

Infinite State Model Checking using Partial Evaluation and Abstract Interpretation

Final Report for GR/N11667/01
(Accompanies Individual Grant Review Form)
Michael Leuschel, University of Southampton

1 Background/Context

Logic programming originated from the discovery that a subset of predicate logic could be given a procedural interpretation which was first embodied in the programming language Prolog. The unique features of logic programming make it appealing for numerous applications in artificial intelligence, computer-aided verification, etc. *Program specialisation*, also called *partial evaluation*, is an automatic tool for program optimisation, similar in concept to, but in several ways stronger than highly optimising compilers. The central idea is to specialise a given source program for a particular application domain. *Program analysis* is about statically inferring information about dynamic program properties. *Abstract interpretation* was developed as a very general, formal framework for specifying and validating program analyses.

The past years have seen dramatic growth in the application of *model checking* techniques to the validation and verification of hardware systems. The main idea is to model a hardware or software system as a finite, labeled transition system (LTS) which is then exhaustively explored to decide whether a given specification holds for all reachable states. Recently there has been interest in applying *logic programming techniques to model checking*. E.g., table-based logic programming can be used as an efficient means of performing explicit model checking. Furthermore, logic programming provides a good basis to formulate many other specification formalisms (a claim which we have further investigated in this project).

However, despite the success of model checking, most systems must be substantially simplified (i.e., *abstracted*) before they can be model checked and substantial human intervention is usually required for this process. Furthermore, most *software* systems cannot be modeled directly by a *finite* state system. For these reasons, there has recently been considerable interest in *infinite model checking*. This, by its very undecidable nature, is a daunting task, for which *abstraction* is a key issue as it may allow one to approximate an infinite system by a finite one.

Overall Goal This research aims at exploring automatic means of building precise but tractable abstractions for infinite model checking (or model checking of finite, but very complex systems). We propose to do this by extending technology that has been developed to tackle similar problems in the context of automatic logic program analysis and specialisation. Indeed, a major concern of partial evaluation is the *automation of control and abstraction*, and, in the current state-of-the-art, this has obtained a level of refinement which goes beyond mere heuristic strategies. We will study to what extent we can leverage this technology for infinite model checking. Finally, it is often felt that there is a close relationship between abstract interpretation and program specialisation and, recently, there has been a lot of interest in integrating these two techniques.

2 Key Advances and Supporting Methodology

Verification Methodology In the course of this project we have developed the following

methodology to tackle model checking tasks by logic programming and partial evaluation technology. In essence, we:

- model a system to be verified as a logic program. This obviously includes finite labeled transition systems, but also allows to express systems with an infinite number of states. Note that this translations is often very straightforward, due to the built-in support of logic programming for non-determinism and unification.
- model the specification formalism as a logic program interpreter acting on the representation above. This interpreter can, e.g., make use of the tight link that exists between the semantics of logic programs and least-fixed points.
- and then either
 - apply tabling based Prolog system (XSB Prolog) to perform explicit finite state model checking
 - or try to *automatically derive abstractions for infinite model checking through a combination of partial evaluation and abstract interpretation technology*. The tools LOGEN and ECCE [5] are used successfully to automate the work.

This methodology is appealing because of its generality (we can adapt it to any formalism expressible as a logic program) and in the way it re-uses existing logic programming technology. In this project we have shown that this approach is also theoretically sound, complete for certain classes of problems, and also surprisingly efficient.

Study of the potential of the method In [12] we have shown the usefulness of the above approach. We have developed a complete interpreter for CTL (Computation Tree Logic) formulae and we have shown it to be correct (under the SLS/well-founded semantics), even for infinite state systems. Contrary to other works our aim was not maximum efficiency, but writing a provably correct interpreter that can be fed into existing tools for the analysis and optimisation of logic programs, which work best on declarative programs.

We have shown how this interpreter can be used for finite state model checking using tabling-based execution. We also have presented a particular technique for infinite state model checking of safety properties, using existing techniques for partial deduction and abstract interpretation, as implemented in the ECCE system. We discussed how this approach has to be extended to handle more complicated infinite state systems and to handle arbitrary CTL formulae. We presented some successful examples but argue that more refined treatment of negation and more refined abstract domains will be required for the method to scale up to such systems and properties.

