Introduction for Special Issue of JSC on Invariant Generation and Advanced Techniques for Reasoning about Loops

Martin Giese

Department of Computer Science
University of Oslo, Norway

Andrew Ireland

School of Mathematical and Computer Sciences
Heriot-Watt University, UK

Laura Kovács

Faculty of Informatics
Vienna University of Technology, Austria

Many groups around the world conduct research on formal methods for software development, and in most of these groups, some of the effort goes into the problem of reasoning about loops. There is of course a well-known classic way of dealing with loops, namely by having the software developer provide an invariant, which can be shown to be preserved by the loop. However, experience shows that writing these invariants can be difficult in some cases, trivial but tedious in others, unnecessary in still others. The hope of obviating the need for hand-written invariants, or at least simplifying their production, is what keeps the research on loops alive.

Research results tend to be presented at general conferences on formal methods, logic, or mathematics, usually depending on their scope and the methods employed, but this means that researchers using different approaches often do not know about each others' work. To improve this situation, we decided in 2007 to organize a Workshop on Invariant Generation with the goal of bringing together people interested particularly in reasoning about loops. The WING 2007 workshop was a satellite workshop of the Calculemus conference held in Hagenberg, Austria.

Email addresses: martingi@ifi.uio.no (Martin Giese), a.ireland@hw.ac.uk (Andrew Ireland), lkovacs@complant.tuwien.ac.at (Laura Kovács).

Based on the success of the kick-off WING workshop in 2007, follow-up workshops have been initiated and organized. WING 2009 was held as part of the ETAPS 2009 conference in York, UK. This year, WING 2010 is affiliated with the IJCAR 2010 conference, and will take place as a satellite workshop of the FLoC 2010 conference in Edinburgh, UK.

Initially, we feared that calling for papers about loop reasoning in general might attract submissions about any work having to do with program verification, on the grounds that reasoning about loops is the theoretically most challenging aspect of deductive program verification. Should the call for papers be restricted to invariant generation techniques, or even to algebraic techniques? It turned out that keeping the call for papers open was the right thing to do, since it attracted an interesting variety of papers, almost all of them quite specific to loop reasoning.

Following the WING 2007 and 2009 workshops, we issued a call for contributions to the present special issue. We have received a great number of highly-original contributions. The selection process, based on peer-reviewing, was very competitive, and resulted in the acceptance of seven research papers targeting various aspects of loop reasoning. We believe that with this special issue of the Journal of Symbolic Computation, we have made a step toward a common forum for an interdisciplinary field bringing together researchers from formal methods, computer algebras, computational intelligence, software engineering, etc.

Finally, we wish to thank the Editor-in-Chief, Prof. Hoon Hong, for accepting to publish this issue as a special issue of the Journal of Symbolic Computation.

A Short Overview of Advanced Methods for Reasoning About Loops

The techniques for reasoning about loops presented in the submissions to the WING workshops and this special issue are quite diverse. We start by pointing out a few principal differences between methods that may be used as a basis for a classification.

One distinction that can be made is that some methods are geared towards particular data types, i.e. usually numbers, while others use more general techniques that can be applied to arbitrary data types, in combination of theories.

Another observation is that some techniques try to derive automatically a complete functional specification of a program, i.e. an invariant that allows to compute the value of each variable after completion of the loop, and maybe after each iteration, while other approaches are heuristics that discover some of the more obvious invariants, but which can be enough to prove absence of null-pointer errors, overflows, type errors, etc.

Finally, while some methods are targeted at a very general form of loops, like general `while`-loops, other methods identify particular shapes of loops, e.g. certain typical kinds of `for`-loops, and perform reasoning only for those.

If a loop works only on numerical variables, methods from computer algebra can be applied. In this category we find for instance the work of Letichevsky and Lvov (1993) or Lvov (2007) which goes back to the work of various researchers at the Institute of Cybernetics at the National Academy of Sciences of Ukraine in the late 1970ies. The essential idea is to restrict oneself to numerical variables and field operations, which give polynomial relationships between variable values. Techniques from computational algebra, like Gröbner bases, are then used to derive invariants. Other work in a similar vein includes that of Müller-Olm and Seidl (2004), that of Sankaranarayanan et al. (2004),

and that of Rodríguez-Carbonell and Kapur (2004, 2007). A slightly different approach is taken in the work of Kovács (2008), which builds on various existing methods for solving sets of recurrence relations.

Olsson and Wallenburg (2005) also concentrate on numerical variables, but they analyse loop code to derive a custom induction rule for cases where induction over the standard ordering is not appropriate.

A method aiming at general data structures implemented using pointers has been proposed by Ireland (2007). It makes use of the rippling technique (see Bundy et al. (2005)) which was originally developed to search for inductive proofs about inductive data types.

