CS-Note* 17

Bill J. Ellis
August 22, 2002

Requirements and design of System: FRiENDLY

Contents

1 Introduction

The EXAMINER generates verification conditions (VCs) in VCG files. These contain full structural information, right down to the names of each constant. This is the ideal form for VCs input to NUSPADE. As noted in CS-note 13 the SIMPLIFIER accepts as input VCG files, trying to prove conclusions and modify structures to make them more suitable for human reading. The result is outputted in SIV files. While the SIMPLIFIER provides assistance in proving conclusions, it also modifies the ideal full structural information of the EXAMINER VCs.

It is considered necessary to exploit the power of the SIMPLIFIER without hindering the operation of NUSPADE. This is the role of FRiENDLY[1].

2 Requirements: A NUSPADE friendly-SIMPLIFIER

2.1 Desired Behaviour Is All-Or-Nothing Simplification

The desired behaviour for a NUSPADE friendly-SIMPLIFIER is outlined in CS-note 13. Essentially this involves an all-or-nothing simplification process, where either hypothesis and conclusions of VCs are fully removed or left completely untouched.

2.2 Beyond All-Or-Nothing Behaviour

An all-or-nothing simplification process sounds fairly principled. However, it may not fully exploit the behaviour of the SIMPLIFIER in practise. There are some further issues that need to be considered:

- **Eliminated hypothesis -** Hypothesis can be eliminated by the SIMPLIFIER using a number of different techniques. For example, if a hypothesis is reduced to true it is removed. There may be an advantage in keeping some of these hypothesis for NUSPADE.
- **Proved conclusions as new hypothesis -** Conclusions can be proved by the SIMPLIFIER. Once proved, a conclusion could be considered a hypothesis for the remaining conclusions. It is conceivable that these extra hypothesis may be of use to NUSPADE.
- **Reaching false -** The SIMPLIFIER might reduce a hypothesis or conclusion to false. These can not be removed, as they have not been proved. However, the information is still valuable to NUSPADE.

*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]The name stems from original ideas of constructing a NUSPADE friendly SIMPLIFIER. The lower case "i" is cosmetic, to make the upper case FRIENDLY look a little more friendly..

- **Restructured and proved -** Hypothesis and conclusions are sometimes restructured into multiple parts. For example, breaking a conjunction into its conjuncts. Each of the restructured components may be broken down yet further until eventually the original conjunction has been reduced to a number of false or true parts. As noted above this information is likely to be useful for NuSPADE. However, extracting this information may be a little more complex.

- **Restructured and half proved -** In restructuring hypothesis and conclusions into multiple parts it may be that not all parts are successfully reduced to false or true. In these cases a lot of SIMPLIFIER effort may be lost by returning to the original structure instead of the one remaining part of this structure.

- **Unreachable code is masked -** Unreachable code translates to contradictory hypothesis. This allows for a false hypothesis to be derived, leading to the automatic proof of the VC. This is the ideal behaviour of a theorem prover. However, knowing the nature of VCs, proving such theorems in this manner indicates a potential issue. As noted in CS-note 13 it may be worthwhile to consider non-theorems in NuSPADE. Therefore it may be necessary to deal with these in the SIMPLIFIER.

2.3 Short Term Proposal

Deciding how to deal with these situations is a long term issue for NuSPADE. In the short term a rough guide is proposed. Analysing the SIV files generated from this initial version is likely to lead to future changes. The general aims of the initial version are summarised below:

- **Remove hypothesis or conclusions reduced to true -** Remove all hypothesis or conclusions that are ultimately reduced to true.

- **Remove eliminated hypothesis or conclusions -** Remove all hypothesis or conclusions that are eliminated. This is a separate notion to truth. For example, a conclusion C1 might be eliminated as it is the same as C2, however this does not mean that C2 is true.

- **Record false hypothesis or conclusions -** Replace all hypothesis or conclusions that are reduced to false, with false. Where contradictory hypothesis are found add a new false hypothesis, rather than discharging the VC.

- **Add any new hypothesis -** There are several points where new hypothesis might be added. In particular proved conclusions might be added as hypothesis. These will be added to aid evaluation. If there is little merit in adding new hypothesis this can be removed later.

