Automatic Guidance for the Formal Verification of High Integrity Ada
Final Report of EPSRC Grant GR/R24081

Andrew Ireland

1 Background/Context

The SPARK Approach[1] to high integrity software development advocates "correctness by construction", where the focus is on bug prevention rather than bug detection. The SPARK Approach has been applied successfully across a wide range of applications including, railway signalling, smartcard security and avionics systems, such as the Lockheed C130J and EuroFighter projects. The approach has been recently (April 2004) recognized by the US National Cyber Security Partnership as one of only three software development processes that can deliver sufficient assurance for security critical systems.

The SPARK Approach is based upon the SPARK programming language, which is defined in terms of an Ada subset. The subset is expressive enough for industrial applications, but restrictive enough to support rigorous analysis early in the development process. In particular, SPARK supports a language of program annotations and associated tools. The annotations allow the programmer to specify the intended behaviour of their programs. The process of proving that a program satisfies its specification involves the generation of *verification conditions*, *i.e.* a set of logical conjectures that are mechanically derived from a program and its specification. To prove the correctness of the program, all the verification conditions must be proved to be true. The SPARK Approach supports *partial correctness proof*, *i.e.* proving the functional correctness of a program. However, the formal verification capabilities of SPARK are most commonly used for what are known as *exception freedom proofs*, *i.e.* proving that a system is free from run-time errors. This property based approach to code-level verification is becoming increasingly popular, *e.g.* other notable successes in this area are the SLAM (Microsoft Research) and ESC/Java (HP Labs) projects. Within safety critical applications, run-time errors may give rise to catastrophic failures, *e.g.* an integer overflow run-time error led to the loss of Ariane 5. The same holds true for security critical applications, *e.g.* buffer overflows have been the most common form of security vulnerability in the last ten years. The ability to verify that software applications are free of such undesirable behaviour has obvious social and economic benefits. The notion of run-time error conditions is built into the SPARK tool that generates verification conditions, so programmers are not burdened with the specification task. In addition, the SPADE Simplifier, a special purpose automated reasoning tool, has had significant success in automating exception freedom proofs. However, as will be explained below, a significant number of exception freedom proofs will require additional program properties (annotations) to be hand-crafted by the programmer. Moreover, where the SPADE Simplifier fails to automatically prove a verification condition, the burden of proof falls to the programmer, *i.e.* the programmer must interactively construct a proof using the SPADE Proof Checker. The need for additional properties and interactive proof represents two major bottle-necks within the SPARK Approach.

Within the NuSPADE[2] project, the goal of our research has been to address the bottle-necks highlighted above, and increase the level of proof automation in general. Our starting point is proof planning, which is a computer-based technique for automating the search for proofs. At the core of the technique are high-level proof outlines, known as proof plans. Proof plans are an extension to tactic-based proof. A proof plan encodes the heuristic knowledge that is used in automating the search for proof via the construction of a proof tactic. A key feature of proof planning is that it separates proof search from proof checking. This gives greater flexibility in the strategies that can be used in guiding proof search as compared to conventional proof development environments. An example of this greater flexibility is the proof critics mechanism that supports the automatic analysis and patching of proof planning failures.

2 Key Advances and Supporting Methodology (Research Quality)

Our research hypothesis was:

> *The coupling of the Clam Proof Planner 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.*

[1] The SPARK Approach is developed by Praxis High Integrity Systems Ltd, formally known as Praxis Critical Systems Ltd and referred to here as simply "Praxis".

[2] SPADE is the name given to the proof tools associated with the SPARK language. NuSPADE, pronounced "new spade", is the name we have given to the extended proof tools.

Through our investigations, we believe that we have made significant progress towards validating this hypothesis. Two key objectives arose from the hypothesis: Firstly, to increase the level of proof automation, and secondly, to reduce the amount of hand-crafted program annotations that are required in order to support the proof process.

