# Cooperative Reasoning for Automatic Software Verification

Andrew Ireland
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh

Case for Support

## 1 Previous Research

Dr Andrew Ireland is a Senior Lecturer in Computer Science within the School of Mathematical and Computer Sciences at Heriot-Watt University. He has substantial research experience in the area of *automated reasoning* and *automated software engineering*. In particular he has played an influential role in the development of proof planning [8, 10, 23, 27, 28, 31, 32, 33, 34, 49, 50]. He has also been involved most recently in more applied research. As principle investigator of the NuSPADE project (GR/R24081) he successfully applied the proof planning ideas to the SPARK Approach [1] to developing safe and secure software. This involved a strong collaborative element with Praxis High Integrity Systems Ltd (Bath), which led to a follow-on Knowledge Transfer project (GR/T11289). The NuSPADE project gave rise to an integrated approach to software verification [16, 17, 24, 29, 30], which was highly rated by the EPSRC reviewers. The integrated theme of the NuSPADE project is also reflected in another application-oriented project called PARTES, which combined model checking and performance modelling. PARTES, was funded by an Industrial Case Award (GR/PO1786) in collaboration with QinetiQ's Systems Assurance Group (Malvern) [20]. Another integrated project in which Dr Ireland was involved in (as co-investigator) focused on the "Parallelising compilation of Standard ML through prototype instrumentation and transformation" (GR/L42889). Here he provided automated reasoning expertise in terms of transformation rule synthesis and verification [12, 13].

During his 12 years at Heriot-Watt, Dr Ireland has maintained a close collaborative relationship with Prof. Bundy's Mathematical Reasoning Group (MRG) at the University of Edinburgh. He is an Honorary Research Fellow of Edinburgh University and is currently a co-investigator on Prof. Bundy's Platform grant, *i.e.* "The Integration and Interaction of Multiple Mathematical Reasoning Processes" EP/E005713 (and GR/S01771). Previously he was a co-investigator on Prof. Bundy's rolling funding grants, *i.e.* GR/M45030, GR/L11724 and GR/J80702.

It should be noted that Platform grant GR/S01771 has provided considerable support in developing the current proposal, and the strong links with the MRG will greatly assist with the development and dissemination of the proposed programme of research.

Dr Ireland has organized a number of national and international workshops in automated reasoning. He is a member of programme and organizing committees for a series of national and international workshops and conferences. He has been a member of the Programme Committees/Expert Reviewer Panels/Conference Committees for a series of IEEE/ACM International Conferences on Automated Software Engineering (ASE-01, ASE-02, ASE-03, ASE-04, ASE-05, ASE-06, ASE-07), as well as the International Conference on Automated Deduction (CADE-18, CADE-19). He was appointed to the Steering Committee for ASE in 2006 and will be Programme Co-Chair for ASE-08 in L'Aquila (Italy). He reviews papers for major journals, *i.e.* JAR, AMAI and JLC. He has been a member of the EPSRC Peer Review College since 2003.

In terms of research environment, Computer Science at Heriot-Watt received a 4A in the 2001 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 founding member of the Dependable Systems Group (DSG), 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 DSG research areas include: Theorem proving, formal verification and synthesis; Design of parallel and distributed functional languages; Performance modelling and simulation of parallel and distributed systems. Since its inception in 1997, the DSG has grown to 2 Professors, 3 Senior Lecturers, 2 Lecturers and 5 Research Assistants and 19 PhD students. Group members currently hold 4 research grants.

## Dr Ewen Maclean (Research Associate)

Dr Maclean gained his PhD from the University of Edinburgh in 2004 for work on automating formal proofs in non-standard analysis using a proof planning approach. For most of 2003 and 2004 he was an RA with the Mathematical Reasoning Group at the University of Edinburgh. He supported the group's research activities with the lambda-Clam proof planner and took over development of the IsaPlanner proof-planning system built on top of the Isabelle/HOL theorem prover.

Recently he was a consultant on a project to determine the market feasibility of program analysis tools based on research at Edinburgh and Heriot-Watt universities. His role was to develop and implement a preliminary prototype system which generated verification conditions and proved these automatically using IsaPlanner. Program properties considered by the system included properties associated with heap data and with resource consumption. Much of the work was concerned with automatically generating loop invariants. The results obtained were very promising and he is presently continuing this work with the company MicroArt in Barcelona, analysing their Java code for possible applications of the research.