- **Record final actions as comments -** As noted above the exact behaviour of FRiENDLY will probably be refined over time. Recording the final actions of the SIMPLIFIER as comments, should be helpful in guiding these changes.

3 Design 1: Inside the SIMPLIFIER

3.1 System Interactions

The friendly-SIMPLIFIER can be implemented inside the SIMPLIFIER. This will exist as extended functionality of the SIMPLIFIER, a friendly-SIMPLIFIER-mode. Access to the friendly-SIMPLIFIER-mode will be controlled through an additional SIMPLIFIER flag. Where using NuSPADE it would normal to always have this flag set.

From this view the system interactions in Figure 1 are created. Note how the scope of NuSPADE includes the friendly-SIMPLIFIER, as proper use of NuSPADE requires use of this component.

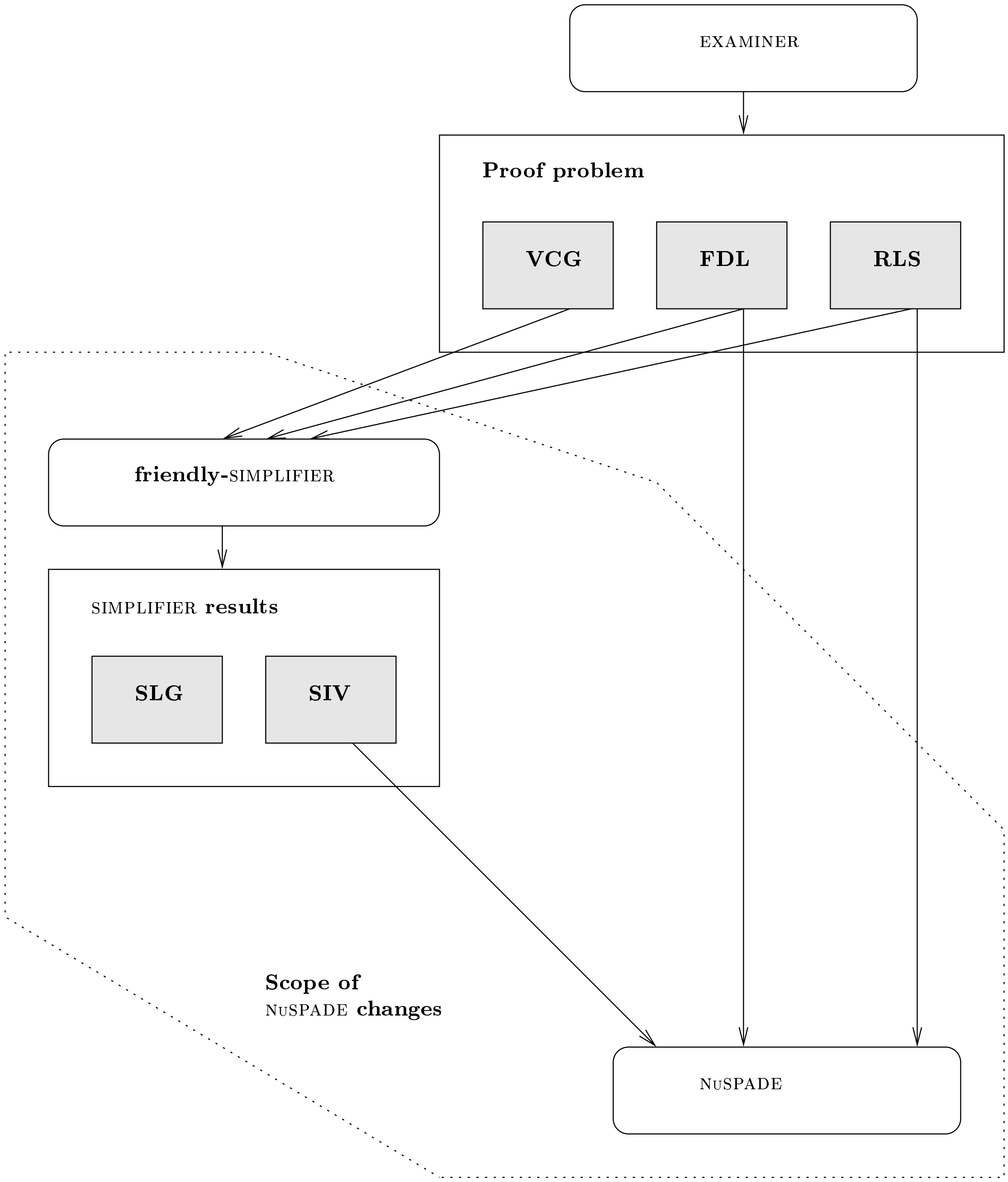

Figure 1: Inside the SIMPLIFIER

3.2 Avoid Modifying the SIMPLIFIER

Initially it was considered that modifying the SIMPLIFIER as outlined above was the ideal solution[2]. However, there are a number of reasons why this may not be ideal for NuSPADE:

- **The friendly-SIMPLIFIER is subject to Praxis and NuSPADE project changes -** The changes proposed to SPADE to implement a restrictive mode (In CS-note 8) were well understood. Here the situation is different. The desired behaviour of the friendly-SIMPLIFIER may change as understanding of the verification problems change. Further, Praxis continue to improve the SIMPLIFIER at a fairly rapid pace. It is in the interests of the NuSPADE project to work with the most recent version of the SIMPLIFIER, to genuinely identify its limits, and position NuSPADE accordingly. Keeping our research changes to the SIMPLIFIER in synch with the ongoing Praxis changes may be an ongoing and time consuming problem.

- **Changes are not naturally inside the SIMPLIFIER -** Although the SIMPLIFIER has not been subjected to much scrutiny, the general approach appears to be:

  1. **Load VC -** Get next VC and assert as predicates `PRED`[3].
  2. **Process VC -** Simplify this VC making changes to and possibly extending `PRED`. During the process assert a number of `log_fact/2` predicates to record the actions taken and their intermediate results.
  3. **Write Result -** Write `PRED` to an SIV file.
  4. **Write Log -** Write out the actions taken along with their intermediate results as an SLG log file.