Theoretical study of power In [11] we gave a first formal answer about the power of our approach and showed that when we encode ordinary *Petri nets* as logic programs and use existing program specialisation algorithms, we can decide so-called “*coverability problems*” (which encompass quasi-liveness, boundedness, determinism, regularity,...). This was achieved by showing that Petri net algorithms by Karp–Miller and Finkel can be exactly mimicked, thus also establishing a surprising link between program specialisation methods and existing algorithms from the Petri net area.

In [10] we have examined the power of partial evaluation (and abstract interpretation) for another particular class of infinite state model checking tasks, namely *covering problems for reset Petri nets*. The latter are particularly interesting as they lie on the “border between decidability and undecidability”. We have proven that again partial evaluation can be used as a decision procedure for these problems.

We have also shown that this result holds for other Petri net extensions which can be viewed as so-called WSTS’s (well structured transition systems). We have also studied other WSTS’s from the process algebra arena. For these we have shown that, to arrive at a full-fledged decision procedure, we will need to move to the more powerful *abstract*

partial deduction framework, which integrates partial evaluation of logic programming with abstract interpretation.

Infinite state model checking for process algebras Based on the insights from [10], we continued work on our *abstract partial deduction* framework [6] and then developed a new concrete abstract partial deduction technique which uses so-called *regular types* as its domain [9]. We provided a detailed description of all the required operations and developed an implementation within the ECCE system. We discuss the power of this new specialisation algorithm, especially in the light of verifying and specialising infinite state process algebras. Here, our new algorithm can provide a more precise treatment of synchronisation and can be used for refinement checking.

The thus extended version of ECCE is available as a free download [5], and we have also developed a more user-friendly graphical front end for the system (cf, e.g., Fig. 1).

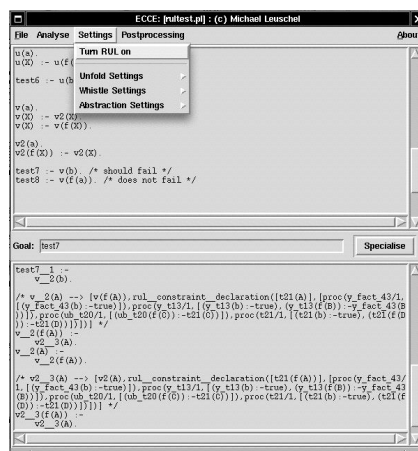


Figure 1: The ECCE system with the new RUL package for regular types

Empirical evaluation In [13] we have empirically evaluated our approaches on various case studies of finite, parameterised and infinite systems (e.g., a Central Server Model and a Flexible Manufacturing System), taken from the Babylon benchmark library. For finite state systems, we show how our approach and tool compares to the standard tools for finite state model checking SPIN and FDR. Our experiments show that the efficiency is surprisingly good, for such a simple, extensible and provably correct system. In fact, our CTL model checker running on XSB-Prolog 2.4 performed better than FDR and only slightly slower than SPIN for most experiments. For parameterised or infinite state model checking, we compared our ECCE system with HYTECH and the Covering-Sharing-Tree method by Raskin et al. Results were again surprisingly good, especially since ECCE has initially not been designed with model checking in mind.

Modeling more high-level formalisms In [7], [14] and [8] we have tried to demonstrate how Prolog technology can help us develop (and adapt) animators and verifiers for more high level specification languages such as CSP and B. We have highlighted how one can obtain specialised model checkers using partial evaluation technology. We have also discussed the possibility of using constraint logic programming to do symbolic animation and model-based model checking. Prototype verification systems were developed using SICStus Prolog and XSB Prolog. In summary, our research has shown that the latest generation Prolog systems are very well suited to prototype and develop high-level, practical specification languages.

Application to the Fluent calculus The Fluent Calculus (FC) is a formalism to reason about action and change. In an unanticipated cross-fertilization we were able to use our insights above from [11] to establish decidability results for planning problems in the FC [2]. Based on our prototype systems, we also developed in [3] an abstract partial deduction method capable of actually solving planning problems in the FC.

Study control techniques in inductive theorem proving In [4] we studied how an inductive theorem proving approach to verification can be compared to our approach in the context of our work on the coverability analysis of Petri nets [10]. We have tried to use the theorem prover ISABELLE to automatically perform model checking of the infinite state Petri nets from [10]. Except for very simple systems, these experiments were unfortunately not successful. We conjecture that the added possibilities in the setting of general theorem proving make automatic control much more difficult (compared to partial evaluation). However, we were able to develop a tool to systematically translate the output of ECCE into an explicit inductive proof for ISABELLE. We were then able to use ISABELLE to check that the verification performed by ECCE is indeed sound.