## Miss Ianthe Hind (Research Student)

Miss Hind is a first year Computer Science PhD student within the School of Mathematical and Computer Sciences at Heriot-Watt University. She is currently a visiting researcher within the Mathematical Reasoning Group at Edinburgh University. Her research interests lie within *automated reasoning* and the applications of logic. She has been tutoring throughout her academic career, most recently Intelligent Agents, Formal Specification and Compiler Theory at Heriot-Watt University.

Miss Hind completed her BSc (Honours) in Computer Science (2.1) at Edinburgh University in 2006. Her Dissertation was entitled Extending the Capabilities of Anastasia. Anastasia is a structural editor for programming languages that uses proofs as programs to exploit the duality between functional programs and proofs in constructive logic. Her work involved writing a grammar for the Haskell language (including list comprehensions and lambda abstractions), investigating the integration of termination checking and the completion a successful evaluation of Anastasia's usefulness compared to that of a free form text editor.

Prior to attending Edinburgh University, Miss Hind gained valuable experience in industry before realising that her true interests lay in research. She completed her BSc in Applied Mathematics and Computer Science (2.1) at the University of Cape Town in 2001 before moving to the United Kingdom.

## 2 Proposed Research

### 2.1 Background

The proliferation of software across all aspects of modern life means that software failures can have significant economic, as well as social impact. The goal of being able to develop software that can be formally verified as correct with respect to its intended behaviour is therefore highly desirable. The foundations of such formal verification have a long and distinguished history, dating back over fifty years [18, 19, 22]. What has remained more elusive are *scalable verification tools* that can deal with the complexities of software systems.

However, times are changing, as reflected by a current renaissance within the formal software verification community. An IFIP working group has been set up with the aim of developing a *Grand Challenge* for *Verified Software* [35]. Dr Ireland has participated in a number of events which have helped shape the Grand Challenge, and the current proposal is closely aligned with its goals. There have also been some notable industrial success stories. For instance, Microsoft's Static Device Verifier (SDV) [40] and the SPARK Approach to developing high integrity software [1]. Both of these successes address the scalability issue by focusing on generic properties and tool integrations that support a high degree of automation. In the case of SDV, the focus is on *deadlock freedom* at the level of resource ownership for device driver software. Abstraction and model checking are used to identify potential defects, which are then refined via theorem proving to eliminate false alarms. The SPARK Approach has been used extensively in the development of safety [36] and security [21] critical applications. It provides a loose coupling of analysis techniques from data and information flow analysis through to formal verification via theorem proving. One of its key selling points is its support for automating so called *exception freedom* proofs, *i.e.* proving that a system is free from common run-time errors such as buffer overflows.

The targeting of generic properties, such as deadlock and exception freedom, has proved both highly effective and extremely valuable to industry. However, to increase the value of software correctness guarantees will ultimately call for a more comprehensive level of specification, *i.e.* correctness specifications. In the case of bespoke applications, this might take the form of correctness specifications developed in conjunction with the customer requirements. Alternatively, the verification of software libraries and components against agreed correctness standards could prove highly valuable across a wide range of sectors. The focus of this proposal is on developing techniques that support the **automatic verification of correctness specifications**.

Verifying code against more comprehensive specifications will call for significant advances in terms of scalable tools. We believe that these advances will require frameworks which provide, i) general support for *modular reasoning* as well as, ii) a flexible basis in which novel *tool integrations* can be investigated. Our proposal builds upon *separation logic*, where modular reasoning is a key feature. At the level of tool integration, we propose the use of *proof planning*, an automated theorem proving technique which has a track-record in successfully combining reasoning tools:

**Separation logic** was developed as an extension to Hoare logic [22], with the aim of simplifying pointer program verification proofs [43, 46]. Pointers are a powerful and widely used programming mechanism, but developing and maintaining correct pointer programs is notoriously hard. A key feature of separation logic is that it focuses the reasoning effort on only those parts of the heap that are relevant to a program, so called *local reasoning.* Because it deals smoothly with pointers, including "dirty" features such as memory disposal and address arithmetic, separation logic holds the promise of allowing verification technology to be applied to a much wider range of **real-world software** than has been possible up to now.

