Automatic Guidance for the Formal Verification of High Integrity Ada

Andrew Ireland
Department of Computing Science and Electrical Engineering
Heriot-Watt University
Edinburgh

Case for Support

1 Previous research track record

1.1 The applicant

Dr Andrew Ireland is a Lecturer in Computer Science. He has substantial research experience in the area of automated reasoning and in particular proof planning on which he has published widely [6, 4, 25, 27, 28, 31, 43, 44, 29]. Currently he is a co-investigator on two EPSRC funded grants: GR/L42889 (1997/00) "Parallelising compilation of Standard ML through prototype instrumentation and transformation" and GR/M45030 (rolling funding) "Computational Modelling of Mathematical Reasoning". The second of these grants is held at University of Edinburgh within the Mathematical Reasoning Group (MRG) where the technique known as proof planning was initially developed. His strong links with the MRG will greatly assist the dissemination of the results from the proposed programme of research. He has organized one national workshop on *automated reasoning* and two international workshops on the *automation of proof by mathematical induction*.

1.2 Research environment

The applicant works in the Department of Computing and Electrical Engineering at Heriot-Watt University. The Department received a 4B in the 1996 Research Assessment Exercise in the Computer Science unit of assessment. It is housed in a modern building, with excellent computer and infrastructural facilities.

The applicant is a founder member of the recently formed Dependable Systems Group, whose research focus is to improve the reliability and predictability of computer systems through the development and application of rigorous design, implementation and verification techniques. The current Group research areas include:

- Theorem proving, formal verification and synthesis.
- Design and implementation of parallel and distributed functional languages.
- Performance modelling and simulation of parallel and distributed systems.

Since its inception in 1997, the Group has grown to 1 Professor, 2 Senior Lecturers, 3 Lecturers, 1 Research Fellow, 4 Research Assistants and 5 PhD students. Group members currently hold 4 EPSRC research grants and have hosted a number of international workshops in the last year.

2 Description of proposed research and its context

2.1 Introduction

A key challenge identified by Foresight is the need for "readier access to formal methods for developers of safety critical systems". Automation will play a crucial role in meeting this challenge. The aim of this project is to investigate techniques for automating the formal verification of software for critical systems. In particular we will focus upon SPARK [1], a subset of the Ada programming language developed by Praxis Critical Systems[1]. SPARK has a proven track record in the production of software for critical systems within the finance, aerospace, rail and telecommunications markets. The value of formal verification is recognized by Praxis as being crucial to the production of critical systems. This is reflected within the SPARK toolkit that supports formal verification through the SPADE Proof Checker. The SPADE Proof Checker, however, is interactive and requires highly skilled users with development times typically measured in person-months. The need for greater efficiency within the formal verification process is recognized by Praxis. The CLAM Proof Planner [5] has a proven track record in increasing the level of automation provided by proof checkers. This project aims to increase the level of automation within the formal verification of software written in SPARK by investigating the coupling of the CLAM Proof Planner with the SPADE Proof Checker.

2.2 Scientific/Technological Relevance

The seminal work of Floyd [13] and Hoare [22] provided the theoretical foundation for many of the early software verification systems [36, 11, 17, 37]. Such systems were batch oriented, requiring the user to annotate their program code with assertions that specified the desired behaviour of the system. From the annotated program code a set of mathematical conjectures, known as *verification conditions*, were generated. Finally, a theorem prover was used to discharge the verification conditions. Although experience of these early systems led to the development of richer specification languages [16, 40], it is generally accepted that the batch approach to verification did not scale up. The onus on the user to supply intermediate lemmas and assertions, in particular loop invariants, is a key contributing factor to the lack of scalability. More generally, the batch approach gave rise to unmanageable verification conditions that provided little guidance when proof attempts failed as to whether the failure was due to bugs in the code or the specification.