From our initial investigations we concluded that focusing solely on proof planning was going to be too restrictive. This led us to explore the interplay between proof planning and program analysis. Through the use of proof critics we were able to develop a tight integration between proof search and program analysis. We identified common patterns of proof-failure with constraints on missing properties. These constraints are used by our program analyzer in automatically generating the missing properties, *i.e.* additional SPARK program annotations. Typically these annotations are crafted by the programmer after they have analyzed a proof-failure. Moreover, the additional program annotations typically increase the burden of interactive proof required of the programmer. Our work has reduced this burden through the development of new proof plans and the reuse of proof plans from other application domains. As well as using proof-failure analysis to constrain property generation, we have also used property generation to constrain the proof patching process (see §2.2). Originally proof patching focused solely on the deductive processes, *i.e.* conjecture generalization and lemma discovery. Within the NuSPADE project we have broadened the role of proof planning, *i.e.* we have used the proof planning framework to represent knowledge about programs and proofs for use in automating program proof. Below we outline the various aspects of our integrated approach and its interactions.

2.1 Proof Planning

2.1.1 Exception Freedom Proofs

We developed a new proof plan and associated proof critics for exception freedom proofs [1, 2, 6]. The new plan is required when proving that a variable does not exceed its legal bounds, *e.g.* an array index can never exceed the range of the associated array or a variable can never be assigned a value outside its legal bounds. At the core of the proof plan is a proof by transitivity in which the bounds on an expression are identified via a process of decomposition. The proof plan can fail in a variety of ways, and each failure is identified with a particular patch. Four new proof critics were developed to support proof-failure analysis and patching. Constraint solving plays a key role in two of the proof critics, *i.e.* it was used to identify why verification conditions were unprovable. The counter-example generated by the constraint solving is then used to guide the search for auxiliary program properties, *i.e.* loop invariants, that will progress the search for a proof. Proof critics provide the interface between the proof planner and our program analysis techniques. In addition to the proof plan for exception freedom, we were able to reuse aspects of a proof plan developed for proof by mathematical induction. In particular the ripple proof method was used during the verification of our generated loop invariants. Moreover, proof by induction and conjecture generalization also found a role within our exception freedom proof investigations. Although the immediate reuse of proof plans across applications was not unexpected, it was encouraging to see it in practice.

2.1.2 Partial Correctness Proofs

In terms of partial correctness proofs, we focused mainly on array based programs. It was encouraging to see that the ripple method was applicable when reasoning about loop invariants. This adds weight to the argument that proof planning promotes the reuse of strategies across applications and logics. This new application of the ripple method also revealed two new generalization proof critics [7].

The first critic, which we call *range generalization*, corresponds to the failure to prove a transitive relation between adjacent elements of an array. Moreover, the expressions corresponding to the index terms must contain a counter variable in common. The associated patch suggests generalizing the failed proof obligation so that a range of elements is considered. Such a generalization step represents the introduction of an auxiliary invariant.

The second critic is called *difference generalization* and is tightly coupled with the ripple method. Previously the failure of the ripple method has been used to guide conjecture generalization [5]. Here our proof-failure analysis led to a different kind of generalization step, *i.e.* the use of term structure hiding to guide the search for auxiliary invariants. Typically, multiple generalizations will be suggested by this analysis.

Both critics use information gathered during program analysis to constrain the proof patching process. Details of the program analysis that we use are given in §2.2.

2.2 Program Analysis

The goal of our program analysis is to automatically generate properties that will progress the proof process. During the course of the project two property generation systems were constructed.

The first system, called AutoGap [7, 3], supports the generation of two distinct kinds of properties. Firstly, it generates simple loop invariants that are required in order to complete a proof. Secondly, it generates information about program variables, *e.g.* knowing that a particular program variable monotonically increases within a given loop. Such "meta-data" is used during proof planning to generate schematic loop invariants, *i.e.* the program knowledge is used to constrain proof patching. In this way the role of proof planning has been extended to integrate algorithmic patterns with proof patterns. This allows for an incremental style of invariant generation, which is in direct contrast to early work in this area which followed a generate-and-test approach.