  To implement the friendly-SIMPLIFIER-mode it will be necessary to keep the original VC as `PRED`, to present its parts that were ultimately not suitably simplified. The final result of `PRED` (The contents of the SIV file) is not useful as it can not be related to the initial `PRED`. Instead the many `log_fact/2` predicates will need to be analysed. The SIMPLIFIER only processes these predicates in generating the SLG log file.

  It will be necessary to make use of the initial `PRED` and analyse the `log_fact/2` predicates, recreating a new final `PRED` accordingly. The friendly-SIMPLIFIER-mode operates on the SIMPLIFIER input and eventual output, but nothing in between. Building the friendly-SIMPLIFIER-mode will involve creating a new program inside the SIMPLIFIER. The only benefits are reusing a VCG parser and having direct access to the `log_fact/2` predicates.

- **Exploit existing flags restricts SIMPLIFIER power -** The SIMPLIFIER already provides a large number of flags that can be tweaked by the user. However, the SIMPLIFIER is a highly tuned program and is generally best used with its default settings. It is certainly possible that changing these flags could restrict the power of the SIMPLIFIER. While tweaking these flags might aid the implementation of the friendly-SIMPLIFIER mode the reduction in SIMPLIFIER power is not acceptable[4].

- **Friendly-SIMPLIFIER-mode is part of NuSPADE -** Key to the complete behaviour of NuSPADE will be the use of the friendly-SIMPLIFIER-mode. Users of NuSPADE need to have access to a suitably modified version of the SIMPLIFIER. Although not a large problem, this may result in a confusing number of different SIMPLIFIER versions. It would be more straight forward if NuSPADE could appear as a single component.

- **Not the Praxis way of doing things -** After generating all of the VCs for a program, a small program SPARKSIMP can be called. This searches for each VCG file, passing it to the SIMPLIFIER. The end result is a directory structure containing all of the original VCG files and simplified SIV files. The POGS tool can then be used to summarise the VCs that remain to be fully discharged. Such a sequence of steps results in the elegant Praxis-triple:

  ```
  spark -exp_checks program.ada
  sparksimp
  pogs
  ```

  To perform this for NuSPADE it would be necessary to use the friendly-SIMPLIFIER-mode. If the directory tree has already been created using a normal SIMPLIFIER then NuSPADE may struggle[5]. Proper use of NuSPADE depends on proper use of the SIMPLIFIER. Again, it would be more straight forward if NuSPADE could appear as a single component, able to start work directly inside a regular Praxis directory structure.