A second generation of verification systems emerged based upon an incremental style of development, similar to that advocated[2] by Dijkstra [12] and Gries [20], *i.e.* the program and its proof of correctness are developed hand-in-hand. The incremental style ensures that verification conditions are kept relatively manageable, proof failures are easier to understand and bugs are identified early. The SPARK approach to programming advocates this style of development, so too does the PENELOPE[3] system [21]. Both PENELOPE and SPARK support Ada subsets. However, while PENELOPE focuses solely on formal verification, SPARK takes a more holistic approach, coupling formal verification with conventional program analysis techniques. SPARK was first defined in the late 1980's at Southampton University [8]. Its technical origins, however, date back to the 1970's at RSRE. Although technically a subset of Ada, SPARK is more than just a programming language. SPARK represents an approach to the production of high integrity software. This is reflected in the SPARK analysis tools which include flow analysis techniques as well as formal verification. The level of analysis selected is dictated by the criticality of the

[1] `http://www.praxis-cs.co.uk`

[2] Although both Dijkstra and Gries were motivated by programming as a more human oriented activity.

[3] Odyssey Research Associates.

application. Formal verification is applicable when dealing with the development of critical systems.

The second generation verification systems do not address the problem of intermediate lemmas and loop invariants mentioned above. Although heuristic based techniques for automating the discovery of loop invariants have been investigated extensively [45, 33, 46, 15, 34, 7, 10, 39, 38], they have had little impact on the mainstream verification systems. The need for cohesion between the heuristic guidance and theorem proving components was identified as a contributing factor early on [45], yet was never investigated.

The novelty of our proposal is that we directly address this issue of cohesion. We aim to do this through proof planning, a technique for guiding the search for proofs that provides a single framework for representing high-level heuristics as well as theorem proving knowledge. The proof planning technique that is implemented in CLAM was first developed within the Department of Artificial Intelligence[4] at Edinburgh University in the late 1980's. Proof planning builds upon the LCF style of theorem proving [19], where primitive proof steps are packaged-up into programs known as *tactics*. Starting with a set of general purpose tactics, plan formation techniques are used to construct a customized tactic for a given conjecture. The search for a customized tactic is constrained by a set of *methods*, each of which specifies the applicability of a general purpose tactic. Collectively a set of methods is known as a *proof plan*. CLAM was initially used to drive the OYSTER interactive proof checker [23], a Prolog re-implementation of the Cornell NUPRL proof development environment [9]. Proof planning has been investigated extensively within the context of proof by mathematical induction [4]. What distinguishes proof planning from other approaches to theorem proving is the flexibility it offers by separating the issues of search (method level) and soundness (tactic level). This means that the planning of a proof need not follow a strict backward or forward style of construction. This flexibility gave rise to *proof critics* [25, 27], an extension that supports the automatic analysis and patching of failed proof planning attempts. The most striking proof planning successes have been achieved where proofs require the discovery of auxiliary inductive lemmas or generalizations [26, 27, 28]. Such proof discovery capabilities outperform the conventional inductive theorem provers [3, 35, 2, 42, 41, 14, 32, 24]. Loop invariants are by their nature inductive so the proof plans for induction easily transferred to the problem of reasoning about imperative programs [31, 43, 44, 30]. The potential for using CLAM to guide proof checkers, other than OYSTER, has already been demonstrated[5] by the successful coupling of CLAM with the Cambridge HOL interactive theorem prover [18]. The difference between our proposal and the CLAM-HOL link is that we are aiming to exploit and extend the proof patching capabilities of CLAM. In addition, a CLAM-SPADE link will allow the proof planning techniques to be tested within the context of safety critical applications.

The time is now ripe to test the proof discovery capabilities of proof planning on "industrial strength" problems. SPARK is the ideal vehicle given its commercial success within the production of critical systems. We believe that proof planning techniques can make a significant impact on the SPARK approach to producing high integrity software. We also believe that working on "industrial strength" problems will be beneficial to our basic research agenda. In the longer term we hope to build upon the investment made within this project. In particular, the insights gained from the work described within this "project definition" proposal will enable us to further improve the efficiency of the proof management component of the SPARK tools.

[4]The Department of Artificial Intelligence has been absorbed within the Division of Informatics.