Another approach, which has been proposed by Leino and Logozzo (2005, 2007), consists in refining loop invariants by analysing failures to prove loop correctness. The method assumes that an SMT (satisfiability modulo theories) prover with a certain structure is used, and uses this knowledge about the prover's inner workings to analyse proof failures and to derive possible invariant strengthenings.

A technique used in the extended static checker ESC/Java (see Kiniry and Cok (2005)) to generate comparatively weak invariants from assertions has been described by Janota (2007). Here, assertions guaranteeing absence of null pointer exceptions or array index range exceptions are propagated from the critical statements to the rest of the program.

Various research has gone into the treatment of certain common patterns. For instance, Gedell and Hähnle (2006b,a) have investigated the verification of parallelisable `for`-loops, i.e. loops where the iterations are independent of each other.

Another approach to the verification of `for`-loops is found in the work of Kauer and Winkler published in this special issue. This work is based on an analysis of the loop's pre- and postcondition (which need to be provided), by using heuristics like replacing a constant in the postcondition by one of the loop variables. The method turns out to be effective for a large part of the `for`-loops occurring in practical programs.

The work of Mili et al. which is also published in this special issue starts from a new fundamental idea in that it considers the relation between two states that are separated by an arbitrary number of loop iterations. Instead of investigating loop invariants, which are predicates on program states, these "reflexive, transitive invariant relations" are considered. After a general analysis of the properties of these objects, Mili et al. give a number of program patterns which again may be used to derive descriptions of invariant relations for commonly occurring cases.

The utility of invariant generation techniques is not limited to sequential programs.

The article of Konnov and Zakharov in this issue considers the problem of verifying properties of systems of parallel processes, where properties are required to hold for systems of arbitrarily many copies of the same process. Such a proof can make use of an invariant argument, and Konnov and Zakharov discuss ways of generating invariants for this task.

The method of Luo, Crãciun, Qin and He included in this special issue is demonstrated on recursive, rather than loop-based code, but essentially the same challenges of generating loop invariants are present. Their work focuses on the automatic generation of program specifications within the context of Hoare-style partial correctness specifications. They specifically target shape properties with respect to heap based data structure,

building upon separation logic and bottom style of abductive reasoning. While shape is quire specific, it also always important properties to be specified, such as the absence of memory leaks.

A complementary paper by Moy and Marche, published in this special issue, investigates an integrated approach to specification generation which combines both forward and backward styles of analysis. Like the work of Luo, Craciun, Qin and He, this paper promotes a modular approach to verification. Their work is applicable to reasoning about arrays, and pointer based programs with aliasing, although they do not consider shape properties.

Recent methods for software verification use interpolation Craig (1957) as a "cheaper" approach to predicate refinement and loop invariant generation. For example, employing resolution proof in quantifier-free theories, Henzinger et al. (2004); Jhala and McMillan (2006) extracts interpolants as quantifier-free loop invariants. Further, using automated first-order theorem provers, in McMillan (2008) quantified invariants as interpolants are generated by extending first-order theory reasoning with additional heuristics, whereas in Kovacs and Voronkov (2009a,b) quantified interpolants and invariants are inferred using both first-order superposition and theory reasoning. Quantified interpolants are also generated in Christ and Hoenicke (2010) - the approach relies on instantiation-based SMT solvers to map quantified formulas into decidable ground theories. The mentioned works are proof-based, and crucially depend on the strength of state-of-the-art theorem provers. This is not the case in the work of Rybalchenko and Sofronie-Stokkermans presented in this special issue. Based upon constraint solving, their interpolation algorithm focuses on the combined theory of linear arithmetic and uninterpreted function symbols. Targeting safety properties, their empirical results are drawn from device driver and train control applications.

In contrast to most of the proceeding papers, which all involve a degree of empirical analysis, the contribution by Xia and Zhang included in this special issue is of a more theoretical nature. Specifically, they consider the problem of termination within the context of loop based programs where the loop conditions are polynomial and the loop body is linear with respect to updates. While termination of such programs over the reals and integers is undecidable, they identify and prove that termination over the reals is decidable if further restrictions are placed on the loop.

To conclude, a wide variety of techniques can be employed to reason about loops, and we hope that this special issue, together with the series of WING workshops, will play a part in bonding the community of researchers investigating this exciting topic at the border of formal methods, computational logic, and computer algebra.

May 2010

Martin Giese
Andrew Ireland
Laura Kovács

References

Bundy, A., Basin, D., Hutter, D., Ireland, A., Jun. 2005. Rippling: Meta-Level Guidance for Mathematical Reasoning. Vol. 56 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Christ, J., Hoenicke, J., 2010. Instantiation-Based Interpolation for Quantified Formulae. In: Proc. of SMT Workshop. To appear.

