CS-Note* 69

Bill J. Ellis
April 14, 2004

Praxis Retreat: A Targeted Survey

1 Introduction

A high integrity SPARK program is surveyed, targeting examples that are unexpected given the previous survey or expected but still interesting.

2 Why the Simplifier fails

The Simplifier can fail if:

- **Goal not provable -**
  - **Missing proof functions -** In SPARK the behaviour of a function is described through an external proof rule. These proof rules are often absent. Even if available, the Simplifier does not exploit such proof rules.
  - **Specification too weak -** A proof may fail because the specification is too weak. In particular stronger preconditions are often required to be propagated up from code that needs to constrained. That is, a deeply nested subprogram requiring constraints on variables imposes constraints on its callers and their callers and so on. This can lead to a lot of preconditions being required.
  - **Missing complex constant rules -** The Examiner does not currently generate suitable constant rules to complete the proof.
- **Goal is provable -**
  - **Not strong enough -** The goal is not off a form currently proved by the Simplifier. Given the growing strength of the Simplifier, these will involve some subtle reasoning.

*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).

3 Provable VCs not discharged by the Simplifier

These are the VCs not currently proved by the Simplifier and not expected to be proved as a result of the Simplifier changes. Note that these may not actually be provable.

Location	VCs
alarm.checklimits	27, 28, 29
alarm.updaterollpitchlimit	11, 12, 13, 14
basictypes.arccos	8
basictypes.round	1,2,3,4
basictypes.sqrt	3,4,5,6,9,11,15,16,17
coc1p.incrementlogtens	3
coc1p.incrementlogunits	3
coc4p.calcadvice	7,9,10,14,19,20
db.convertdbsholtoshol	38,39,40,41,42
display.updatecones	2
graphics.insertnumber	26, 35, 41, 43, 44, 45, 47, 48[1], 49, 50
graphics.polargraphcoord	1
graphics.showcoc1pageheading	15
graphics.showcoc4pageheading	16, 49, 62, 63
graphics.showcurrentshol	49, 56
graphics.updaterelwindmarker	10
histp.filterpitch.computepitchfilter	4
injp.decrementbyten	3
injp.up10	3
sensor.averagerelwinddirection	8, 13
sensor.averagetruewinddirection	12
sensor.sensorupdate	33
sensor.updaterelwinddir	14
sensortypes.newheading	9
sio5.computeloopbackchecksum	12,13
sio5.copydisplaybuffertoslot5sio	8,9,21,22,28,29
sio5.initiatebdutransmit	19
sio5.initiatefddutransmit	19

4 Zipping through the code

In many cases our ideas have the potential to find assertions used in this system. The problem is not finding new ideas, but demonstrating the ones we have. However, the loop assertions typically only assert type based information. Further, although assertions are added to cut the code this may not be needed to complete a proof. This cutting the code feature is practically non existent in the newer system T, indicating that programmer style and improved proof support are sufficient to deal with the large numbers of VCs, eliminating the need for extra cuts to minimise VC numbers.

The assertions that we do not expect to find but could technically do so typically come from a much deeper understanding of the algorithm. These algorithms tend to be strongly math based and are very infrequent.

[1]This is flagged by Ian as 'I doubt we can automate this'...