Systems The systems LOGEN and ECCE have been made available for download at <http://www.ecs.soton.ac.uk/~mal/systems/>). During this project we have also given both systems a more user-friendly graphical interface, and have provided pre-configured settings for model checking. Specific prototypes for animating and verifying CSP and B have also been developed.

All papers are available for download at: <http://www.ecs.soton.ac.uk/~mal/ISM.htm>.

3 Project Plan Review

We repeat the 7 individual tasks:

1. *Theoretical requirements study for doing infinite model checking in a logic programming setting.* We have characterised the power of our approach and compared it to existing techniques for handling infinite state systems [11, 10], we have identified shortcomings of the first prototype for process algebras and for handling liveness properties [12]. Our experiments have shown that tabling as provided by XSB Prolog already yields an efficient way of handling large state spaces, and it was thus not necessary to investigate how BDD-like representations could be integrated.
2. *Development of a combined partial evaluation and abstract interpretation system.* A first prototype system, based upon ECCE was developed and it was possible to validate the theoretical claims of [11, 10].
3. *Apply the system to small to medium-sized examples.* Very first experiments were carried out in [12, 11, 10]. A practical case study was investigated before the start of the project in [1]. These initial, informal experiments underlined the potential of our method and warranted further, more in depth investigations. In [7, 8] further small experiments were conducted on more realistic domain specific languages and specification languages.
4. *Study control techniques in inductive theorem proving.* This was achieved in [4]. While using the existing theorem prover ISABELLE as is turned out to be impractical, we managed to convert output from our system so as to control ISABELLE.
5. *Address the limitations identified in task 3 and incorporate insights from tasks 1 and 4.* Building upon the further developed framework [6], and based upon our insights, we have developed an extension of ECCE in [9].
6. *Tackle a larger, more realistic example of software verification.* In [13] we performed a full fledged empirical investigation on infinite state model checking benchmarks from the Babylon suite.

7. The *final evaluation and report*. In addition to the present report, we also maintain a web site (see below) from where all publications and systems will be available.

We believe that we have achieved the three criteria for success of the project: **1.** the delivery of a working system has been achieved, **2.** the validation of the system and the theoretical feasibility of the approach on medium-size examples has been completed, and **3.** the evaluation of the practicality of the approach along with discovery of problems and solutions has also been performed.

4 Research Impact and Benefits to Society

First, our project has put existing specialisation and analysis technology on the testbed and new insights about the power and complexity of existing program specialisation tools have been gained. We also believe to have stimulated further research, notably from the unfold/fold area. Hence, we believe that researchers in program analysis and transformation in the first instance, and users of program specialisation and analysis tools in the second instance have benefited from this project.

We also believe that our proposed methodology, because it can be quickly adapted to new specification languages, has benefits to people and companies developing (safety critical) software systems. First contacts have been made with Formal Systems Europe Ltd. concerning our tools to animate and verify CSP. We believe that our technology will be useful for treating high-level specifications written in B, and in a new project proposal (AMBER) we strive to apply our technology to medical knowledge expressed in the Proforma language. We also plan to apply our technology within our EPSRC funded project ABCD.

Finally, I believe that this project has achieved some cross-fertilisation of the fields of software verification, program analysis and specialisation, and inductive theorem proving.

5 Explanation of Expenditure

Expenditure on salaries has been largely as planned, despite some problems finding suitable staff for the project. We spent more on hardware, consumables, and especially travel than originally budgeted, effectively using the unspent salary money. The large travel money reflects the high number of publications (with ensuing presentations and travel to conferences) achieved under the project. The extra expenses on consumable are due to the extra dissemination activities related to the VCL workshop series (see below).

6 Further Research or Dissemination Activities

In addition to presenting all the conference papers at mainly international events, we have also further disseminated the idea of using computational logic in general and logic programming tools in particular for verification by founding the VCL (Verification and Computational Logic) workshop series and editing the proceedings of VCL'2000, VCL'2001 (<http://www.ecs.soton.ac.uk/~mal/vcl2000.html> and [vcl2001.html](http://www.ecs.soton.ac.uk/~mal/vcl2001.html)), and also of VCL'2002 (<http://www.dsse.ecs.soton.ac.uk/VCL2002/>). In the same line we are also in the process of editing a special issue on Verification and Computational Logic in the journal "Theory and Practice of Logic Programming," Cambridge University Press (<http://www.ecs.soton.ac.uk/~mal/tp1p/>). Finally, we have met up with fellow researchers in Oxford, Bristol, and Munich, and have given invited seminars there, presenting and discussing results from the present project.

