Bill J. Ellis and Andrew Ireland
October 30, 2003

Experiment: Industrial strength analysis

1 Introduction

NuSPADE has evolved to become two interacting systems. There is a program analysis aspect, called PROPGEN (property generation), and a proof planner aspect, called SPADE-PP (the SPADE proof planner). Two questions are asked:

1. How effective are the properties discovered by PROPGEN in increasing the number of exception freedom VCs automatically discharged?
2. Following PROPGEN how effective is SPADE-PP in discharging the remaining VCs?

Although PROPGEN and SPADE-PP will ultimately be integrated into the single system NuSPADE, we propose two experiments built around these systems working independently. This is more practical and still provides meaningful and valuable results.

2 Experiment A: Test PROPGEN

This experiment aims to evaluate the ability of PROPGEN in discovering properties that then lead to automatic SIMPLIFIER proofs.

1. Get a copy of a High Integrity SPARK Software System (HISSS), for example: SHOLLIS.
2. Strip all proof annotations from the HISSS. This can be achieved with a small AWK script.
3. Pass the stripped HISSS through the EXAMINER generating exception freedom VCs. Pass these VCs through the SIMPLIFIER and then run POGS. The POGS summary describes the starting point, the amount of automation offered by the SPARK tools with zero user interaction.
4. Pass the stripped HISSS through PROPGEN. This will automatically strengthen the HISSS annotations.
5. Pass the strengthened HISSS through the EXAMINER generating exception freedom VCs. Pass these VCs through the SIMPLIFIER and then run POGS. The POGS summary describes the finishing point. This can be compared with the starting point to determine the amount of extra automation offered by applying PROPGEN.

2.1 Notes

2.1.1 Counting VC?

There are some issues is comparing the number of proved VCs against remaining VCs to describe proof automation:

- **When a proof is not a proof -** Proving all of the VCs will lead to a proof of exception freedom. However, some VCs may not be provable, especially if the annotations are too weak. In these cases the unprovable VCs represent two problems to be solved. A stronger property must be inserted and proved and the previously unprovable VC must now be proved with this stronger property. Thus, each unprovable VC could be considered as accounting for two unperformed VC proofs. Of course, were the specification available, the SIMPLIFIER may complete both these proofs.
- **Every conclusion counts -** VCs contain a collection of essentially individual conclusions. Each conclusion discharged by the SIMPLIFIER accounts for a step of automation. However, this automation is only noted if every conclusion of a VC is proved.

*Notes in this series are for ϵ-baked ideas, for $1 \geq \epsilon \geq 0$. Only exceptionally should they be cited or distributed outwith the NuSPADE project (EPSRC GR/R24081).

- **Equivalent specifications -** It is possible to envisage two specifications which are semantically the same yet syntactically different. It is not guaranteed that the SIMPLIFIER or SPADE-PP will behave the same in the presence of these specifications. Thus the style of annotations is a factor in proof automation not revealed in counting VC proofs.

It is difficult to address these issues in a suitable manner. It might be possible to establish close to minimum specifications sufficient for complete proof of exception freedom. The annotations might then be compared with the annotations added by PROPGEN. The SIMPLIFIER report of the automatic proof of the provable code is more truthful, as is any increased automation offered by SPADE-PP on top of this. However, identifying the minimum specifications sufficient for proof would be difficult, possibly impossible since different syntactic presentations of the same semantic specification can lead to different amounts of proof automaton. Further, comparing these minimum specifications with the generated specifications would be difficult to automate.

Thus, although not perfect, the SPARK approach of counting VCs will be adopted. The alternatives would be difficult to formulate and even more difficult to implement.

2.1.2 Necessary tweaking

It is likely that our program analysis tool will require a fair amount of tweaking to behave appropriately. This may even stretch to some hand carving of the original SPARK, for example removing any low level or hidden code.

2.1.3 Execution time not a large concern

Running the SIMPLIFIER is time consuming. However, it is likely to be far less time consuming than completing the required tweaking to get the experiment ready. Thus the several hours execution time required for these experiments are not considered an issue.

3 Experiment B: test SPADE-PP

This experiment aims to evaluate the ability of SPADE-PP in automating VCs not proved by the SIMPLIFIER following specifications being discovered by PROPGEN.

1. Perform Experiment A.
2. Apply SPADE-PP to the remaining VCs.
3. Run POGS. The result can be compared with the finishing point of experiment A to determine the additional VCs proved.

3.1 Notes

3.1.1 Test existing proof methods

Experiment B will provide us with an opportunity to test our existing proof methods, such as ripple and trans.

3.1.2 Learn from experiment A

Experiment B may be performed a few months after experiment A. This would allow us to learn from the program analysis, identifying the patterns of VCs it creates. Such results will provide a strong indication of new proof methods to build into the proof planner. These new methods can be developed and tested alongside our existing methods.

3.1.3 Eventual integration

Experiments A and B will give insights into the integration of PROPGEN and SPADE-PP.

4 Conclusion

Crucial to the success of the NuSPADE project is the ‘industrial evaluation’ phase (workpackage WP5 in the proposal). Here we propose an experiment that begins this evaluation phase.