In terms of tool development, the main focus has been on shape analysis. Such analysis can be used to verify properties about the structure (*shape*) of data structures within the heap. For example, given a sorting program that operates on singly linked list, then shape analysis techniques can be used to verify that for an arbitrary singly linked list the program will always output a singly linked list. Note that shape analysis ignores the *content* of data structures. Smallfoot [3] is an experimental tool that supports the automatic verification of shape properties specified in separation logic. Smallfoot uses a form of symbolic execution [4], where loop invariants are required. Related tools are SLAyer [2], Space Invader [14] and Smallfoot-RG [11], all of which build directly upon the foundations of Smallfoot. Within SLAyer, higher-order generic predicates are used to express families of complex composite data structures. A restricted form of predicate synthesis is used to instantiate the generic predicates during shape analysis. Space Invader, unlike Smallfoot supports loop invariant discovery via fixed point analysis. Abstraction is used to overcome divergence in the search for a fixed point. Smallfoot-RG also includes Space Invader's invariant discovery strategy. Closely related to Space Invader is an algorithm developed at CMU for inferring loop invariants within the context of separation logic [38]; again fixed point analysis is the underlying mechanism. Another interesting tool is reported in [41], which combines shape and size analysis. Inductive predicates play a significant role in specifying pointer programs within separation logic. A limitation of the program analysis tools mentioned above is that they are hard-wired with pre-defined inductive predicates, *e.g.* singly linked lists, binary trees etc. To make these tools extensible would require the ability to add new user-defined inductive predicates on-the-fly. To achieve this would require the ability to automate proof by mathematical induction. We will return to this point later.

Less work has been undertaken in the area of theorem proving for separation logic. In [45] a partial formalization within PVS [44] is presented which supports the verification of recursive procedures. A complementary formalization, which includes simple while loops, but not recursive procedures, is presented in [51]. In [37] separation logic is added to Isabelle [42] using the Schirmer's verification environment for sequential imperative programs [47]. This integration was used to reason about pointer programs written in C. Finally, in [39] the Coq proof environment has been extended with separation logic in order to verify the C source code of the Topsy heap manager. All these applications of theorem proving to separation logic have involved significant user interaction, *e.g.* user specified induction rules. In contrast, our proposal focuses on **verification automation** in which user interaction is eliminated as far as possible.

**Proof planning** is a technique for automating the search for proofs through the use of high-level proof outlines, known as *proof plans* [5]. The current state-of-the-art proof planner is called IsaPlanner [15], which is Isabelle based. Proof planning has been used extensively for proof by mathematical induction [9]. Mathematical induction is essential for the synthesis and verification of the inductively defined predicates that arise within separation logic specifications. Proof planning therefore offers significant benefits for reasoning about separation logic specifications. In addition, the kinds of data structures that arise naturally when reasoning about pointer programs, *i.e.* a queue implemented as a "circular" linked list, will provide challenging examples which will advance the existing proof plans. A distinctive feature of proof planning is *middle-out reasoning* [7], a technique where meta-variables, typically higher-order, are used to delay choice during the search for a proof. Middle-out reasoning has been used to greatest effect within the context of *proof critics* [23], a technique that supports the automatic analysis and patching of failed proof attempts. Such *proof patching* has been applied successfully to the problems of inductive conjecture generalization and lemma discovery [27, 28], as well as loop invariant discovery [33]. This work is currently being integrated and extended within IsaPlanner. The tool integration capabilities of proof planning have been demonstrated through the Clam-HOL [48] and NuSPADE projects[1] [29]. The NuSPADE project targeted the SPARK Approach [1], and integrated proof planning with light-weight program analysis in order to increase proof automation for loop-based code. The resulting integration was applied to industrial strength problems and successfully increased the level of proof automation for exception freedom proofs [29].