[2] Although only hinted at CS-note 13, this was the intended plan.
[3] To be specific: `hyp/3`, `conc/3`, `found_contradiction/0`, `join_hyp/4`, `subst_hyp/3`, `could_infer/2` and `could_not_infer/1`.
[4] Taken to the extreme, this might lead to the SIMPLIFIER achieving little more than renaming the VCG file as an SIV file.
[5] It should be quite possible to work with the normal SIMPLIFIER results. However, NuSPADE is likely to struggle with these.

4 Design 2: Outside the SIMPLIFIER

4.1 System Interactions

Instead of modifying the SIMPLIFIER to provide a NuSPADE friendly mode, an external program could be constructed that performs the actions of the friendly-SIMPLIFIER mode. This is system FRiENDLY. As noted in section 3, the implementation of FRiENDLY does not require access to the SIMPLIFIER internal state. It involves comparing the initial problem (Held in a VCG file) and the actions taken during simplification (As default, recorded in a SLG file).

From this view the system interactions in Figure 2 are created. Note how the scope of NuSPADE no longer includes the SIMPLIFIER. By extracting the friendly-SIMPLIFIER mode as an external system any version of the SIMPLIFIER can be used alongside NuSPADE.

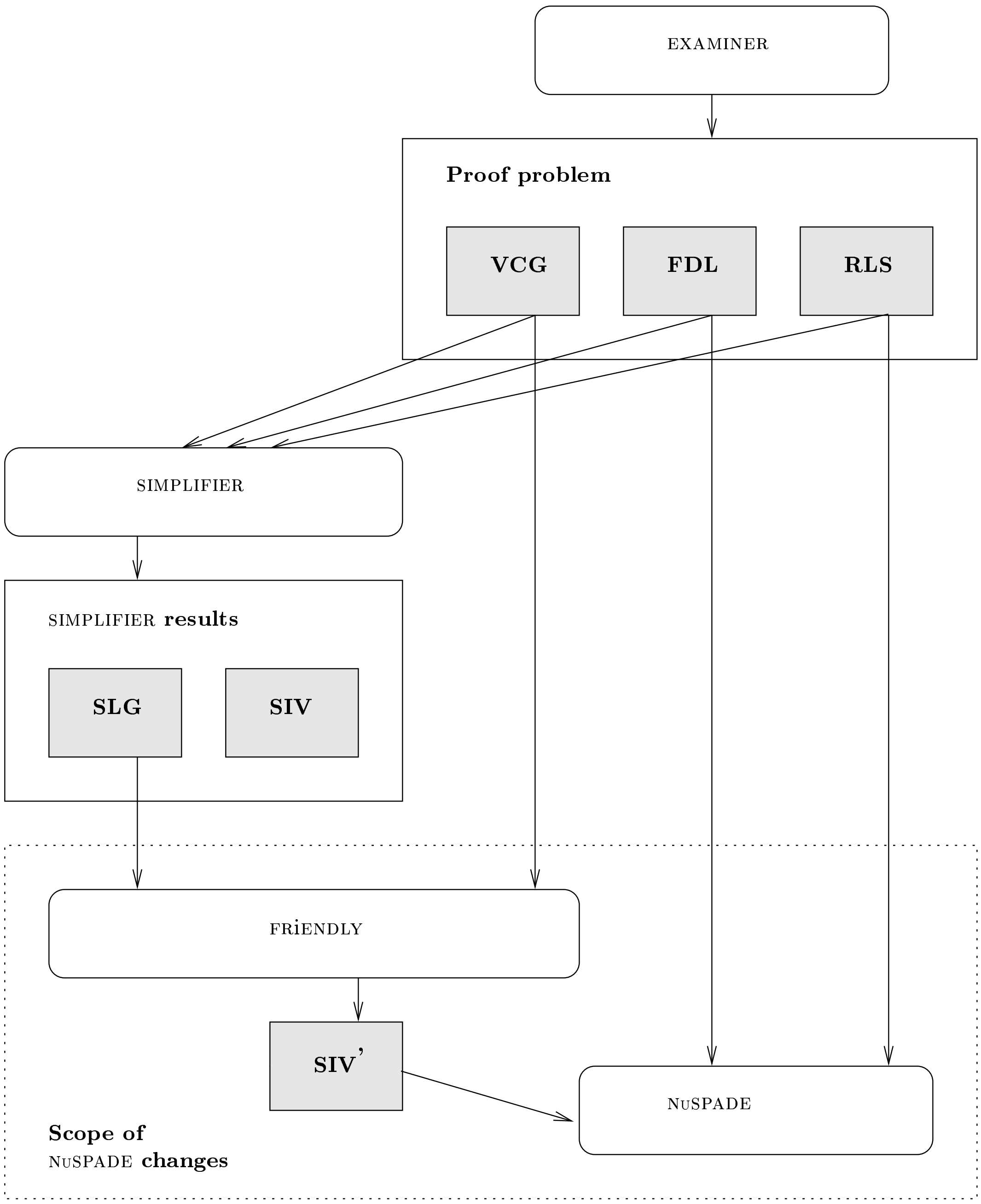

Figure 2: Outside the SIMPLIFIER

4.2 Trouble Parsing SLG File

For the reasons presented in section 3 this design is much more favourable. However, the design was followed for short while, reaching ever problematic issues in parsing the SLG file. Although it is felt that it would be possible to parse the SLG file, this is a much more difficult exercise that had originally been anticipated:

- **One entry with multiple information -** It was originally thought that each entry would translate to a single piece of information. However, this is not the case. For example, where a hypothesis is restructured it is wanted to recorded the restructuring and record if any of the new components have reached true or false. In general it is wanted to access all of the information in each entry prior to performing any analysis. This requires more sophisticated parsing than was originally considered[6].
- **Difficult to predict newlines -** The SIMPLIFIER makes use of POPLOG-PROLOG pretty printer in outputting the SLG entries. This pretty printer indeed performs clean output, inserting newlines at sensible points. Unfortunately, this means that many SLG entries are on multiple lines, requiring more sophisticated parsing to ensure robust behaviour.
- **Terms often not terminated by full stop -** It is occasionally advantageous to read in terms from the SLG entries. Unfortunately, these terms are often not terminated by a full stop (Or any other know character), preventing direct use of the prolog parser.

[6] A lot of progress can be made in examining each line in isolation.

5 Design 3: Outside the SIMPLIFIER (With small SIMPLIFIER changes)

5.1 System Interactions

To overcome the problems in parsing the human readable SLG log the SIMPLIFIER might be modified to output the SLG log in more machine readable format. However, there is no need to focus on the SLG file. Instead the SIMPLIFIER could be modified to output a second machine readable log MLG in a format that can be directly parsed. This log would then replace the SLG log as the input to FRiENDLY.

From this view the system interactions in Figure 3 are created. Note how the scope of NuSPADE does not include the SIMPLIFIER. Although the SIMPLIFIER will be modified it is hoped that the changes will reach the distribution SIMPLIFIER. The issues raised in 3 are still overcome by putting the majority of the effort in an external system.