Craig, W., 1957. Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory. JSL 22 (3), 269–285.

Gedell, T., Hähnle, R., 2006a. Automating verification of loops by parallelization. In: Logic for Programming, Artificial Intelligence, and Reasoning. Vol. 4246 of Lecture Notes in Computer Science. Springer, pp. 332–346.

Gedell, T., Hähnle, R., 2006b. Verification by parallelization of parametric code. In: Aguzzoli, S., Ciabattoni, A., Gerla, B., Manara, C., Marra, V. (Eds.), Algebraic and Proof-theoretic Aspects of Non-classical Logics. Vol. 4460 of Lecture Notes in Computer Science. Springer, pp. 138–159.

Henzinger, T. A., Jhala, R., Majumdar, R., McMillan, K. L., 2004. Abstractions from Proofs. In: Proceedings of POPL. pp. 232–244.

Ireland, A., 2007. A cooperative approach for pointer programs. In: Giese, M., Jebelean, T. (Eds.), WING 2007, Workshop on Invariant Generation, Hagenberg, Austria. pp. 2–14, RISC Report 07-07.

Janota, M., 2007. Assertion-based loop invariant generation. In: Giese, M., Jebelean, T. (Eds.), WING 2007, Workshop on Invariant Generation, Hagenberg, Austria. pp. 15–26, RISC Report 07-07.

Jhala, R., McMillan, K. L., 2006. A Practical and Complete Approach to Predicate Refinement. In: Proceedings of TACAS. pp. 459–473.

Kiniry, J. R., Cok, D. R., 2005. ESC/Java2: Uniting ESC/Java and JML. In: Construction and Analysis of Safe, Secure and Interoperable Smart Devices: International Workshop, CASSIS 2004. Vol. 3362 of LNCS. Springer Verlag, pp. 108–128.

Kovács, L., 2008. Reasoning algebraically about p-solvable loops. In: Ramakrishnan, C. R., Rehof, J. (Eds.), TACAS. Vol. 4963 of Lecture Notes in Computer Science. Springer, pp. 249–264.

Kovacs, L., Voronkov, A., 2009a. Finding Loop Invariants for Programs over Arrays Using a Theorem Prover. In: Proceeding of FASE. pp. 470–485.

Kovacs, L., Voronkov, A., 2009b. Interpolation and Symbol Elimination. In: Proceeding of CADE. pp. 199–213.

Leino, K. R. M., Logozzo, F., 2005. Loop invariants on demand. In: Yi, K. (Ed.), Proc. 3rd Asian Symp. on Programming Languages and Systems, APLAS 2005, Tsukuba, Japan. Vol. 3780 of Lecture Notes in Computer Science. Springer, pp. 119–134.

Leino, K. R. M., Logozzo, F., 2007. Using widenings to infer loop invariants inside an smt solver, or: A theorem prover as abstract domain. In: Giese, M., Jebelean, T. (Eds.), WING 2007, Workshop on Invariant Generation, Hagenberg, Austria. pp. 70–84, RISC Report 07-07.

Letichevsky, A. A., Lvov, M. S., 1993. Discovery of invariant equalities in programs over data fields. Appl. Algebra Eng. Commun. Comput. 4, 269–286.

Lvov, M. S., 2007. Program polynomial invariants generation. In: Giese, M., Jebelean, T. (Eds.), WING 2007, Workshop on Invariant Generation, Hagenberg, Austria. pp. 85–99, RISC Report 07-07.

McMillan, K. L., 2008. Quantified Invariant Generation Using an Interpolating Saturation Prover. In: Proceedings of TACAS. pp. 413–427.

Müller-Olm, M., Seidl, H., 2004. Computing polynomial program invariants. Inf. Process. Lett. 91 (5), 233–244.

Olsson, O., Wallenburg, A., 2005. Customised induction rules for proving correctness of imperative programs. In: Aichernig, B., Beckert, B. (Eds.), Software Engineering and Formal Methods. 3rd IEEE International Conference, SEFM 2005, Koblenz, Germany, September 7–9, 2005, Proceedings. IEEE Computer Society Press, pp. 180–189.

Rodríguez-Carbonell, E., Kapur, D., 2004. Automatic generation of polynomial loop invariants: Algebraic foundations. In: ISSAC '04: Proceedings of the 2004 international symposium on Symbolic and algebraic computation. ACM, New York, NY, USA, pp. 266–273.

Rodríguez-Carbonell, E., Kapur, D., 2007. Automatic generation of polynomial invariants of bounded degree using abstract interpretation. Sci. Comput. Program. 64 (1), 54–75.

Sankaranarayanan, S., Sipma, H. B., Manna, Z., 2004. Non-linear loop invariant generation using Göbner bases. In: POPL '04: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages. ACM, New York, NY, USA, pp. 318–329.