## 2.2 Programme and Methodology

### 2.2.1 Research Aims

There are two long-term aspects to our work. Firstly, we aim to deepen our understanding of the complementary nature of program analysis and deductive reasoning. Secondly, we aim to use this understanding to inform the development of integrated approaches to the problem of automating software verification. In particular, we are interested in approaches where there is *real cooperation* between complementary techniques. That is, where individual techniques combine their strengths, but crucially compensate for each other's weaknesses through the communication of partial results and failures. More general evidence as to the merits of such cooperation can be found in [6]. For us the pay-off of achieving this level of cooperation will be measured in terms of automation, *i.e.* we believe that this form of cooperation will deliver verification automation where skilled human interaction is currently essential. Our starting point is the proof planning paradigm, and our research hypothesis is:

*Proof planning provides a framework within which real cooperation between program analysis and deductive reasoning can be achieved.*

As a first step towards validating this hypothesis we have chosen to focus upon a specific instance, *i.e.* an integration of the Smallfoot program analyzer and the IsaPlanner proof planner. We believe that this will take 3 years. After which point we will review progress with respect to the more general hypothesis. It should be noted that the generic nature of IsaPlanner/Isabelle means that one of the outcomes of the programme of work will be a generic framework which will directly support our longer-term aims. Figure 1 provides some details as to how cooperation between Smallfoot and IsaPlanner could be achieved.

[1]NuSPADE project: `http://www.macs.hw.ac.uk/nuspade`

We see two areas where proof planning could be combined cooperatively with shape analysis. When reasoning about recursively defined procedures, the discovery of an appropriate frame axiom plays a pivotal role. Likewise when reasoning about iterative code, the discovery of an appropriate loop invariant plays a pivotal role. The Frame and While-loop rules shown below provide the logical foundation for reasoning about frame axioms and loop invariants respectively:

Frame rule:

$$\frac{\{P\}\ C\ \{Q\}}{\{R * P\}\ C\ \{R * Q\}}$$

While-loop rule:

$$\frac{\{P \rightarrow R\}\quad \{R \wedge S\}\ C\ \{R\}\quad \{\neg S \wedge R \rightarrow Q\}}{\{P\}\ \textbf{while}\ S\ \textbf{do}\ \{R\}\ C\ \textbf{od}\ \{Q\}}$$

The frame rule imposes a side condition, *i.e.* no variable occurring free in $R$ is modified by $C$. Note also that in both rules, $R$ will typically not form part of a program's overall correctness specification. That is, $R$ represents an auxiliary part of the specification, typically supplied via user interaction. In terms of proof search, $R$ represents an infinite choice point, and therefore a major challenge to achieving verification automation. Our proposal directly addresses this challenge. We are focusing on correctness specifications, so $R$ describes both the **shape** and **content** of heap data structures. Note that Smallfoot, and its related program analysis tools, support the automatic discovery of shape properties, but they do not address the issue of content. We believe that proof planning, via middle-out reasoning and proof patching, will enable shape properties to be automatically extended to include properties about the content of data structures within the heap. Smallfoot provides strength in terms of automating the discovery of shape properties while the strength of IsaPlanner lies in its ability to automate the discovery of properties about the content of data structures. In addition, the inductive theorem proving capabilities of IsaPlanner will enable us to compensate for the limitations of current program analysis tools, *i.e.* as mentioned above, extensibility requires the ability to automatically reason about inductively defined predicates. More details on how we believe real cooperation can be achieved are provided in [25, 26].

Figure 1: Automating the Discovery of Frame Axioms & Loop Invariants

### 2.2.2 Research Objectives and Programme of Work

To achieve our integration, we plan to build upon Schirmer's generic verification environment mentioned in §2.1. As it is currently funded by the Verisoft[2] project, we will refer to it as the VerisoftVE. A key reason for selecting VerisoftVE is the fact that it is Isabelle based, which makes it an ideal choice given that IsaPlanner is also Isabelle based. Our principal objectives for this three year research project are:

- Develop an extensible program analyzer that automates the discovery of shape properties within separation logic.
- Develop proof plans that work cooperatively with the program analyzer in order to automatically verify the correctness of pointer programs, both iterative and recursive.
- Evaluate the effectiveness of the cooperation in terms of verification automation on applications where user interaction is currently required.

The research programme has been broken down into 5 Work Packages (WP). A description of the aims of each WP is given below together with a breakdown of the tasks, deliverables and responsibilities. The research will require a Postdoctoral Research Associate (RA) and a Research Student (RS) for 36-months. A diagrammatic project plan that details the WPs and milestones is provided in the attached Work Plan.

**WP1: Extensible shape analysis** [Months 1 to 12: Total 12 person months] Develop an extensible program analyzer, called NuSmallfoot, which supports shape invariant discovery and is compatible with IsaPlanner [ 100% RS ].