[5]A collaborative research project between the Division of Informatics in Edinburgh and the Computer Laboratory in Cambridge funded by EPSRC grants GR/L/14381 and GR/L/03071.

2.3 Programme and methodology

2.3.1 Aims and Objectives

We wish to investigate the coupling of CLAM to the SPADE Proof Checker. Our research hypothesis is:

> *The coupling of* CLAM *with the* SPADE *Proof Checker will significantly reduce the amount and sophistication of user interaction that is required in order to complete the formal verification of software written in* SPARK.

In particular, we will provide automatic guidance in the following key areas:

Assertion discovery: missing assertions, in particular loop invariants, are a major bottle-neck within the formal verification of software. Our aim is to build upon our previous success in automating the discovery of inductive generalizations/ invariants. Our contribution will be to reduce the number of intermediate assertions that a user of the SPADE Proof Checker will typically be required to provide.

Lemma discovery: anticipating all the lemmas that are required before starting the formal verification is unrealistic. Our aim is to build upon our previous success in automating the discovery of inductive lemmas on-the-fly during a proof attempt. Our contribution will be to reduce the number of lemmas (rules) that a user of the SPADE Proof Checker will typically be required to provide.

Praxis have demonstrated that the SPARK approach reduces the time-to-market for high integrity software. The work being proposed would further reduce time-to-market by increasing the overall productivity of a software engineer using SPARK. Our research hypothesis will be tested through empirical analysis. While a corpus of standard "text book" examples is envisaged for the initial development phase, "industrial strength" problems provided by Praxis will be used during the evaluation phase. We believe that our approach will work because of our previous success in the area of proof discovery [26, 27, 28, 31, 43, 44, 30]. We anticipate, however, that the investigation of the "industrial strength" problems will lead to natural generalizations and extensions to the current set of proof plans. Finally, it is worth noting that both CLAM and the SPADE Proof Checker are implemented in Prolog, this should ease the low-level implementation details when coupling the systems.

2.3.2 The major tasks and timetable

The research programme has been broken down into five *workpackages*. A description of the aims of each workpackage is given below together with a breakdown of the tasks and associated deliverables. A diagrammatic project plan that details the workpackages and milestones is provided in the appendix.

WP1: Construction of Experimental System [Months 1 to 8]

The aim is to develop a prototype interface between CLAM and the SPADE Proof Checker.
Tasks:

T1: Achieve compatibility between CLAM and the SPADE Proof Checker in terms of the representation of definitions, rules, lemmas, conjectures *etc.* [2 person months]

T2: Modify the CLAM library mechanism in order to support the needs of the SPADE Proof Checker. [2 person months]

T3: Develop a "tactic" like mechanism that will enable CLAM to guide the application of atomic proof steps within the SPADE Proof Checker. [2 person months]

T4: Build a corpus of test conjectures to be used during the initial testing phase. [2 person months]

Deliverables:

D1: Experimental system (prototype 1).

D2: Corpus of conjectures.

[Total: 8 person months]

WP2: Adaption of the Key Proof Methods & Critics [Months 9 to 14]

The aim is to transfer and adapt as necessary the existing proof plans that are key to this application, *i.e.* proof methods for mathematical induction, loop invariant verification and some simple proof critics. This will enable initial testing to be carried out.
Tasks:

T5: Adapt proof methods associated with mathematical induction and loop invariant verification. [2 person months]

T6: Construct the corresponding "tactics" for the SPADE Proof Checker. [2 person months]

T7: Adapt a couple of related proof critics. [2 person months]

Deliverables:

D3: An initial set of proof methods and critics together with SPADE Proof Checker level "tactics".

[Total: 6 person months]

WP3: Initial Testing & Re-design [Months 15 to 18]

The aim of the initial testing phase is to gain feedback on the first prototype before starting the more in-depth investigations. It is anticipated that this feedback will lead to modifications of the prototype.
Tasks:

T8: Conduct initial testing using the examples corpus (see D2). [2 person months]

T9: Evaluate the results of the testing and implement modifications as required to the system. [2 person months]

Deliverables:

D4: Experimental system (prototype 2).