The second system, called PropGen [1, 2, 6], extends the property generation capabilities of AutoGap. While AutoGap targeted partial correctness proof, the development of PropGen was driven by exception freedom proofs. We targeted the exception freedom proofs where the SPADE Simplifier failed. The analysis performed by PropGen involves the translation of SPARK source code into a conventional flow graph representation. Based upon this representation, a range of analysis methods were developed that automatically generate properties which support exception freedom proof. One of the key methods is based upon the use of *recurrence relations*. Given the nature of exception freedom proofs, we introduced the notion of *extreme recurrence relations*. These relations enabled us to model the upper and lower bounds of a variable on an arbitrary iteration, *i.e.* precisely the information which is required in order to discharge exception freedom verification conditions generated from loop-based code. We exploited existing tools to solve the extreme recurrence relations and support constraint solving. In addition, lightweight equational reasoning was supported via proof planning.

2.3 Low-level Integration Challenges

The fact that the SPADE Proof Checker and the Clam Proof Planner are implemented in Prolog significantly eased the integration process. In particular, the translation of the various SPADE data files was relatively straight forward. What we had not anticipated was the complexity involved in translating the tactics generated by the proof planner into command files for execution within the SPADE Proof Checker. Tactics are essentially programs that control the application of low-level inference rules. The SPADE Proof Checker is not a tactic-based proof checker. As a result, proof checking involves translating the proof planner generated tactics into SPADE command files, *i.e.* a sequential list of commands that are executed automatically within a SPADE Proof Checker session. All the information that is required in checking the proof must be specified in the command file. The SPADE Proof Checker is more than just a "proof checker"; it attempts to automate many of the small steps within the development of a proof. This can be very useful when interactively developing a proof. However, when used purely as a proof checker, these "automated steps" greatly complicate the proof planning process. To overcome this problem we added additional commands to the SPADE Proof Checker which, in effect, make it more controllable, *i.e.* the automated steps are switched-off.

2.4 Results

Our evaluation predominantly focused on exception freedom proofs. The reason for this was that exception freedom proofs are a major component of the SPARK development process while partial correctness proofs are extremely rare. The analysis of NuSPADE drew upon two sources of data. Our first source was provided by Praxis while the second source of examples were drawn from text books. Praxis provided access to verification conditions arising from two safety critical industrial applications. While the details are confidential, one of the industrial applications was the Ship Helicopter Operating Limits Information System (SHOLIS) [9]. SHOLIS was the first system developed to meet the UK Ministry of Defence Interim Defence Standards 00-55 [11] and 00-56 [10]. Our techniques are aimed at addressing the exception freedom verification conditions on which the SPADE Simplifier fails. In particular, we were concerned with failures that related to loop-based code. While industrial strength critical software systems are engineered to minimize the number and complexity of loops, we found that 80% of the loops that we did encounter were provable using our techniques. That is, our program analysis, guided by proof-failure analysis, automatically generated auxiliary program annotations that enabled subsequent proof planning and proof checking attempts to succeed. Two key reasons were identified for the 20% of loops that our techniques failed to prove. Firstly, in some situations a stronger precondition to the enclosing subprogram was required in order to complete a proof. Secondly, our program analysis is sometimes too coarse grained, *e.g.* insufficient discrimination between conditional branches. These limitations, however, represent opportunities for future work. The details of the results are presented in [6].

In terms of partial correctness, our evaluation was based upon examples drawn from the literature. The AutoGap system was successfully applied to around twenty examples including, bubble sort, the Find algorithm, and a prime number generator algorithm [3, 7].

3 Research Impact and Benefits to Society

Proof planning is a knowledge based approach to automated theorem proving. In terms of scientific impact, the NuSPADE project has broadened the role of proof planning, *i.e.* we have extended the knowledge based aspect to include algorithmic, as well as proof knowledge. More generally, we have demonstrated the merits of proof planning as a framework for integrating formal methods, as well as reusing high-level proof strategies.