**Tasks:** Building upon Smallfoot (and related tools), develop a program analyzer, which supports shape invariant discovery and is compatible with IsaPlanner (T1); Investigate how IsaPlanner's inductive reasoning capabilities can be used to make NuSmallfoot extensible (T2).

**Deliverables:** An experimental program analyzer (NuSmallfoot) (D1); Evidence of NuSmallfoot's loop invariant discovery capabilities and ideas on how to make NuSmallfoot extensible (D2). A system description and conference paper describing D1 and D2 (D3).

**WP2: Separation logic for VerisoftVE** [Months 1 to 12: Total 12 person months] Extend VerisoftVE with separation logic, *i.e.* provide a basis for reasoning about the correctness of pointer programs [ 100% RA ].

[2] `http://www.verisoft.de`

**Tasks:** Extend VerisoftVE with separation logic (T3); Building upon T3, model a basic pointer programming language within the extended Verisoft environment (T4).

**Deliverables:** Extended VerisoftVE capable of modelling and reasoning about the correctness of pointer programs (D4). A conference paper describing the extension (D5).

**WP3: Proof Plans for Separation Logic** [Months 13 to 24: 21 person months] The aim is to adapt and extend the existing IsaPlanner proof plans for use within VerisoftVE and NuSmallfoot. In terms of theorem proving, the focus will be on invariant verification, rather than discovery. Automating the discovery of invariants via proof planning will be addressed in WP4 [ 75% RA + 100% RS ].

**Tasks:** Adapt IsaPlanner's planning mechanism so that it can automate the search for proofs within VerisoftVE (T5); Extend IsaPlanner's proof plans for induction, rippling and fertilization to deal with pointer references (T6) and the kinds of inductive predicates that are required when reasoning about separation logic specifications (T7); Evaluate the effectiveness of D6 and D7 in controlling the search for verification proofs within separation logic (T8).

**Deliverables:** An extensible version of NuSmallfoot (D6); An IsaPlanner adapted for VerisoftVE (D7); An extended library of proof plans (D8); Conference and journal papers describing D6, D7 and D8 and an initial evaluation (D9).

**WP4: Meta-Level Cooperation** [Months 19 to 30: 6 person months] The aim is to develop meta-level control techniques that automate the discovery of frame axioms and loop invariants via cooperation [ 50% RA ].

**Tasks:** Develop assertion refinement in support of frame axiom instantiation and integrate with NuSmallfoot (T9); Develop middle-out reasoning in support of content invariant discovery and integrate with NuSmallfoot (T10).

**Deliverables:** A NuSmallfoot and IsaPlanner integration which supports automatic, i) loop invariant discovery, and ii) assertion refinement (D10); A system description and conference paper describing the integration (D11).

**WP5: Evaluation Phase** [Months 25 to 36: 21 person months] The aim is to consolidate and evaluate the effectiveness of the cooperation that is achieved via the NuSmallfoot and IsaPlanner integration. In particular, we will revisit the loop invariants and frame axioms mentioned in WP4 in order to evaluate the discovery capabilities resulting from the integration [ 75% RA + 100% RS ].

**Tasks:** Evaluation based upon standard pointer programs and library routines, as well as challenge examples drawn from the literature and other related projects (T11); Thesis writing (T12).

**Deliverables:** PhD Thesis (D12); A conference paper describing the results of T11 (D13); A journal paper describing the achievements of project as a whole (D14); Final report (D15).

## 2.3 Relevance to Beneficiaries

The immediate beneficiaries of the work will be researchers working in the areas of separation logic, program analysis and automated theorem proving. As mentioned previously, our goal is to develop a generic framework. Having such a framework will accelerate further investigations into novel integrations of program analysis and theorem proving techniques. In addition, we would expect our results will be of interest to the wider automated deduction and formal methods communities, as well as the program analysis community. If our cooperative style of integration is successful, then it could have a significant impact on reducing the cost of developing highly reliable software.