D5: A research report describing the system and the results of the initial testing.

[Total: 4 person months]

WP4: Extending the Proof Patching Capabilities [Months 19 to 24]

The aim is to ramp up the proof patching capabilities of the experimental system which will include significant extensions to the existing loop invariant discovery techniques. This will involve an investigation into the kind of proof obligations that arise in the large proof projects that Praxis have undertaken.
Tasks:

T10: Initial investigation of SPARK applications. [2 person months]

T11: Extend and develop new proof methods and critics based upon the needs of SPARK applications. [4 person months]

Deliverables:

D6: Extensions to existing proof critics as well as new proof methods and critics driven by the needs of the SPARK applications.

D7: A research report describing the initial investigation of the SPARK applications.

D8: A research report describing the new and improved proof methods and critics.

[Total: 6 person months]

WP5: Industrial Strength Evaluation [Months 25 to 36]

The aim is to consolidate and evaluate the system on "industrial strength" problems provided by Praxis. It is anticipated that this process will lead to relatively minor modifications to the system and proof plans. At the end of this phase we hope to be able to demonstrate benefits of proof planning through significant reductions in the level of user interaction required during proof efforts.
Tasks:

T12: Evaluation based upon "industrial strength" problems, including minor modifications to the proof plans as required.

Deliverables:

D9: Experimental system (prototype 3).

D10: A research report that documents the "industrial strength" evaluation phase.

D11: Final report.

[Total: 12 person months]

2.4 Relevance to beneficiaries

The immediate beneficiaries of the work will be SPARK users and researchers working in the area of proof planning. The SPARK users will have a tool that significantly reduces the amount and sophistication of user interaction required in formally verifying critical systems implemented in SPARK.

More widely, we hope that the results will be of material use to other research and development teams working in the area of critical systems development. In addition, we would expect that aspects of the theorem proving work will be of interest to the wider automated deduction and formal methods communities, in particular the case studies and the new proof plans that are developed.

2.5 Dissemination and exploitation

We will seek to publish our results in high quality journals and at the relevant major international conferences and workshops. We anticipate at least two journal publications to come from this project. The system itself and associated deliverables will be made available via the web.

2.6 Management & Resources

Andrew Ireland will have overall responsibility for project management and will oversee the the development of the system. He will spend 10% of his time working on the project.

As well as interaction with Praxis, the Heriot-Watt group will maintain their strong links with the Mathematical Reasoning Group within the Division of Informatics at the University of Edinburgh. These links will provide crucial feedback on the project's progress.

Staff: We request a Research Assistant, to be appointed on the RA 1A scale, for 36 months. The RA will have responsibility for constructing and developing the overall system, as well as conducting the "industrial strength" case studies. A background in theorem proving and programming logics will be essential. While the majority of the system development work will be carried out by the RA, Andrew Ireland will contribute in terms of the overall system architecture and the investigation into new proof patching techniques. Funding is also requested to partially support a Computing Officer.

Equipment: We request funding to purchase 2 high power Pentium PCs. These will provide the necessary performance for developing and evaluating the the prototype systems. A small sum to cover consumables during the project will also be required. Although our Department will provide general networking support (see below), we will require funds in order to interface with the network from within our research lab.

