Bill J. Ellis
February 2, 2005

The SPADE Helper Architecture

Contents

*Hums are short notes intended for distribution between those involved with EPSRC grant GR/T12289/01. Hums describe ϵ-baked ideas, where $1 \geq \epsilon \geq 0$. (Hum refers to both the Praxis Humming bird and Winnie-the-Pooh's indirect approach to writing: "Poetry and Hums aren't things which you get, they're things which get you.").

1 Introduction

The SPADE Helper will exist within the context of the SPARK toolset. It will necessarily have to integrate both conceptually and practically with the existing SPARK tools. Here the behaviour of the SPARK tools are considered, noting their resulting constraints on the development of the SPADE Helper. Where these constraints are fundamentally unworkable, suitable modifications to the existing SPARK tools will be outlined. Such changes are expected to be small but significant. Finally, an overall architecture linking the SPADE Helper and the SPARK tools will be presented.

2 Key behaviours of the SPARK tools

2.1 The SPARK Examiner

The SPARK Examiner is responsible for analysing SPARK source code and producing verification conditions (VCs) for each subprogram.

2.1.1 Describing a subprograms VCs

A subprograms VCs comprises off:

- **VCG -** The VCs
- **RLS -** Corresponding rules for the VCs
- **FDL -** Corresponding type for the VCs

2.1.2 Locating subprogram's VCs

Subprogram's VCs are typically presented in a directory tree structure off the form:

- **Package directory 1**
- **Package directory 2**
- ...
- **Package directory n**
  - **VCs subprogram 1 (in package directory n)**
  - **VCs subprogram 2 (in package directory n)**
  - ...
  - **VCs subprogram m (in package directory n)**
  - **Package directory n-1 (in package directory n)**
  - **Package directory n-2 (in package directory n)**
  - ...
  - **Package directory n-o (in package directory n)**

Note that there may be a number of subprogram VCs and deeper nested packages in a given package. Thus to uniquely identify a subprogram's VCs the following are required:

- **Full location of package directory**
- **Name of subprogram**

Note that this is the same input used to start the SPADE Simplifier on a subprograms VCs.

2.1.3 Locating the source code

Typically the tree of packages and subprogram VCs will exist directly inside a directory containing all of the source code.

2.2 The SPADE Simplifier

The SPADE Simplifier analyses a subprogram's VCs, automatically proving as many as possible and structurally simplifying the rest.

2.2.1 Amending subprograms VCs

The SPADE Simplifier stores its output by extending the existing description off a subprogram's VCs to:

- **SIV** - The simplified VCs
- **VCG** - The original VCs
- **RLS** - Corresponding rules for the VCs
- **FDL** - Corresponding type for the VCs

Note that the simplified VCs replace the original VCs. Creating a new file, rather than replacing the old file, supports tracing the proof process. Note that the original VCs contain the unsimplified structural form not seen in the simplified VCs. This process of amending the description of VCs by overruling rather than overwriting seems a good policy for the SPADE Helper and would be consistent with the existing tools.

2.3 POGS

The Proof ObliGation Summariser tool (POGS) trawls the tree of packages and subprogram VCs, presenting the proof progress to the user. Although POGS will not directly communicate with the SPADE Helper, its interaction with the other proof tools warrants special attention. In particular, the SPADE Helper should interact with the SPADE Simplifier and the SPADE Proof Checker in a manner that will be accepted by POGS.

2.3.1 POGS and proof review files

Proof review files are used to indicate that VCs have been proved through manual inspection. They conform to the following format (Extract from POGS User Manual):

```
Proof review files have the following grammar:
     PRVfile ::= {Line}
     Line ::= [VCNumber][Comment]end_of_line_character
     VCNumber ::= digit{digit}
     Comment ::= --{non_end_of_line_character}

Example:
For procedure DoIt we would have a file doit.prv as follows

     -- .pvr for procedure DoIt.
     -- The following VCs were proved by review:
     1 -- Proved by method x
     3 -- Proved by method x
     4 -- Proved by method y
     7 -- Proved by method y
     9 -- Proved by method y
     10 -- Proved by method y
```

It would be possible for the SPADE Helper to hijack the proof review mechanism to identify proved VCs. However, as the SPADE Helper performs formal proof, this would be an inconsistent and confusing use of proof review files.

An alternative, and attractive, proposal would be a single proof status file. This would record the progress of each conclusion in each VC. The various proof tools and the user could edit this file to indicate what has been proved, indicating where evidence of this proof can be found.

2.3.2 POGS and the SPADE Simplifier

The Simplifier overrides the original subprogram VCs held in a VCG file by generating a new SIV file. POGS inspects this SIV file to identify the VCs automatically discharged by the SPADE Simplifier.