In terms of integration, we have had discussions with Dr Predrag Janicic (Faculty of Mathematics, Belgrade) in relation to integrating his decision procedure framework within our work. We have also had discussions with Bernd Fischer (NASA Ames) on comparing our NuSPADE work with the AutoBayes certified automatic synthesis system.

In terms of industrial impact, and the wider benefits to society, it is too early to tell. However, we have secured EPSRC funding for a follow-on project that will provide the first step towards technology transfer (see §4.2). In addition, a proposal is currently being developed with Dr Lau (Manchester University) and Praxis on component based software development which will build upon the NuSPADE project (see §4.2).

4 Project Plan Review (Research Planning and Practice)

As mentioned above, the majority of our efforts were focused on exception freedom proofs, rather than partial correctness proofs. This choice reflects not only the emphasis on exception freedom proofs within the SPARK Approach, but also a wider trend towards property based code level verification *e.g.* SLAM (Microsoft Research) and ESC/Java (HP Labs). The original project proposal emphasized automatic guidance for program assertion (annotation) and lemma discovery. As the project evolved we found that program annotation discovery was the more significant issue. As a consequence we did not pursue the goal of lemma discovery.

4.1 Output of Research Staff

One post-graduate researcher, Mr Bill Ellis, was employed as the RA funded by this grant (RA1A). The original grant application requested funds to employ a named post-doctoral researcher. However, because of delays in the grants announcement, the post-doctoral researcher named in the original application was no longer available for employment. Following extensive advertising of the post, no suitable replacement post-doctoral researcher was identified. It should be noted that Mr Ellis has been highly effective as the RA on the project. Moreover, Mr Ellis has been studying for a PhD during the period of the grant. While his PhD is part-time, he is now currently writing-up with a plan to submit his thesis early in 2005.

The budget saving that resulted in the appointment of a post-graduate researcher was put to good use in that it enabled two additional researchers to be employed on short-term contracts. Firstly, an undergraduate student, Mr Tommy Ingulfsen, was employed for 2 months during the summer vacation of 2002. Mr Ingulfsen developed a Short Term Conflict Alert (STCA) system for a fictional air traffic control system. This software provided a significant corpus of exception freedom verification conditions. Moreover, Mr Ingulfsen's work led to the subsequent development of AutoGap, a static analysis tool for the automatic verification of software written in SPARK. The AutoGap tool formed the basis for Mr Ingulfsen's BSc final year dissertation within Computer Science at Heriot-Watt. As well as scoring the top mark for his dissertation, Mr Ingulfsen won the Spektra Award at the Young Software Engineer Awards in November 2003 for the AutoGap project. After graduating, Mr Ingulfsen has completed a Masters degree at Cambridge University. Secondly, Dr Andrew Cook was employed for 1.5 months towards the end of the project. Dr Cook's work involved developing the translator that generates SPADE command files from instantiated proof plans. It should be noted that six additional undergraduate and master level projects have arisen because of this project.

4.2 Dissemination Activities and Further Research

Dissemination has been predominantly through academic publications. The grant has produced 4 conference papers; 1 journal paper accepted for publication; 1 journal paper invited for a special issue; 3 workshop contributions. In addition, an invited article was recently published in a special issue of the ERCIM News that focuses on Automated Software Engineering [4]. ERCIM News has a broad European readership, ranging across science and engineering, and therefore provided an excellent vehicle to communicate the results of our project beyond the boundaries of our core academic publications.

In addition to publications, conference and workshop presentations have arisen from our work. In particular, we presented progress reports of our research at the following events:

- 8th Workshop on Automated Reasoning, York, 2001.