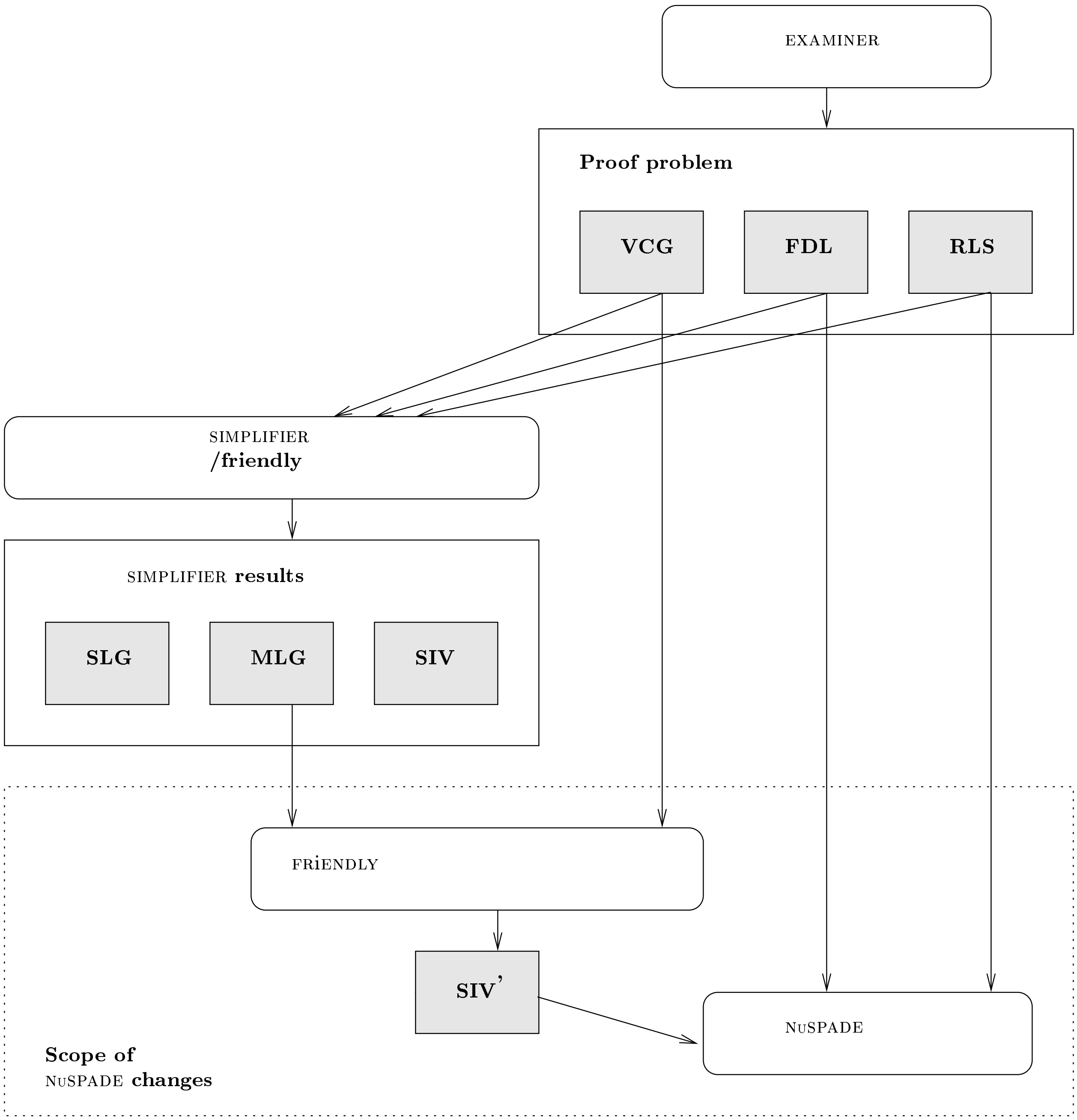

Figure 3: Outside the SIMPLIFIER (With small SIMPLIFIER changes)

5.2 Compromises

Modifying the SIMPLIFIER, even it this minimal way, is unfortunate. However, this design is seen as the best compromise. Using the guidelines from CS-note 15 it is shown that the proposed changes meet the preconditions to modifying the SPARK tools.

Precondition	Meets how
Necessary	As discussed in section 3 it is necessary to access the record of changes made by the SIMPLIFIER. As discussed in section 4 accessing the SLG log is somewhat difficult. Though it may be possible to parse the SLG log, it is not the sensible route.
Definite	The modifications to the SIMPLIFIER are very clear cut. Essentially they involve outputting the `log_fact/2` predicates in a quoted Prolog form to a file. So long as Praxis modifications to the SIMPLIFIER continue to use `log_fact/2` in the same manner, the modifications will continue to be correct across different versions of the SIMPLIFIER.
Unobtrusive	The storage of a machine readable log is a natural extension to the SIMPLIFIER which already outputs a human readable log. As the machine readable log makes use of the data stored for the human readable log there is no large additional processing to be conducted.

6 Conclusion

The behaviour of a SIMPLIFIER geared for NuSPADE has been elaborated. From this consideration requirements for a new system FRiENDLY have been established. Through fairly rapid prototyping[7] three different designs have been explored for this system. The chosen design balances the desire to minimise the changes made to the SIMPLIFIER while not undertaking major work to avoid doing so.

[7]Read as: Getting it wrong, really quickly.