Hum[*] 7

Bill J. Ellis
February 28, 2005

Hey, what does the SPADEase-Parser look like, anyway?

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

In Hum 2 an architecture was proposed that separated the key components of SPADEase into the SPADEase-Parser, SPADEase-Verifier and SPADEase-Planner[1]. The behaviour of the SPADEase-Parser was thought to be quite intuitive, loading all of the information required for proof planning. This neatly corresponded to the project plan of Hum 1 (in particular, Work Package 2, Task 1, 'Import Verification Conditions'). However, on reflection, the SPADEase-Parser serves a higher purpose, acting as the interface between the external world and the SPADEase-Planner. Essentially, the SPADEase-Parser draws together all of the information required for proof planning into a single, easily accessible, location for direct and immediate access by the SPADEase-Planner. The SPADEase-Parser describes the problem ; SPADEase-Planner solves it ; and SPADEase-Verifier checks the solution.

As a consequence of this observations, the required behaviour of the SPADEase-Parser is no longer obvious. The SPADEase-Parser is also concerned with additional tasks originally expected to be considered as part of the planner (in particular, the 'Wave rule (and regular rule) management' and 'Method and critic loading' off Work Package 2, Task 2, 'Data management enhancements'). This note aims to describe the requirements and present a rough design for the SPADEase-Parser.

2 Observations

2.1 Lots of information

The following information will be held in external files and accessed by the SPADEase-Parser:

- **VCG** - The original verification conditions outputted by the SPARK Examiner.
- **SIV** - The simplified verification conditions outputted by the SPADE Simplifier.
- **FDL** - The Functional Description Language outputted by the SPARK Examiner alongside the VCG.
- **RLS** - The rule files outputted by the SPARK Examiner alongside the VCG.
- **RUL (built-in)** - The built-in rule files automatically available when using the SPADE Proof Checker.
- **RUL (user)** - Additional rule files created by the user for use within the SPADE Proof Checker.
- **WAVE RULES** - Every rule has a number (possibly zero) of wave rules. These wave rules can be calculated from the original rule or (for performance reasons) stored in a suitable file for quick retrieval.
- **METHODS** - The key components controlling the behaviour of the proof planer.
- **CRITICS** - Components, associated with methods, also controlling the behaviour of the proof planer.
- **WATERFALL** - The order in which methods are explored is implicitly encoded in the order in which methods are loaded. This order may instead be made explicit and stored in a suitable file.

2.2 Do not cache the built-in rules

The built-in rules are essentially static. Thus these rules may be treated specially to avoid their repeated processing as each subprogram's VCs are tackled. However, this special treatment would dilute the elegant concept of describing the problem in a single location. Further, implementing this special treatment will require additional programming. Finally, the advantage of this special treatment is to avoid the *loading* of information not the *processing* of information. Experience with NuSPADE suggests this loading phase could be comfortably achieved in under a second. Thus, although there is an obvious efficiency saving in dividing the problem, the limited gains and the cost of implementation mean it is not worth while pursuing.

[1] The existence of a distinct planner component was implicit in Hum 2 and was called SPADEase. This use of SPADEase to refer to both the system and the planner component is confusing. Thus, instead, SPADEase will be only be used to refer to the whole system and the planner component will be called SPADEase-Planner.

2.3 Do cache the wave rules

As the SPADEase-Parser formulates the problem, it should take control of caching the wave rules. Where a new rule is encountered its wave rules should be generated and stored. Where a previously seen rule is encountered its wave rules should exist within the SPADEase-Parser database, ready for immediate extraction and use.

Note this architecture means that the SPADEase-Parser becomes aware of information processing features. It requires access to the wave rule generator. This code exists within NuSPADE in a fairly self contained form and thus it should be possible to extract this for the SPADEase-Parser. However, the need for this does tend to suggest the SPADEase-Parser is doing too much, and that it should be divided into an information parser and information processor. It is thought that this division will lead to different problems, in particular having to integrate separate systems, that are best avoided. The SPADEase-Parser gets the information into the form for loading into the SPADEase-Planner. This process can reasonably involve more than the trivial parsing seen for other file formats.

