CS-Note* 38

Andrew Ireland
January 28, 2003

VASST Trip Report

1 Introduction

On January 23^{rd} I was invited to attend a meeting of the VASTT[1] group that was being hosted by Brunel University. For more background on VASTT see `http://www.brunel.ac.uk/~csstmmh2/vast/home.html`. There were around 16 participants, including: Mark Harman (Brunel), Rob Hierons (Brunel), Sebastian Danicic (Goldsmiths), Chris Fox (Essex), Norbert Voelker (Essex), Kalpesh Kapoor (Southbank), Lin Hu (Brunel), Ranjit Singh (Brunel), Dave Daoudi (Goldsmiths).

2 My Talk

I presented a general talk on proof planning entitled *Proof Critics: Exploiting Patterns of Failure within Formal Reasoning.* The talk included details of the current NuSPADE project. Below is a summary of the questions I was asked:

Q: What would one have to do in order to turn an Isabelle tactic into a proof plan?

- *Tactic preconditions would have to be defined together with the associated effects that would simulate the result of applying the tactic on the proof state. There is ongoing work within the Edinburgh MRG on proof planning for Isabelle [Lucas Dixon].*

Q: Don't tactics already contain preconditions?

- *Yes. The preconditions represented within a proof plan can typically be embedded within a tactic. However I would argue that the pay-off of decoupling tactic and preconditions comes when reasoning about the failure to find a proof, as I will go on to show ...*

Q: Do the tactics generated contain iteration?

- *No. All search takes place within the proof planner. N applications of a particular rewrite rule will result in a sequence of N explicit rewrite rule tactic applications.*

Q: What is the most sophisticated lemma your system was able to discovery?

- *Within the rotate-length example the lemma speculation critic was able to discovery 2 lemmas corresponding to variants of associativity of list concatenation.*

In the conclusion to my talk I outlined a proposal for exploiting proof failure in the search for effective test cases. This idea was received positively by the audience, in particularly by those involved on testing.

3 Conditioned Program Slicing

Chris Fox presented ideas on conditioned program slicing, where:

- Slicing corresponds to deleting statements that do not effect a given subset of the program variables. Slicing can be applied either forward or backward from a given location within the code. Slicing can also take into account the input values of variables, and is called dynamic slicing. Slicing that does not take into account input values is called static.
- Conditioned slicing can be either forward or backward. Forward conditioned slicing deletes statements which are not executed when the initial state satisfies the condition. Backward conditioning deletes statements which cannot cause execution to enter a state which satisfies the condition. Conditioned slicing can be seen as a compromise between static and dynamic slicing.

*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).

[1] Verification and Analysis using Slicing, Testing and Transformation

Conditioned slicing is implemented in terms of symbolic execution and theorem proving. Symbolic execution locates the paths while theorem proving is used to determine whether or not particular paths are feasible. Chris has experimented with Isabelle for the theorem proving component, however, the over head of using Isabelle made this impractical. The theorem proving required is relatively light-weight and I suggested the use of a decision procedure approach, in particular Predrag Janicic's GS-light system. The work presented involved a toy language imperative language. I believe SPARK would provide an ideal vehicle to see if the ideas scale up.

4 Pre/Post Conditioned slicing

Mark Harman presented an interesting extension to conditioned slicing. Consider some code specified with pre- and postconditions. Now consider applying:

- forward conditioned slicing with respect to the precondition and
- backward conditioned slicing with respect to the negated postconditions

In an ideal world the slice left would correspond to skip, *i.e.* the slice will contain statements which could potentially break the specification. Obviously this process will not succeed in general, loop invariants being the obvious hurdle. However, this represents an interesting approach to partial correctness with minimum theorem proving.