2.3.3 POGS and the SPADE Proof Checker

The Proof Checker generates a proof log (PLG) file during the interactive proof of a subprograms VCs. This log is inspected by POGS to identify any VCs proved in the SPADE Proof Checker. Although the proof log is fundamental to the documentation of proved VCs it is easily deleted by starting a new Proof Checker session[1]. Further, the proof log is used to indicate *interactive* proofs. Although the SPADE Helper will control the SPADE Proof Checker, this is a fundamentally different operation. Exploiting the PLG mechanism for the SPADE Helper seems both difficult and inappropriate.

3 Considering the SPADE Helper

3.1 Minimise scope of changes

The SPARK tools may be extend to recognise the SPADE Helper as being distinct from the SPADE Simplifier and the SPADE Proof Checker. This would require changes to POGS and, more significantly, changes to the existing interaction of the SPARK tools. Recently, Praxis have completed the SPARK toolset 7.2 and are beginning to turn their attention to:

- **Integrated development environment** - Creating a top-level tool providing a unified integrated interface to the various SPARK tools.
- **SPADE Simplifier changes** - Continue work on improving the SPADE Simplifier inference engine to discharge further categories of problems.
- **Reappraise the proof tools** - Pursue a rational reconstruction of the proof tools. In particular, extract a core VC management library from the SPADE Simplifier and the SPADE Proof Checker. Further, the general problem of VC management will be considered, especially in the context of a new integrated development environment.

Thus, the SPARK tools, and proof tools in particular, will soon be modified with respect to a global, coherent, tool refinement. The changes that might be proposed for the SPADE Helper would be confusing and (realistically) rather poor in comparison. Thus as few changes as possible should be made to the SPARK tools in view of getting the SPADE Helper to operate. Larger changes to the SPARK tools may be considered where orchestrated by Praxis and the SPARK team.

Note that, in terms of intellectual property agreements, it is generally more agreeable if software produced for the project is in easily distinguishable sections which are not integral to the behaviour of the SPARK tools. Although it is strongly suspected that the genuinely valuable components of the project will be intangible (such as ideas and training) it is prudent to keep the tangible sections of the project as organised and as distinct as possible.

3.2 The SPADE Helper as a seamless extension to the SPADE Simplifier

If minimising the changes made to the proof tools, the SPADE Helper must appear to the SPARK world as being an extended SPADE Simplifier or an extended SPADE Proof Checker. As the SPADE Simplifier performs a similar task of proof automation, behaving as an extended SPADE Simplifier seems the most appropriate. The user will execute:

```
simplifier directory file
helper     directory file
```

And the resulting SIV file will reflect the total amount of automated proof.

3.3 The SPADE Simplifier should only simplify

The SPADE Helper requires access to the original structural form of VCs. An extra switch might be added to the SPADE Simplifier to preserve the structure of the outputted unproved VCs. However, those VCs not discharged by the SPADE Helper will have the structurally preserved form and the user may prefer the simplified form. Further, if the SPADE Helper is acting as a seamless extension of the SPADE Simplifier then any changes (desired or not) should be at least configurable. Thus the SPADE Helper should have access to both the original and the simplified VCs. This might be achieved by tweaking the SPADE Simplifier to output both the original and simplified VCs in a form directly retrievable by the SPADE Helper. This is leaning towards a tighter integration between the SPADE Simplifier and SPADE Helper than originally considered. Essentially, the SPADE Simplifier is becoming increasingly and unnecessarily aware of the SPADE Helper.

[1] Note this deleting might be overcome by using resume. This is untested.

3.4 The SPADE Helper Parser

The various files used to describe a subprogram's VCs are not held in a form immediately accessible by Prolog. Parsers for these files exist within the SPADE Simplifier, the SPADE Proof Checker and NuSPADE. The SPADE Helper will require these parsing services also. To avoid burdening the prove centric SPADE Helper with such details and avoid extending the SPADE Simplifier with unnatural extensions, a separate, small parsing component may be considered. This small parsing component may access the simplified SIV file in combination with the unsimplified VCG file to retrieve the required information for the SPADE Helper. Alas this is difficult at the moment due to subtle behaviour of the SPADE Simplifier. The SPADE Simplifier renumbers conclusions in its output, making it very difficult to relate the SIV file to the original VCG file. If the SPADE Simplifier did not do this, then there should not be a problem in building this small parser program. Addressing this would make for a small targeted change to the SPADE Simplifier.

Note that the SPADE Helper Parser would provide additional parsing services, also considering the FDL and RLS (and RUL) files.

