductive system - a set of axiom schemes and inference rules - characterising the relation connecting programs to specifications. Then, verifying boils down to proving in the deductive system.
The second, contains *model-based* methods. The semantics of both the programming and the specification languages associate respectively with a program a model m and with a specification a class of models M. Then, verifying boils down to testing that m belongs to M.

A main obstacle to the effective exploitation of verification methods of general applicability is the absence of general verification algorithms, due to the undecidability of most of the non trivial program properties such as, termination, deadlockfreeness, boundedness of variables. Thus, only interactive general methods can be used whose efficiency heavily depends on the user's skills.
The undecidability limitation breaks down when one restricts verification to finite state systems. It is now generally admitted that essential aspects of the functioning of many classes of safety critical systems such as communication protocols, systems of real time control and hardware, can be modelled by using finite state models. This is explained by the fact that their complexity is essentially due to their control and only a finite set of data values is relevant to their behaviour.
These facts have motivated, during the last decade, active research and development of model-based verification methods applicable to concurrent finite state systems.

This paper is a survey and synthesis on these methods focusing on applicability and exploitability issues. It is composed of three sections. The first two are devoted to the presentation of the transition-based and logic-based approaches; the last tackles the problem of combining the two approaches and their underlying verification methods.

Process algebras are typical representatives of transition-based specification languages. Their terms are built by using operators corresponding to process constructors such as, prefixing by an action, non deterministic choice, parallel composition, etc. We show that the various behavioural semantics used for process algebras such as trace, failure, testing, acceptance semantics, can be viewed as bisimulation or simulation semantics on particular classes of labelled transition systems. A practical consequence of this fact is that deciding behavioural equivalences boils down to deciding bisimulation or simulation equivalences on transition systems.

The logic-based specification languages presented are the propositional temporal logics PTL for linear time and CTL for branching time. For PTL, the verification methods are based on well-known results showing that with any formula can be associated an ω-automaton accepting the set of its models. Then, verifying that a model satisfies a formula amounts to checking that the trace language of the model is included in the language of the ω-automaton representing the formula. In principle, a similar automata-based verification method can be applied for CTL but it turns out to be much less tractable as it requires the use of automata on infinite trees. However, there exist simpler *model checking* methods, specific to branching time, which consist in evaluating the characteristic sets of formulas on the model. We describe these methods and discuss their advantages and limitations.

The paper terminates with a comparison of the two specification approaches and a presentation of the results about their unification. These results concern the definition of logics that are *adequate* or even *expressive* with respect to a behavioural semantics. Adequacy means that the equivalence induced by the logic on models agrees with behavioural equivalence, and expressivity means that any equivalence class can be characterised in the logic. The temporal logics PTL and CTL are respectively adequate for trace and bisimulation semantics but are not expressive for them. On the other hand, behavioural equivalences cannot capture all kinds of properties expressed in temporal logics, for instance liveness. A framework for combining the two approaches is provided by μ-calculi as they are expressive enough to characterise equivalence classes, and they are more expressive than the temporal logics with the same underlying semantics.