References

- [1] P. Hartel, M. Butler, A. Currie, P. Henderson, M. Leuschel, A. Martin, A. Smith, U. Ultes-Nitsche, and B. Walters. Questions and answers about ten formal methods. In S. Gnesi and D. Latella, editors, *Proceedings of FMICS'99*, pages 179–203, Trento, Italy, July 1999. Extended version as technical report DSSE-TR-99-1, Department of Electronics and Computer Science, University of Southampton.
- [2] H. Lehmann and M. Leuschel. Decidability results for the propositional fluent calculus. In J. Lloyd, editor, *Proceedings of the International Conference on Computational Logic (CL'2000)*, LNAI 1861, pages 762–776, London, UK, 2000. Springer-Verlag.
- [3] H. Lehmann and M. Leuschel. Solving planning problems by partial deduction. In M. Parigot and A. Voronkov, editors, *Proceedings of the International Conference on Logic for Programming and Automated Reasoning (LPAR'2000)*, LNAI 1955, pages 451–468, Reunion Island, France, 2000. Springer-Verlag.
- [4] H. Lehmann and M. Leuschel. Generating inductive verification proofs for Isabelle using the partial evaluator Ecce. Technical Report DSSE-TR-2002-2, Department of Electronics and Computer Science, University of Southampton, UK, September 2002.
- [5] M. Leuschel. The ECCE partial deduction system and the DPPD library of benchmarks. Obtainable via <http://www.ecs.soton.ac.uk/~mal>, 1996-2002.
- [6] M. Leuschel. Logic program specialisation and top-down abstract interpretation reconciled. Technical Report DSSE-TR-2000-3, Department of Electronics and Computer Science, University of Southampton, May 2000.
- [7] M. Leuschel. Design and implementation of the high-level specification language CSP(LP) in Prolog. In I. V. Ramakrishnan, editor, *Proceedings of PADL'01*, LNCS 1990, pages 14–28. Springer-Verlag, March 2001.
- [8] M. Leuschel, L. Adhianto, M. Butler, C. Ferreira, and L. Mikhailov. Animation and model checking of CSP and B using prolog technology. In *Proceedings of VCL'2001*, pages 97–109, Florence, Italy, September 2001.
- [9] M. Leuschel and S. Gruner. Abstract partial deduction using regular types and its application to model checking. In A. Pettorossi, editor, *Proc. of 11th Int'l Workshop on Logic-based Program Synthesis and Transformation, LOPSTR'2001*, LNCS 2372, pages 91–110, Paphos, Cyprus, 2001. Springer-Verlag.
- [10] M. Leuschel and H. Lehmann. Coverability of reset Petri nets and other well-structured transition systems by partial deduction. In J. Lloyd, editor, *Proceedings of the International Conference on Computational Logic (CL'2000)*, LNAI 1861, pages 101–115, London, UK, 2000. Springer-Verlag.
- [11] M. Leuschel and H. Lehmann. Solving coverability problems of Petri nets by partial deduction. In M. Gabbriellini and F. Pfenning, editors, *Proceedings of PPDP'2000*, pages 268–279, Montreal, Canada, 2000. ACM Press.
- [12] M. Leuschel and T. Massart. Infinite state model checking by abstract interpretation and program specialisation. In A. Bossi, editor, *Logic-Based Program Synthesis and Transformation. Proceedings of LOPSTR'99*, LNCS 1817, pages 63–82, Venice, Italy, 2000.
- [13] M. Leuschel and T. Massart. Logic programming and partial deduction for the verification of reactive systems: An experimental evaluation. In G. Norman, M. Kwiatkowska, and D. Guelev, editors, *Proceedings of AVoCS 2002, Second Workshop on Automated Verification of Critical Systems*, pages 143–149, Birmingham, UK, 2002. Available as Technical Report CSR-02-6, University of Birmingham.
- [14] M. Leuschel, I. Wolton, T. Massart, and L. Adhianto. Temporal model checking of csp: Tools and techniques. In D. Nowak, editor, *Proceedings of the Workshop on Automated Verification of Critical Systems (AVoCS 2001)*, Oxford, UK, 2002. Available as Technical Report PRG-RR-01-07, Oxford University Computing Laboratory.