3.5 The SPADE Helper Verifier

A proof plan discovered by the SPADE Helper is only deemed acceptable once it has been checked by the SPADE Proof Checker. This checking phase is fundamentally separate to the proof planning phase. It is natural to consider this process in a separate system. Thus the SPADE Helper Verifier can act as a checker management component, checking discovered plans and producing the resulting SIV file based on the success or otherwise of these plans.

Note that the SPADE Helper Verifier should directly modify the SIV file generated by the SPADE Simplifier to appear as a seamless extension to the SPADE Simplifier. This is unfortunate. The SPARK tools favour modification through overriding rather than overwriting. Further, if executing the SPADE Helper Parser a second time the updated SIV file will be processed as if the initial SIV file, opening up the possibility of unusual system configurations. It should be ensured that the SPADE Helper Parser is not executed after the SIV file has been updated by the SPADE Helper Verifier[2].

3.6 Turning the SPADE Proof Checker into a *proof checker*

A proof checker offers no interaction. It receives a conjecture and a proof script as input. It then checks whether or not the proof script proves the conjecture, reporting the success or failure accordingly. The SPADE Proof Checker is an interactive theorem prover and not a dedicated proof checker. However, at the core of every interactive theorem prover should lie an inference engine performing the tasks of a proof checker. It should be straight forward to turn the SPADE Proof Checker into a traditional proof checker.

In the development of NuSPADE the SPADE Proof Checker was modified to behave more consistently allowing for the reliable and faithful execution of discovered proof plans. In particular we made some tiny changes to more strictly enforce the behaviour of the existing SPADE Proof Checker switches. Further, we introduced a few new commands to the SPADE Proof Checker. In developing NuSPADE we originally tried to configure the existing SPADE Proof Checker commands. However, as the commands we required were so trivial, we found it much more straight forward to implement new commands (in a few lines of code) rather than adjust the existing commands (which cover many thousands of lines of code).

However, issues still remain in using multiple plans for different conclusions from the same VC. Further, the SPADE Proof Checker must be manually started and pointed at the relevant proof script. To overcome such issues the SPADE Proof Checker may accept arguments off the form:

```
checker -proof-check directory file vc-number conc-number proof-script result-file
```

Where result-file will contain one of the single predicates depending on the success of the proof checking:

```
check(success).
check(failure).
```

[2] This could be avoided by having the SPADE Helper Verifier simply generate a new VC file: HLP. However, this would not be detected by POGS and would generally confuse the existing SPARK proof style. A seamless extension of the SPADE Simplifier is a reasonable compromise.

4 Finalising the SPADE Helper

4.1 Architecture

```
----------    +++++    ------------------    +++++    +++++    +++++    +++++
-Examiner->>>>+VCG+>>>>-SPADE Simplifier->>>>+SIV+    +FDL+    +RLS+    +RUL+
----------    +++++    ------------------    +++++    +++++    +++++    +++++
                V                              V        V        V        V
                V                              V        V        V        V
               --------------                  V        V        V        V
               -HelperParser-<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
               --------------
                      V
                      V
           ++++++++++++++++++++++
           +PreParsedProblemFile+
           ++++++++++++++++++++++
                      V
                      V
                   -------->>>>++++++++++++++++
                   -Helper-    +HelperDataFile+
                   --------<<<<++++++++++++++++
                                      V
                                      V
                               ---------------->>>>---------------
                               -HelperVerifier-    -SPADE Checker-
                               ----------------<<<<---------------
                                      V
                                      V
                                    +++++
                                    +SIV+
                                    +++++
```

5 Executive summary

The SPARK tools will be extended with the SPADE Helper: a tool designed to automatically discharge VCs not proved by the SPADE Simplifier. The SPADE Helper will exploit the ideas and technologies of the NuSPADE project. The SPADE Helper will be supported by two additional systems:

- **The SPADE Helper Parser** - Prepares verification problems for analysis by the SPADE Helper.
- **The SPADE Helper Verifier** - Invokes the SPADE Proof Checker to check the SPADE Helper results and generate the refined proof problem files accordingly.

5.1 Changes to SPADE Simplifier

- **Consistent numbering** - Modify the SPADE Simplifier so that the conclusion numbers in the SIV files refer to the same conclusions in the VCG files. This feature should be accessed by providing a suitable switch.

5.2 Changes to SPADE Proof Checker

- **Small, significant, changes** - Make some small but significant changes to the SPADE Proof Checker so that it more faithfully implements existing switches to switch off automated simplification procedures.
- **Additional constrained tactic commands** - Add new constrained commands to the SPADE Proof Checker for the purpose of proof checking.
- **Proof checker mode** - Add a new proof checker mode to the SPADE Proof Checker allowing the automated checking of a particular conclusion of a particular VC of a particular subprogram, storing the result in a provided file.