An Automata-Theoretic Approach to Software Model-Checking

Previous research track record

Personnel

Prof. Javier Esparza is Professor of Theoretical Computer Science at the University of Edinburgh. He is co-author of a monograph in the series Cambridge Tracts in Theoretical Computer Science, and has published over 70 scientific papers. He has delivered invited lectures at many international conferences and workshops, including ETAPS and CONCUR, and has served in over fifteen international programme committees, including CAV, CONCUR, ETAPS, ICALP, and LICS. Since 1995 he has held grants in Germany and the UK funding more than 35 person-years of research. He has made significant contributions to the automatic verification of finite and infinite state systems and to concurrency theory. In collaboration with colleagues and Ph.D. students, he has developed verification tools like PEP, The Model-Checking Kit, and MOPED.

Stefan Schwoon is the named researcher for one of the positions solicited. He graduated in Computer Science at the University of Hildesheim (Germany) in 1998 with the overall top mark. In December 1998 he obtained a grant at a Graduiertenkolleg of the Technical University of Munich to do his Ph.D. under the supervision of Javier Esparza. He will complete his thesis in March 2002. He has published three papers in international conferences, one of which has been invited to a special issue of Information and Computation. He is the main developer of MOPED, a model-checker for procedural programs.

Host organisation

The **Division of Informatics** at the University of Edinburgh is widely recognised as a world class center for Informatics research, as confirmed by a 5^*A rating in the last RAE. Javier Esparza is a member of the **Laboratory for Foundations of Computer Science** (LFCS), one of the largest institutes of the Division. The basis of this proposal is the computer-assisted verification of systems with an infinite state space, an area which was pioneered at Edinburgh through the work of LFCS members like Stirling and Bradfield. The proposal focuses on systems obtained (after possibly an abstraction process) from C and Java programs, and LFCS hosts a strong group on programming languages and their semantics, with Plotkin and Sannella as leaders.

An Automata-Theoretic Approach to Software Model-Checking

Proposed research and its context

Background

Model-checking is a formal verification technique in which a desired behavioural property is verified over a given system (the model) through exhaustive exploration of the reachable states and possible behaviours of the system. Its main advantages with respect to other techniques, like theorem proving, are its high degree of automation and the fact that model-checkers provide counterexamples: when the design does not satisfy a property, a behaviour is produced that falsifies it.

Model-checking has already had an important impact in the hardware industry. The number of researchers and engineers working on the development of in-house model-checkers and their applications within companies like IBM, Intel, Lucent, Nokia, Ericsson, or Siemens is measured in hundreds.

Even though the software industry is in large need of automatic analysis tools (as witnessed e.g. by recent failures of space missions due to software errors), it currently uses model-checking techniques in a much smaller scale than the hardware industry. One of the main reasons is the fact that model-checking has focused mostly on the analysis of systems with a possibly large but finite state space. While the finiteness constraint is appropriate for hardware, even the simplest software systems may have an infinite state space. The sources of infinity in software systems can be roughly classified into two groups:

- Control: (possibly recursive) procedures and process creation require to enrich the notion of state with unbounded stacks or sets.
- Data: the use of any datatype with an infinite range, like integers, lists, queues, etc. potentially leads to an infinite state space.

In the last years, a lot of effort has been devoted to dealing with these infinities. The principal investigator, in collaboration with colleagues and Ph. D. students, has developed an automata-theoretic approach to the infinities raised by control structures. Most of the work has been devoted to the analysis of programs with possibly recursive procedures, for which basic theory, efficient algorithms, and prototype tools have been developed. In procedural programs, one of the components of a global state is a stack of activation records, which can be modelled as a word over a suitable alphabet. In the automata-theoretic approach, finite automata over this alphabet are used as a finite representation of a possibly infinite set of stack contents. In [5], the verification problem was reduced to the construction and subsequent automatic analysis of finite automata representing the set of predecessors or successors of a set of reachable states. In [6], efficient algorithms for these tasks were developed. In [7], the algorithms were generalised to check more general properties. Two prototype implementations and first experimental results have been described in [6, 9].