## 2.4 Dissemination and Exploitation

We will seek to publish our results in high quality journals and at the relevant major international conferences. Note that we will target the general software engineering community, *e.g.* ASE & ICSE, as well as the more specialized formal methods community, *e.g.* FME & IFM. We will also target relevant international workshops such as WING, Verify and ARW. We anticipate at least 7 conference and 2 journal publications, as well as a PhD Thesis, to be generated by this project. The system itself and associated deliverables will be made available via the web. Moreover, we would seek to publicize our techniques by giving tutorials at international conferences. As with our previous NuSPADE project, we will continue to work closely with Heriot-Watt's Technology Transfer Services in order to capitalize on any commercial opportunities that may arise from the work.

## References

[1] J. Barnes. *High Integrity Software: The SPARK Approach to Safety and Security.* Addison-Wesley, 2003.

[2] J. Berdine, C. Calcagno, B. Cook, D. Distefano, P. O'Hearn, T. Wies, and H. Yang. Shape analysis for composite data structures. 2007. To appear at CAV'07.

[3] J. Berdine, C. Calcagno, and P. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *FMCO*, LNCS Vol 4111, Springer, 2005.

[4] J. Berdine, C. Calcagno, and P. O'Hearn. Symbolic execution with separation logic. In *APLAS*, 2005.

[5] A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th International Conference on Automated Deduction*, Springer, 1988.

[6] A. Bundy. Cooperating reasoning processes: more than just the sum of their parts. In M. Veloso, editor, *Proceedings of IJCAI 2007*, IJCAI Inc, 2007. Acceptance speech for Research Excellence Award.

[7] A. Bundy, A. Smaill, and J. Hesketh. Turning eureka steps into calculations in automatic program synthesis. In S. L. H. Clarke, editor, *Proceedings of UK IT 90*, IEE, 1990.

[8] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62, 1993.

[9] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *J. of Automated Reasoning*, 7, 1991.

[10] 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*, LNAI Vol 449, Springer, 1990.

[11] C. Calcagno, M. Parkinson, and V. Vafeiadis. Modular safety checking for fine-grained concurrency. In *To appear in the Proceedings of SAS 2007*, 2007.

[12] A. Cook, A. Ireland, G.J. Michaelson, and N. Scaife. Discovering applications of higher order functions through proof planning. *Journal of Formal Aspects of Computing*, 17(1):38–57, 2005.

[13] A. Cook, A. Ireland and G.J. Michaelson. Higher order function synthesis through proof planning. In *Proceedings of the* $16^{th}$ *IEEE International Conference on Automated Software Engineering.* IEEE Computer Society, 2001.

[14] D. Distefano, P.W. O'Hearn, and H. Yang. A local shape analysis based on separation logic. In *Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS*, LNCS Vol. 3920, Springer, 2006.

[15] L. Dixon and J. D. Fleuriot. IsaPlanner: A prototype proof planner in Isabelle. In *Proceedings of CADE'03*, LNCS Vol. 2741, 2003.

[16] B. J. Ellis and A. Ireland. Automation for exception freedom proofs. In *Proceedings of the* $18^{th}$ *IEEE International Conference on Automated Software Engineering*, pages 343–346. IEEE Computer Society, 2003.

[17] B.J. Ellis and A. Ireland. An integration of program analysis and automated theorem proving. In E.A. Boiten, J. Derrick, and G. Smith, editors, *Proceedings of IFM-04*, LNCS Vol. 2999, Springer, 2004.

[18] 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.

[19] H.H. Goldstine and J. von Neumann. Planning and coding of problems for an electronic computing instrument. In A.H. Taub, editor, *J. von Neumann: Collected Works*, pages 80–151. Pergamon Press, 1963. Originally, part II, vol. 1 of a report of the U.S. Ordinance Department 1947.

[20] B. Gorry, A. Ireland, and P. King. PARTES: Performance analysis of real-time embedded systems. In *To appear in the Proceedings of QEST'07*, 2007.

[21] A. Hall and R. Chapman. Correctness by construction: Developing a commercial secure system. *IEEE Software*, 19(2), 2002.

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

[23] 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*, LNAI Vol. 624. Springer, 1992.

[24] A. Ireland. Towards increased verification automation for high integrity software engineering. *ERCIM News: Special Issue on Automated Software Engineering*, (58), July 2004.

[25] A. Ireland. Towards automatic assertion refinement for separation logic. In *Proceedings of the* $21^{st}$ *IEEE International Conference on Automated Software Engineering.* IEEE Computer Society, 2006.

