Bill J. Ellis
May 9, 2002

Framework for NUSPADE Development

1 Introduction

A number of ideas, suggestions and proposals surrounding the development of NUSPADE are presented.

2 Operational Environment

2.1 Manage SIMPLIFIER Simplification

The EXAMINER generates the initial problem. This can be passed to the SIMPLIFIER to produce a refined problem. The refined problem is simplified by eliminating, restructuring, or otherwise changing the initial problem. While this can be clearer to read, it can also mutate goal structure that would otherwise be helpful in guiding an automated search for proof. Thus, where the SIMPLIFIER is used there is a need to ensure it does not adversely affect NUSPADE.

3 Features Required

3.1 Demonstrate Soundness

Proof planning takes a higher level view of constructing a proof. Operating at this level it is not possible to guarantee that the proof plan translates into a sound proof of the current problem. For this reason it is necessary to demonstrate the soundness of a proof plan by actually performing the proof in a theorem prover.

The CLAM proof planner relies on OYSTER to demonstrate the soundness of its proof plans. The ΩMEGA proof planner [1] relies on an independent PROOF-CHECKER component[1]. And NUSPADE relies on SPADE. It is therefore essential that every NUSPADE proof plan is converted into a command log that can be executed by SPADE. Only through successfully executing this command log inside SPADE will the NUSPADE proof plan be shown to be sound.

3.2 Provide Setting Mechanism

As a research system NUSPADE is certainly going to be tweaked and reworked to incorporate and evaluate emerging ideas. To help cater for these changes, it is important that NUSPADE has a reasonably sophisticated settings facility.

Such a facility should allow toggling the use of sections of code. For example, it may be desirable to switch off proof methods so that new proof methods can be tested in isolation. It should also be possible to change the value of key system variables. For example, to change the depth used for iterative deepening, or the extent of user interaction during proof planning.

4 Practical Development Guides

4.1 Implementation Language

On all platforms SPADE is implemented in Poplog-Prolog. Thus, NUSPADE will also be developed in Poplog-Prolog. However, care will be taken to ensure that, like SPADE, NUSPADE does not become too reliant on Poplog-Prolog specific commands.

4.2 Module Mechanism

As default all Prolog predicates are loaded at the same flat level. Where programming at this level there is no scope for abstraction. Where debugging at this level, it is not clear which part of the system each predicate resides. To overcome these problems, many implementations of Prolog have adopted some form of module system. These tend to have a separate Prolog environment for each module, and allow the declaration of interface predicates that

*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] This should not be confused with the SPADE proof checker.

can be called from outside the module. Further, where modules have been implemented, the normal Prolog flat level is usually implemented as a module.

Unfortunately, as module support is not part of standard Prolog, there is not a standard implementation. Different Prologs have different notions on what constitutes a module and the syntax that should support them. Sicstus adopts a module structure of at most one depth. A similar approach is adopted by Quintus and SWI. Poplog-Prolog supports a full nested structure of modules.

The benefits of modules are sufficient to warrant their adoption in NUSPADE. To avoid becoming reliant on Poplog-Prolog, the depth of these modules should be restricted to one. It is expected that, in practise, this constraint would never want to be broken.

4.3 Observe Module Abstraction

The module mechanism provides a supported method for partitioning software components. However, the partitioning is not strictly enforced. It is possible to access the inner details of modules using appropriate hierarchical syntax.

Such misuse of the module mechanism should be avoided. This should mean that a modules data structures are only accessed through defined interface predicates. Note that this does not prevent the existence of global data. Suitable global data modules can be declared for this purpose.

4.4 A Module for SPADE

SPADE may be packaged as a single, large, module. This would allow using the module facility to explicitly document which SPADE predicates are accessed by NUSPADE. Further, it would ensure that SPADE and NUSPADE are truly separated in the Prolog environment.

4.5 Support Debugging

All versions of Prolog provides some form of debugging facilities. This are often powerful debugging tools, and should be sufficient for debugging NUSPADE during development.

It is worth noting that consideration had been given for adopting a NUSPADE specific debugging framework. Although this may have brought some benefits, the implementation would be costly and, given the very good Prolog debugger, somewhat unnecessary.

4.6 Defensive Programming

During the execution of NUSPADE there will be many points where anticipated yet incorrect behaviour can occur. For example passing a string of letters to a predicate expecting a string of numbers. At these points there is the option of assuming the incorrect behaviour will never occur, or to test for the behaviour and act accordingly. The latter is preferred as, repeated often, such defensive programming can lead to a stronger system. The following special actions are proposed to cater for incorrect behaviour:

- **WARNING -** Incorrect behaviour detected. However, potentially through some counter measures, a valid action can be performed.
- **ERROR -** Incorrect behaviour detected. As a result of this behaviour a valid action can not be performed. The system should exit.

References

[1] C. Benzmüller, L. Cheikhrouhou, D Fehrer, A. Fiedler, X. Huang, M. Kerber, K. Kohlhase, A Meirer, E. Melis, W. Schaarschmidt, J. Siekmann, and V. Sorge. Ωmega: Towards a mathematical assistant. pages 252–255. Springer-Verlag, 1997.