- Workshop on Automated Verification of Critical Systems (AVoCS-01), Oxford, 2001.
- Workshop on Automated Verification of Critical Systems (AVoCS-02), Birmingham, 2002.
- Technical Cooperation Panel (TTCP) Workshop: NATO Defence related system & software safety issues, York, 2002.
- FACS/FORTEST Meeting, York, 2002.
- VAST meeting, Brunel, 2003.
- NASA Ames Workshop on Frontiers of Automated Software Engineering, Mountain View CA, 2003.
- 18th IEEE International Conference on Automated Software Engineering (ASE), Montreal, 2003.
- FORTEST Meeting, Bath, 2003.
- CIAO-03, Dagstuhl, Germany, 2003.
- Grand Challenges of Computing Conference: Dependable Evolving Systems, Newcastle 2004.
- 4th International Conference on Integrated Formal Methods, Canterbury, 2004.
- 3rd Mexican International Conference on Artificial Intelligence, Mexico City, 2004.

Recently Dr Ireland has been invited to present the results of the project at a SPARK User Group Meeting. This will provide an excellent opportunity to communicate the results of our research to the SPARK practitioners within industry. More locally, progress on the project was presented through the Scottish Theorem Proving (STP) seminar series. The STP forum supports automated reasoning research within central Scotland. Dr Ireland's close working relationship with the Mathematical Reasoning Group (MRG) at the University of Edinburgh (see §6) has also provided a good route for dissemination as well as providing critical feedback at key points during the project. The following web page gives more details on the NuSPADE project, including all our papers and presentations:

`http://www.macs.hw.ac.uk/nuspade`

In terms of future work, we are interested in investigating the automatic generation of preconditions. As noted in §2.4, the introduction of a precondition can often be required in order to complete a proof. Again we see a role for using counter-examples generated by constraint solving as a basis for discovering such preconditions. We also wish to strengthen our defect finding capabilities through constraint solving. While constraint solving works well in practice, it was limited to reasoning with integers that lie within the range $-(2^{25}), \ldots, 2^{25}-1$. Dealing with integers outside this range represents an interesting area of future research. An area which we have not considered, and which we believe would yield useful results, is the integration of decision procedures within NuSPADE. The integration of decision procedures within proof planning is an active area of research [8].

A follow-on project starts in January 2005, supported by EPSRC's Research Assistants Industrial Secondment (RAIS) scheme. The aim of the RAIS scheme is to support knowledge transfer where a research project has had a strong industrial collaborative component. Our RAIS project will build upon the results of the NuSPADE project in working towards a verification toolset for industrial use. This will involve extracting from NuSPADE a core set of tools that can be readily extended for use on industrial projects. The RAIS project represents the first step towards technology transfer.

More generally, an EPSRC project proposal is currently in preparation to investigate the development of critical software components in SPARK. This will be a collaborative project between Heriot-Watt University, Manchester University and Praxis. There will be two formal verification aspects to the project proposal. Firstly, software components written in SPARK will be formally verified. Secondly, formal reasoning will be used to support the retrieval and composition of the verified components. The results of the NuSPADE project will provide a key foundation for this new proposal. We believe that such a component based approach is timely as it fits well with the Dependable Evolving Systems theme of the UK Computing Research Committee's Grand Challenges of Computing enterprise.

5 Explanation of Expenditure (Cost Effectiveness)

As discussed in §4.1, this project trained 1 Research Associate (Mr Ellis) during the period of the grant, this involved supporting their studies towards a PhD which is nearing completion. Supported by EPSRC's RAIS Scheme, Mr Ellis will take up a 6 month placement with Praxis early in 2005. Through this knowledge transfer phase, the value of the current project will be further extended. It is possible that Mr Ellis may be offered a position within Praxis at the end of the RAIS placement. In addition to the RA position, the grant employed 1 undergraduate student (Mr Ingulfsen) during the summer of 2002, which led to a prize at the Young Software Engineer Awards 2003 and a major component of the NuSPADE system. The project also funded Mr Cook towards the end of the project when additional programming support was required. As well as funding attendance of conferences and workshops, the grant has also supported our involvement with the STP and FORTEST forums (see §4.2). As part of his training, the grant partially supported Mr Ellis attendance of the Calculemus Autumn School in Pisa (2002). Most importantly, the grant funded a number of project reviews that took place at Praxis in Bath as well as extended visits to Praxis by Mr Ellis during our evaluation phase. We believe that this grant has been highly cost effective. It has had considerable impact, as detailed in §3, and has led to a follow-on knowledge transfer project as well as a proposal for a spin-off project.