This research has been carried out in a very competitive environment. Several groups in the USA have developed algorithms for the same or closely related problems. Our approach distinguishes itself from others by its generality (other approaches, like that of [2], can only be applied to safety properties), and by the existence of efficient implementations (the approaches of [1, 3, 10] have not been implemented, while experiments carried out in November 2001 in collaboration with colleagues at Microsoft Research show that our model-checker is more efficient than bebop [2] in most examples).

Many other researchers have studied and provided solutions for the infinities raised by data. We only mention here the work of a group in Liège around Pierre Wolper working on integer and real variables.

The automata-theoretic approach faces now three major challenges:

Integration of techniques for control and data. Due to the difficulty of the problems, groups working on handling infinite data structures (or variables with finite but large domains) have so far restricted their attention to programs with very simple control structures, basically guarded commands. Similarly, groups working on control problems (like us) have so far work on programs or abstractions of programs whose variables have small finite domains. While model-checking programs without any kind of constraint is unrealistic, in practice one would often like to abstract away or finitise the range of all integer variables but a few, or replace a queue by its length; this could increase the precision of the analysis and prove the property at hand.

Extension to multithreaded programs. Communication and concurrency are pervasive in a large number of software applications, in particular in the area of dependability. However, extending procedural programming with multithread features poses formidable problems, since the combination of recursive procedures and communication is known to yield a Turing powerful control structure. In consequence, even pure control analysis problems, in which data is completely abstracted away, are undecidable. It is therefore necessary to analyse special cases and find reasonable abstractions to make the problem tractable.

Application to real programming languages, and large programs In order to provide a high degree of automation, model-checkers must be able to work directly on the original code. (The programmer may be requested to add information about which code can be abstracted away, but he or she cannot be requested to rewrite the code in a low level language adequate for the model-checker.) Moreover, in order to deal with large programs, checkers must be able to adapt the basic algorithms to the nature of the code being analysed.

The second of this challenges addresses a fundamental, so far unexplored problem. No foundational approaches to the analysis of communicating procedural programs exist yet; only negative results have been reported in the literature (see for instance [13]).

The first and third challenges consist of developing an existing verification framework (or the one to be developed by attacking the second challenge). They require to apply and develop technology in the areas of algorithms, data structures, and software engineering.

Overall aims

The overall aim is the development of a model-checker for multithreaded programs with procedures, accepting real programs (usually obtained from more complex programs by means of an automatic abstraction process) as inputs. Due to the undecidability of the model-checking problem in this setting, such a checker is not be guaranteed to terminate in general. Precise information will be provided about the cases in which termination is guaranteed and the associated running times. The checker will be used to search for bugs in large real programs.

Objectives

The objectives are to provide solutions to the three challenges mentioned in the background section, and use them to build the model-checker of the overall aim. More specifically, the objectives are

- To integrate symbolic techniques for possibly recursive procedures with symbolic techniques for handling data.

- To extend the existing theory and implementations to multithread programs with communication between threads.
- To apply the results to the problem of finding bugs in large C and Java programs. (The reasons for choosing both C and Java are explained in the Methodology section.)

Methodology

We describe the methods we will use to attack the three goals stated in the last section

Integration of techniques for control and data

Work on this problem will proceed along two lines:

BDD techniques. We have already done some work along this line. In [9], BDDs have been applied to model data with a finite but possibly large range. Experiments carried out with MOPED, a prototype implementation, show that the verification times are very sensitive to issues like: the ordering of BDD variables, in particular with respect to global and local program variables; the choice of algorithm to detect cycles in the state space, required for checking liveness properties; the strategy used to compute the effects of procedures (eager or lazy), and the particular combination of backward and forward search used to solve reachability problems. These issues and the resulting trade-offs will be analysed in detail, both theoretically and experimentally. We will perform experiments with new versions of MOPED to evaluate the results.