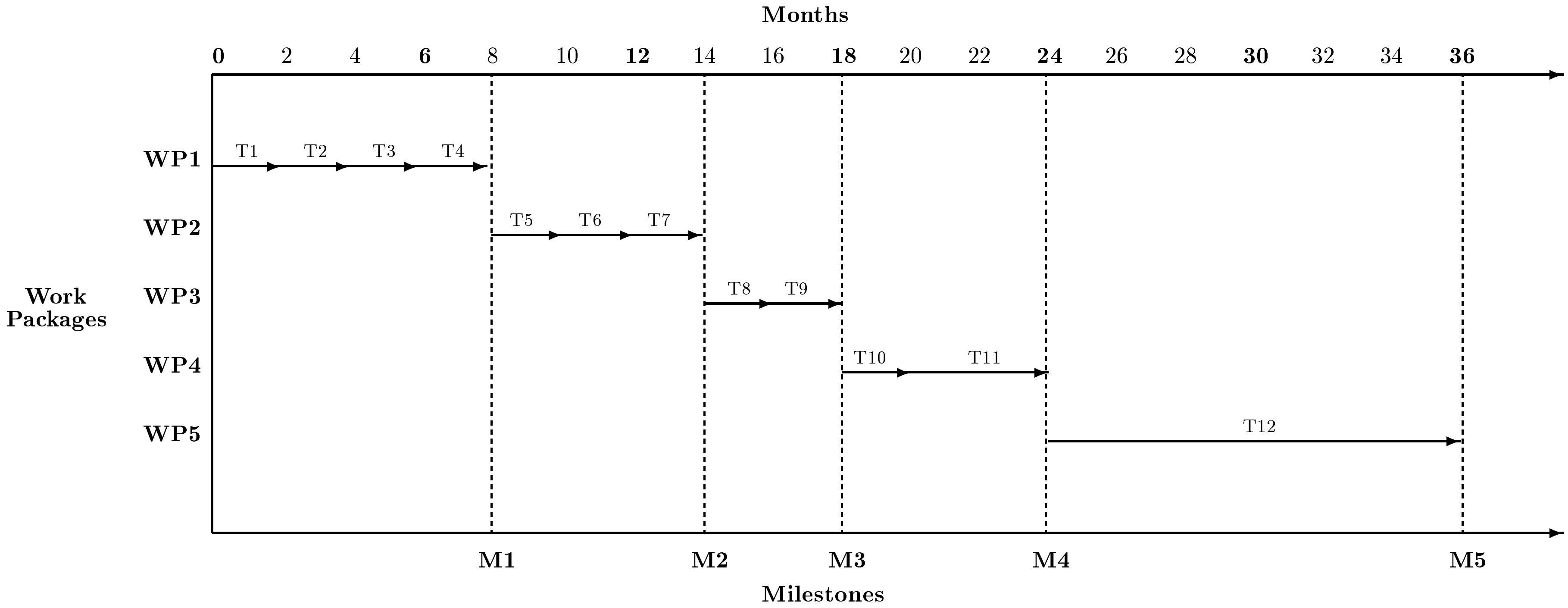
Travel: Interaction with Praxis will play a critical role within the project. The sensitive nature of certain aspects of the case study material will also require the RA to spend time working at Praxis. In order to facilitate this interaction we request funds to cover 12 weeks of visits to Praxis over the life-time of the project. We plan for a 2 week visit during workpackage 1, a 4 week visit during workpackage 4, and a 6 week visit during workpackage 5.

It is difficult to state specifically which conferences we will attend given conference deadlines and our publication timing. However, we have identified the following conferences from the Automated Deduction and Formal Verification areas as being appropriate: Formal Methods Europe (FME), Conference on Automated Deduction (CADE), Computer Aided Verification (CAV), IEEE Conference on Automated Software Engineering (ASE). In all we request funds for presenting papers at four conferences.

Industrial resources: Our partners, Praxis Critical Systems, will provide indirect contributions to the project (see letter of support). These will take the form of the SPARK toolkit together with access to a problem set based upon their experience of developing software for critical systems. The value of their indirect contribution will be in excess of £40,000.

Institutional resources: Our Department will provide laboratory and office space for the RA. It will also provide the computer and networking infrastructure support, telephone, photocopying, and the other usual facilities.

Appendix: Diagrammatic Project Plan



M1: First prototype system (D1) and corpus of conjectures (D2).

M2: Initial set of proof methods and critics together with SPARK proof checker level “tactics” (D3).

M3: Second prototype system (D4), research report describing the system and the initial testing (D5).

M4: Extensions to techniques driven by the needs of the SPARK applications (D6), research report describing the initial investigation of the SPARK case study material (D7), research report describing the new and improved proof methods and critics (D8).

M5: Third prototype system (D9), research report documenting the “industrial strength” evaluation phase (D10), final report (D11).