[26] A. Ireland. A cooperative approach to loop invariant discovery for pointer programs. *Presented at 1st International Workshop on Invariant Generation (WING) 2007, a satellite workshop of Calculemus 2007*, 2007. To appear within the RISC (Hagenberg) Technical Report Series.

[27] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *J. of Automated Reasoning*, 16(1–2), 1996.

[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), 1999.

[29] A. Ireland, B. J. Ellis, A. Cook, R. Chapman, and J. Barnes. An integrated approach to high integrity software verification. *J. of Automated Reasoning: Special Issue on Empirically Successful Automated Reasoning*, 36(4), 2006.

[30] A. Ireland, B. J. Ellis, and T. Ingulfsen. Invariant patterns for program reasoning. In R. Monroy, G. Arroyo-Figueroa, L.E. Sucar, and H. Sossa, editors, *Proceedings of MICAI-04*, LNAI Vol 2972, Springer, 2004.

[31] A. Ireland, M. Jackson, and G. Reid. Interactive proof critics. *Formal Aspects of Computing: The International Journal of Formal Methods*, 11(3), 1999.

[32] 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.

[33] A. Ireland and J. Stark. Proof planning for strategy development. *Annals of Mathematics and Artificial Intelligence*, 29(1-4), 2001.

[34] A. Ireland and J. Stark. Combining proof plans with partial order planning for imperative program synthesis. *Journal of Automated Software Engineering*, 13(1):65–105, 2005.

[35] C.B. Jones, P. O'Hearn, and J. Woodcock. Verified software: A grand challenge. In *IEEE Computer*, 2006.

[36] S. King, J. Hammond, R. Chapman, and A. Pryor. Is proof more cost effective than testing? *IEEE Trans. on SE*, 26(8), 2000.

[37] G. Klein, H. Tuch, and M. Norrish. Types, bytes, and separation logic. In M. Hofmann and M. Felleisen, editors, *Proc. 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'07)*, 2007. To appear.

[38] S. Magill, A. Nanevski, E. Clarke, and P. Lee. Inferring invariants in separation logic for imperative list-processing programs. In *Proceedings of SPACE'06*, 2006.

[39] N. Marti, R. Affeldt, and A. Yonezawa. Formal verification of the heap manager of an operating system using separation logic. In *Proceedings of ICFEM'06*, LNCS Vol. 4260, Springer, 2006.

[40] Microsoft. *Static Driver Verifier (SDV)*. `http://www.microsoft.com/whdc/devtools/tools/sdv.mspx`.

[41] H.H. Nguyen, C. David, S. Qin, and W.N. Chin. Automated verification of shape and size properties via separation logic. 2007. To appear at VMCAI'07.

[42] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, LNCS Vol 2283, Springer, 2002.

[43] P. O'Hearn, J. Reynolds, and Y. Hongseok. Local reasoning about programs that alter data structures. In *Proceedings of CSL'01*, LNCS Vol. 2142, Springer, 2001.

[44] S. Owre, N. Shankar, and J. Rushby. PVS: A prototype verification system. In D. Kapur, editor, *Proceedings of CADE-11*. LNAI Vol. 607, Springer, 1992.

[45] V. Preoteasa. Mechanical verification of recursive procedures manipulating pointers using separation logic. TUCS Technical Report 753, Turku Centre for Computer Science, 2006.

[46] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Logic in Computer Science*. IEEE Computer Society, 2002.

[47] N. Schirmer. A Verification Environment for Sequential Imperative Programs in Isabelle/HOL. In G. Klein, editor, *Proc. NICTA Workshop on OS Verification 2004*, 2004.

[48] K. Slind, M. Gordon, R. Boulton, and A. Bundy. System description: An interface between CLAM and HOL. In C. Kirchner and H. Kirchner, editors, *15th International Conference on Automated Deduction*, LNAI Vol. 1421, Springer, 1998.

[49] J. Stark and A. Ireland. Invariant discovery via failed proof attempts. In P. Flener, editor, *Logic-Based Program Synthesis and Transformation*, LNCS Vol. 1559, pages 271–288. Springer-Verlag, 1998.

[50] 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.

[51] T. Weber. Towards mechanized program verification with separation logic. In J. Marcinkowski and A. Tarlecki, editors, *Computer Science Logic – 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL, Karpacz, Poland, September 2004, Proceedings*, LNCS Vol. 3210 Springer, 2004.