Automata-theoretic approaches. In parallel, we will also work on the use of symbolic techniques for manipulating data with an infinite range. Of special interest for this proposal is the work done in [4] on an automata-theoretic approach for integer variables. (It is important to notice that the addition of integer variables makes the model-checking undecidable even for plain while programs. The technique of [4] allows to compute a finite representation of an approximation of the set of reachable states, which can be exact sometimes.) The fact that this technique is also based on automata on finite or infinite words provides a common ground for the integration. The integration of heterogeneous systems is difficult because performance improvements on one part do not extend to others; moreover, communication between the parts usually requires to write converters between different representations of data, leading to high efficiency loses. On the contrary, in an automata-based approach, advances made on the implementation of data structures and/or algorithms for the manipulation of automata may improve performance at many different points.

Extension to multithreaded programs

In two papers by the principal investigator and one of his Ph.D. students, some results have been obtained which open the way to two possible attacks on the problem. In the project, we will explore both of them.

In [11] it has been shown that some important verification problems for a model of computation called Process Rewrite Systems (PRSs) are decidable. It turns out that PRSs can model control structure with both recursive procedures and communication, as long as communication is done exclusively through "local channels". Just as in the case of local variables, the local channels of a called procedure are not visible to the caller. While local channels are not a realistic model of communication, in the sense that they are not used by multithread programming languages, they can be seen as a very good approximation. Moreover, the accuracy of the approximation can be arbitrarily improved at the expense of computational resources. The main problem of this approach is analysability: The algorithms derived from the results of [11] have a very high time and space complexity, and are very complicated themselves. We will search for better algorithms.

In [8], the efficient algorithms of [6] have been extended to programs with multiple threads, but without thread synchronisation. In this case, and due to the different threads, states are naturally represented by trees. Tree automata are used to symbolically represent possibly infinite sets of states. Surprisingly, the complexity of the algorithms is very similar to that of the purely sequential case. We will investigate techniques for refining the analysis and removing spurious behaviours, incompatible with the constraints introduced by communication between threads. In particular, we will investigate approximation techniques allowing to remove more and more behaviours at the expense of more and more computational resources.

Application to C and Java programs

Through our past work [7] and a recent cooperation with Tom Ball and Sriram Rajamani of Microsoft Research, Redmond, we have identified two possible applications of our model-checker.

Finding bugs in drivers for Windows XP. In preliminary experiments carried out together with Ball and Rajamani MOPED has found bugs in drivers containing 10000 lines of C code within minutes. Drivers are an excellent application because they may have to be checked in large numbers, and contain a fair number of procedures (which cause problems to standard checkers).

Verifying security properties of Java programs. The Java Development Kit (JDK 1.2) introduced a new security model based on *protection domains*. Every class is assigned to a protection domain, and every domain has associated permissions. The `CheckPermission` method checks if a permission is granted (according to a certain policy) and raises an exception if not. Several interesting security properties of this model (for instance, if the calls to `CheckPermission` inserted by the programmer in the code will actually catch all security violations or not) can be formalised as invariants of the stack of activation records. In [7] (and inspired by previous work by Jensen and LeMétayer) we have extended our model-checking techniques to deal with this problem.

We will do work towards these two applications. The first one is firmly established as a good candidate for industrial impact. The second one is chosen for its own potential and also for its perfect match with the functionality of our model-checker: The algorithms developed by our competitors are unnatural for applications requiring to compute the stack contents reachable along the possible computations.

In order to deal with these applications, the model-checking process must proceed in three steps: (1) conversion of the program into an adequate intermediate representation; (2) simplification of the input and customisation of the checker for the problem at hand; and (3) actual check, and in case the property fails, display of information allowing to analyse the failure. We describe our approach to these three steps separately.

Conversion of the program This task is very demanding, and carrying it out for two languages like C and Java clearly exceeds the personnel resources requested for this project. Fortunately, we will profit from work done by colleagues with whom we have good cooperation links. Within the SLAM project, researchers at Microsoft in Redmond have developed a converter mapping C programs (after an abstraction procedure) onto so called boolean programs, a language close to MOPED's front-end. (The experiments on drivers described above have been carried out using boolean programs as intermediate language.) The BANDERA project at Kansas University has developed a set of tools allowing to transform Java programs into simpler internal representations. Finally, a converter from a subset of Java into pushdown automata, again close to MOPED's front-end, has been described in [12], whose author is now a Ph.D. student at Edinburgh.