6 Related Activities

During the period of the grant Dr Ireland has been involved in the following related activities:

- Co-authored a research monograph entitled "Rippling: Meta-Level Guidance for Mathematical Reasoning", to be published by Cambridge University Press 2005.
- Served on the Programme Committees for Automated Software Engineering, *i.e.* ASE-16, ASE-18, ASE-19, as well as Computer Aided Deduction, *i.e.* CADE-18, CADE-19. In addition, he was Local Arrangement Chair for CHARME-2001 IFIP WG1.5: Advanced Research Working Conference on Correct Hardware Design and the IEEE International Conference on Automated Software Engineering (ASE-17).
- Invited to present the case for the Verifying Compiler proposal, within the Dependable Evolving Systems theme of the Grand Challenges of Computing Conference.
- Member of Formal Methods for Testing (FORTEST), an EPSRC sponsored Network. Given the overlap between static analysis and formal verification, FORTEST provided an excellent forum during the development of our work.
- Co-investigator on EPSRC Platform grant GR/S01771. The principle investigator on this grant is Professor Alan Bundy who leads the MRG. The rolling nature of the MRG funding provided crucial support for the development of the ideas that ultimately led to the NuSPADE project proposal.

References

[1] 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. Also available from the School of Mathematical and Computer Sciences, Heriot-Watt University, as Technical Report HW-MACS-TR-0010.

[2] 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 4th International Conference on Integrated Formal Methods (IFM-04)*, volume 2999 of *Lecture Notes in Computer Science*, pages 67–86. Springer Verlag, 2004. Also available from the School of Mathematical and Computer Sciences, Heriot-Watt University, as Technical Report HW-MACS-TR-0014.

[3] T. Ingulfsen. *Automatic Generation of Algorithmic Properties (AutoGAP)*. Undergraduate BSc Computer Science Project Dissertation, School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, 2003. Available on-line: `http://www.macs.hw.ac.uk/nuspade/publications/`.

[4] A. Ireland. Towards increased verification automation for high integrity software engineering. *ERCIM News: Special Issue on Automated Software Engineering*, (58), July 2004. Available on-line: `http://www.ercim.org/publication/Ercim_News/enw58/`.

[5] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1–2):79–111, 1996.

[6] A. Ireland, B.J. Ellis, A. Cook, R. Chapman, and J. Barnes. An integrated approach to program reasoning. 2004. Submitted to a Special Issue of the Journal of Formal Aspects of Computing on Integrated Formal Methods. Available from the School of Mathematical and Computer Sciences, Heriot-Watt University as Technical Report HW-MACS-TR-0027.

[7] 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 3rd Mexican International Conference on Artificial Intelligence (MICAI-04)*, volume 2972 of *Lecture Notes in Artificial Intelligence*, pages 190–201. Springer Verlag, 2004. Also available from the School of Mathematical and Computer Sciences, Heriot-Watt University, as Technical Report HW-MACS-TR-0011.

[8] P. Janičić and A. Bundy. A general setting for flexibly combining and augmenting decision procedures. *Journal of Automated Reasoning*, 28(3):257–305, April 2002.

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

[10] MoD. Hazard analysis and safety classification of the computer and programmable electronic system elements of defence equipment. Interim Defence Standard 00-56, Issue 1, Ministry of Defence, 1991.

[11] MoD. The procurement of safety critical software in defence equipment (part 1: Requirements, part 2: Guidance). Interim Defence Standard 00-55, Issue 1, Ministry of Defence, 1991.