References

[1] J. Barnes. *High Integrity Ada: The Spark Approach.* Addison-Wesley, 1997.

[2] A. Bouhoula and M. Rusinowitch. Automatic case analysis in proof by induction. In *Proceedings of the 13th IJCAI.* International Joint Conference on Artificial Intelligence, 1993.

[3] R. S. Boyer and J S. Moore. *A Computational Logic Handbook.* Academic Press, 1988. Perspectives in Computing, Vol 23.

[4] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62:185–253, 1993. Also available from Edinburgh as DAI Research Paper No. 567.

[5] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 507.

[6] A. Bundy, F. van Harmelen, A. Smaill, and A. Ireland. Extensions to the rippling-out tactic for guiding inductive proofs. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 132–146. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 459.

[7] M. Caplain. Finding invariant assertions for proving programs. In *Proceedings of the International Conference on Reliable Software*, Los Angeles, CA, 1975.

[8] B.A. Carre and T.J. Jennings. *SPARK – The SPADE Ada Kernel.* Dept. of Electronics and Computer Science, University of Southampton, 1988.

[9] R. L. Constable, S. F. Allen, H. M. Bromley, et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, 1986.

[10] N. Dershowitz and Z. Manna. Inference rules for program annotation. *IEEE Trans. on Software Engineering*, SE-7(2):207–222, 1981.

[11] L.P. Deutsch. *An Interative Program Verifier.* PhD thesis, University of California, Berkeley, 1973.

[12] E. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[13] R. W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science, Proceedings of Symposia in Applied Mathematics 19*, pages 19–32. American Mathematical Society, 1967.

[14] S. J. Garland and John. V. Guttag. *A Guide to LP, The Larch Prover*, November 1991.

[15] S.M. German and B. Wegbreit. A synthesizer of inductive assertions. *IEEE Trans. on Software Engineering*, SE-1(1):68–75, 1975.

[16] D. I. Good. Mechanical proofs about computer programs. In C. A.R. Hoare and J. C. Shepherdson, editors, *Mathematical Logic and Programming Languages*, chapter 3, pages 55–75. Prentice-Hall, 1985.

[17] D.I. Good, R.L. London, and W.W. Bledsoe. An interative program verification system. *IEEE Trans. on Software Engineering*, SE-1(1):59–67, 1975.

[18] M. J. Gordon. HOL: A proof generating system for higher-order logic. In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*. Kluwer, 1988.

[19] M. J. Gordon, A. J. Milner, and C. P. Wadsworth. *Edinburgh LCF - A mechanised logic of computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[20] David Gries. *The Science of Programming*. Springer-Verlag, New York, 1981.

[21] D. Guaspari, C. Marceau, and W. Polak. Formal verification of Ada programs. *IEEE Trans. on Software Engineering*, 16(9):1058–1075, September 1990.

[22] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–583, 1969.

[23] C. Horn and A. Smaill. Theorem proving and program synthesis with Oyster. In *Proceedings of the IMA Unified Computation Laboratory*, Stirling, 1990.

[24] D. Hutter and C. Sengler. INKA: The Next Generation. In M.A. McRobbie and J.K. Slaney, editors, *Proceedings of CADE-13*. Springer Verlag, 1996. (LNAI vol. 1104).

[25] A. Ireland. The Use of Planning Critics in Mechanizing Inductive Proofs. In A. Voronkov, editor, *International Conference on Logic Programming and Automated Reasoning – LPAR 92, St. Petersburg*, Lecture Notes in Artificial Intelligence No. 624, pages 178–189. Springer-Verlag, 1992. Also available from Edinburgh as DAI Research Paper 592.

[26] A. Ireland and A. Bundy. Extensions to a Generalization Critic for Inductive Proof. In M. A. McRobbie and J. K. Slaney, editors, *13th International Conference on Automated Deduction*, pages 47–61. Springer-Verlag, 1996. Springer Lecture Notes in Artificial Intelligence No. 1104. Also available from Edinburgh as DAI Research Paper 786.

[27] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1–2):79–111, 1996. Also available as DAI Research Paper No 716, Dept. of Artificial Intelligence, Edinburgh.