Simplification and customisation Since the intermediate representation is produced automatically, it may contain many redundancies. Moreover, the characteristics of the input may differ widely. In our experiments we have identified three important points to be addressed: application of program analysis techniques, like removal of dead code; use of locality (the fact that most assignments only affect a small number of variables) to reduce the size of symbolic representations;

adaptation of the type of search (breadth-first or depth-first) according to the characteristics of the program (control-intensive or data-intensive).

Check and failure reports We will explore the use of heuristic searches to return short counterexamples, as well as the best way to check if the computed counterexample can be executed (i.e., whether it is a real counterexample or an artifact caused by abstracting part of the program).

Timeliness

Software model-checking and, more generally, the verification of infinite-state systems are extremely active research topics. This is witnessed, e.g., by the large number of recent workshops or special sessions explicitely devoted to them (SPIN 2001 Workshop on Model Checking of Software, Software-Model Checking Workshop in CAV 2001, Session in PLDI 2001, INFINITY Workshop in CONCUR 2002).
As mentioned in the Background section, our research takes place in a very competitive environment, with groups at Bell Labs and Microsoft Research actively working on verification of programs with procedures [1, 3], and well-known researchers in academia contributing to the topic (see for instance [10]). The extension to communicating systems is also a hot topic, and undoubtedly our competitors are also working towards this goal. Due to our preliminary work [8, 11], we possibly have the leading position at this moment.

Relevance to beneficiaries

The immediate beneficiary of this research is the model-checking community, which has followed with interest and appreciation the developments in model-checking recursive programs, as shown by the large number of recent publications on this topic (see the list of references).

The model-checker produced by the project will be of interest to a group of software engineers in charge of some specialised certification tasks, like the certification of drivers mentioned before in this proposal: Since operating systems are distributed with a possibly large set of drivers, programmed by many different people, the distributors need to gain confidence in their correctness; a pass through our model-checker will use few human resources due to the high degree of automation, and will have a feasible computational cost. So the checker will only have to be able to find a few relevant bugs to make the process cost effective.

In the longer term, model-checkers like ours should find their place in development environments and be used by many programmers of dependable systems.

Dissemination of results

During the project we will publish our results in the form of reports and articles, which will be submitted to international conferences and journals. We will also deliver an implementation of our model-checker, which will downloadable from the Web with an open source policy.

Justification of resources

People Two RA positions are requested (see the Workplan section). For timeliness reasons (see the Timeliness section), the positions cannot be filled with Ph.D. students, who would need about one year of training. One of the two positions will be filled by Stefan Schwoon, who will complete his Ph.D. Thesis in March 2002 on model-checking programs with possibly recursive procedures. Schwoon has done excellent work both at the theoretical and at the implementation levels, with over proportional contributions to [6, 7, 9]. The main investigator will contribute 6 hours per week.

Travel We will visit research groups working in related areas, both in Europe and overseas. This includes researchers at Bell Labs New Jersey and Chicago, Microsoft Redmond, the Weizmann Institute in Israel, and at the universities of Oxford, Southampton, Aachen, Dortmund, Munich, Saarbrücken, Uppsala, Paris, and Rice. We are asking for funding a number of trips within the UK, to the rest of Europe and overseas, for both the principal investigator and the two RAs. We also seeks costs por participating in relevant workshops and conferences, like CAV, CONCUR, FSE, ICCAD, ISSTA, ICALP, LICS, PLDI, POPL, SAS, SPIN Workshop, TACAS.

Equipment and support The experimental work of the project requires a desktop with a 1.5 GHz processor and 2 GB of memory. We also request a standard laptop for use of the RAs.

We request 25% of a computing officer to cover direct support of the machines, the staff component of the Divisional infrastructure charge, and for support of the tool development; admin support at 15% of a secretary; networking and fileserver charges as set by the Division; and a contribution to books and consumables.

References

[1] R. Alur, K. Etessami and M. Yannakakis: Analysis of Recursive State Machines. CAV '01, LNCS 2102, 207–220 (2001).