2.4 Qualifiers

The qualifiers will provided the location of every data source. In particular, no information will be provided through environment variables[2]:

```
parser <directory containing subprograms VCs>
       <the base name of the VCG file>
       <built-in rule directory>
       <wave-rule cache directory>
       <planner component directory>
```

2.5 NuSPADE is not neat and tidy

The existing NuSPADE parsing system is distributed across a collection of programs. Some of these programs are built around old offshoots from NuSPADE related programs. These programs spend effort communicating with NuSPADE in its language, rather than modifying NuSPADE for a more convenient interface.

The SPADEase-Parser should not accommodate for the complexities of the NuSPADE input. NuSPADE may be modified to accept more natural loading of information, or an extension may be created to convert from the natural presentation into that for NuSPADE. Either way, the SPADEase-Parser will not get involved in this task, and instead focus on presenting the information in a natural and flexible form.

This effectively postpones the difficult decision of how to modify NuSPADE to create SPADEase-Planner. It is expected that NuSPADE will be incrementally refined to accept information directly from SPADEase-Parser. Thus, once the SPADEase-Parser is in operation, the route to refine NuSPADE should be clearer[3].

2.6 Missing arrow?

In Hum 2 there is not an arrow leading from the output of the SPADEase-Parser to the SPADEase-Verifier. Where the SPADEase-Planner fails to find a plan the SPADEase-Verifier should output the simplified form of VCs, retrieved and stored by the SPADEase-Parser. Should the SPADEase-Planner propagate this information for retrieval by the SPADEase-Verifier? If not, there should be an arrow from output of the SPADEase-Parser to the SPADEase-Verifier.

Note: Perhaps a single database, rather than one for the SPADEase-Parser and SPADEase-Planner would be much more straight forward? Especially since the SPADEase-Planner does, essentially, lay additional information on to the SPADEase-Parser information (e.g. plans, laid on top of the description of VCs)?[4]

3 Implementation

3.1 Continue to use pages

The pages mechanism as seen in PropGen will be repeated for the SPADEase-Parser. A page is a SICstus Prolog module containing asserted predicates and additional predicates to access and modify these asserted predicates.

[2]SICstus is perfectly capable of dealing with environment variables. I just don't like them very much. All the parameters should be visible and provided via the command line.

[3]Some tentative exploration of the deep inner workings of NuSPADE reveals that NuSPADE, internally, makes a fairly straight forward use of the Prolog database. However, this is shrouded with a fancy front end. Bypassing this front end should be straight forward, allowing for a more compact NuSPADE that directly accesses the information provided by the SPADEase-Parser.

[4]In which case the arrows are correct, but the boxes labelled `PreParsedProblemFile` and `HelperDataFile` are one and the same...

The pages serve as an accessible description of grouped data while also implementing an interface to this grouped data. A page will be created for each related group of data. However, the eventual output will be a single page for more straight forward access by NuSPADE.

3.2 Rough plan

- **VCG** and **SIV -** A parameterised page will be created to hold these two different collections of VCs. A further page will be used to describe the output. The contents of the output page will be found by processing the input VCG and SIV pages.
- **FDL -** A page will be created to hold the FDL, the same page can be used directly for the output.
- **RLS** and **RUL** and **RUL (built-in)** and **RUL (user) -** These are more complicated to process. Every rule can be described as a theorem, recording its provenance, and held in a suitable page. These rules can then be processed (pruning awkward rules, stripping quantifiers, generating rewrite rules, retrieving or generating wave rules in cache) and stored in a rules page.
- **METHODS** and **CRITICS** and **WATERFALL -** These are static data held and processed as Prolog predicates. These can be extracted on to a page and the same page can be used directly for the output.

3.3 Modified project plan

Crossing and ticking parts from the original plan that apply to SPADEase-Parser gives:

- **Import verification conditions (2 weeks)**
  - **Verification conditions** $\surd$
  - **Functional Description Language (FDL)** $\surd$
  - **Rule files** $\surd$
- **Data management enhancements (4 weeks)**
  - **Eliminate needs file** $\times$
  - **Wave rule (and regular rule) management** $\surd$
  - **Method and critic loading** $\surd$
  - **Core planner** $\times$

Thus four weeks seems realistic. It is noted that completing the **Core planner** task in two weeks (to stay on schedule) appears plausible but ambitious. Thus, shaving time off the SPADEase-Parser is a high priority.

4 Executive summary

The SPADEase-Parser will do a little more than simply parse. Rather, it is the interface between the external world and the SPADEase-Planner. Although the details outlined above are quite sketchy, they do capture the boundary of the system and provide a good framework for implementation.