[28] A. Ireland and A. Bundy. Automatic Verification of Functions with Accumulating Parameters. *Journal of Functional Programming: Special Issue on Theorem Proving & Functional Programming*, 9(2):225–245, March 1999. A longer version is available from Dept. of Computing and Electrical Engineering, Heriot-Watt University, Research Memo RM/97/11.

[29] A. Ireland, M. Jackson, and G. Reid. Interactive Proof Critics. *Formal Aspects of Computing: The International Journal of Formal Methods*, 11(3):302–325, 1999. A longer version is available from Dept. of Computing and Electrical Engineering, Heriot-Watt University, Research Memo RM/98/15.

[30] A. Ireland and J. Spark. Proof planning for strategy development. Research Memo RM/00/3, Dept. of Computing and Electrical Engineering, Heriot-Watt University, 2000. To appear in a Special Issue of the Annals of Mathematics and Artificial Intelligence on Strategies in Automated Deduction.

[31] A. Ireland and J. Stark. On the Automatic Discovery of Loop Invariants. In *Proceedings of the Fourth NASA Langley Formal Methods Workshop – NASA Conference Publication 3356*, 1997. Also available from Dept. of Computing and Electrical Engineering, Heriot-Watt University, Research Memo RM/97/1.

[32] D. Kapur and H Zhang. An Overview of Rewrite Rule Laboratory (RRL). *Journal of Computer and Mathematics with Applications*, 29(2):91–114, 1995.

[33] S.M. Katz and Z. Manna. A heuristic approach to program verification. In *Proceedings of IJCAI-73*. International Joint Conference on Artificial Intelligence, 1973.

[34] S.M. Katz and Z. Manna. Logical analysis of programs. *Communications of the ACM*, 19(4):188–206, 1976.

[35] M. Kaufmann and J. Moore. An Industrial Strength Theorem Prover for a Logic Based on Common Lisp. *IEEE Transactions on Software Engineering*, 23(4):203–213, 1997.

[36] J. King. *A Program Verifier*. PhD thesis, Carnegie-Mellon University, 1969.

[37] D. C. Luckham, S.M. German, F.W. v.Henke, R.A. Karp, P.W. Milne, D.C. Oppen, W. Polak, and W.L. Scherlis. Stanford pascal verifier user manual. Research Report CS-TR-79-731, Stanford University, Department of Computer Science, 1979.

[38] A. Mili, J. Desharhais, and J. Gagne. Strongest invariant functions: Their use in the systematic analysis of while statements. *Acta Informatica*, 22:47–66, 1985.

[39] A. Mili, J. Desharhais, and F. Mili. *Computer Program Construction*. Oxford University Press, 1994.

[40] D.R. Musser. Abstract data type specification in the AFFIRM system. *IEEE Trans. on Software Engineering*, SE-6(1):24–32, January 1980.

[41] ORA. Introduction to EVES: Eercises and Notes. In *Odyssey Research Associates, Ottawa Ontario, Canada*, 1996.

[42] S. Owre, J. M. Rushby, and N. Shankar. PVS : An integrated approach to specification and verification. Tech report, SRI International, 1992.

[43] J. Stark and A. Ireland. Invariant discovery via failed proof attempts. In P. Flener, editor, *Logic-Based Program Synthesis and Transformation*, number 1559 in LNCS, pages 271–288. Springer-Verlag, 1998. An earlier version is available from the Dept. of Computing and Electrical Engineering, Heriot-Watt University, Research Memo RM/98/2.

[44] J. Stark and A. Ireland. Towards automatic imperative program synthesis through proof planning. In *The* 14^{th} *IEEE International Conference on Automated Software Engineering*, pages 44–51. IEEE Computer Society, 1999.

[45] B. Wegbreit. Heuristic methods for mechanically deriving inductive assertions. In *Proceedings of IJCAI-73*. International Joint Conference on Artificial Intelligence, 1973.

[46] B. Wegbreit. The synthesis of loop predicates. *Comm. ACM*, 17(2):102–122, 1974.