[2] T. Ball, S.K. Rajamani: Bebop: A Symbolic Model Checker for Boolean Programs. SPIN '00, LNCS 1885, 113–130 (2000).

[3] M. Benedikt, P. Godefroid and T.W. Reps: Model Checking of Unrestricted Hierarchical State Machines. ICALP '01, LNCS 2076, 652–666 (2001).

[4] B. Boigelot, S. Jodogne and P. Wolper: On the Use of Weak Automata for Deciding Linear Arithmetic with Integer and Real Variables. IJCAR '01, LNCS 2083, 611–625, (2001).

[5] A. Bouajjani, J. Esparza and O. Maler: Reachability Analysis of Pushdown Automata: Application to Model-Checking. CONCUR '97, LNCS 1243, 135–150 (1997).

[6] J. Esparza, D. Hansel, P. Rossmanith and S. Schwoon: Efficient Model Checking Algorithms for Pushdown Systems. CAV '00, LNCS 1855, 232–247 (2000).

[7] J. Esparza, A. Kučera and S. Schwoon: Model-Checking LTL with Regular Valuations for Pushdown Systems. TACS '01, LNCS 2215 316–339 (2001).

[8] J.Esparza and A. Podelski: Efficient algorithms for pre^* and $post^*$ on Interprocedural Parallel Flow Graphs. POPL '00, 1–11, ACM Press (2000).

[9] J. Esparza and S. Schwoon: A BBD-Based Model Checker for Recursive Programs. CAV '01, LNCS 2102, 324–336, (2001).

[10] O. Kupferman and M.Y. Vardi: An Automata-Theoretic Approach to Reasoning about Infinite-State Systems. CAV '00, LNCS 1855, 36–52, (2000).

[11] R. Mayr: Process Rewrite Systems. Information and Computation 156(1-2), 264–286 (2000)

[12] Jan Obdržálek: Formal Verification of Sequential Systems with Infinitely Many States Master's Thesis, Masaryk University (2001).

[13] G. Ramalingam: Context-sensitive synchronization-sensitive analysis is undecidable. TOPLAS 22(2): 416-430 (2000).

Workplan

We provide a list of expected outcomes to be achieved at three milestones. We also include the estimated effort in personmonths (PM). Notice that (1.3) and (2.3) are new ground, and so the estimated efforts are rather speculative.

Milestone 1: 12 months

(1.1) An optimized implementation of a BDD model-checker (called MOPED-C in the sequel) for the analysis of C code. A detailed study of the optimal choice of the checker's parameters depending on the characteristics of the input. (8 PM)

(1.2) A prototype checker (called MOPED-J in the sequel) for a subset of Java, including exception handling, based on [12]. (Requires to extend the results and models of [12], and write a converter from the Java subset to MOPED's abstract description language.) (4 PM)

(1.3) An automata-theoretic approach to the computation of successive approximations to the set of backward and forward reachable states of systems modelled in an extension of boolean programs with parallelism and communication primitives. Study of the worst-case complexity. (14 PM)

Milestone 2: 27 months

(2.1) Improvement of the failure reports of MOPED-C. Experimental results on the application to the verification of drivers. Modifications to MOPED-C according on the results. (9 PM)

(2.2) An extension of MOPED-C and MOPED-J with automata-theoretic techniques for manipulation of integer data, yielding MOPED-C-INT and MOPED-J-INT for the analysis of C and Java code. Experimental results on sorting algorithms and other academic examples. (9 PM)

(2.3) Development of adequate data structures and algorithms for (1.3). An implementation of (1.3) as an extension of MOPED (called MOPED-COMM in the sequel). (14 PM)

Milestone 3: 36 months

(3.1) Experimental results on the application of MOPED-C-INT to the verification of drivers. Optimisation of MOPED-C-INT (4 PM)

(3.2) Application of MOPED-COMM to case studies in client-server architectures applications. Study of the optimal choice of parameters. Improvement of failure reports allowing to trace the contributions of different processes to the global execution. (10 PM)

(3.3) Integration of (3.1) and (3.2). Final case studies. (5 PM)