CS-Note* 86

Bill J. Ellis
August 18, 2005

# SPARK Plugs

## Contents



# 1 Introduction

This short note plugs useful research proposals that would help advance ongoing research into the SPARK Approach.

# 2 Scalable proof planning

## 2.1 Overview

Proof planning has traditionally focused on small but significant proof efforts. Consequently, research emphasis tends to be placed on the strength of automated reasoning techniques. However, the practical dimension, in selecting an architecture and implementation of proof planning, remains important. In particular, in the domain of program reasoning, the proof planning framework can expect to encounter thousands of individual conjectures, some of significant size.

Note that this argument is not surprising. Proof planning conducts, essentially, an exhaustive search of a state space trying to achieve a particular goal. Other systems conducting exhaustive search, such as model checkers and refutation based provers, have invested significant effort in the practical task of streamlining implementation. Although compared to these techniques proof planning conducts a narrow search, this can still be a significant overhead.

## 2.2 Detail

Proof planning systems can be separated into an *inference engine* and a *database*. During a proof, the inference engine is responsible for executing methods (automated reasoning heuristics) and controlling the search. The database stores all of the information associated with the proof, including all of the rules, annotated rules, hypothesis, and goal tree. Existing proof planning systems focus on the inference system, implementing the database in the most convenient manner.

- **Clam -** The Clam inference engine is written in Prolog. Prolog is particularly well suited for developing a proof planner inference engine. The Clam database builds on Prolog. While Prolog does support database access, it can be slow and does not naturally scale up to larger problems.
- **$\lambda$Clam -** All aspects of $\lambda$Clam build directly upon $\lambda$Prolog. Development on $\lambda$Clam has ceased in favour of IsaPlanner.

*Notes in this series are for $\epsilon$-baked ideas, for $1 \geq \epsilon \geq 0$. Only exceptionally should they be cited or distributed outwith the NuSPADE project (EPSRC GR/R24081).

- **IsaPlanner -** IsaPlanner is the cutting edge proof planner at Edinburgh University. IsaPlanner is an extension to the interactive theorem prover Isabelle. While the implementation of IsaPlanner is ongoing, it seems likely that it will use the internal Isabelle programming language Isar. Certainly IsaPlanner is a strong proof planner, however the emphasis remains on proof plan development and debugging rather than efficient execution of proof plans.

- **SPADEase -** SPADEase is a refinement to NuSPADE, which is itself an extension to Clam. SPADEase addresses some of the weaknesses in Clam, for example by having a slightly more efficient data storage policy. However, SPADEase continues to rely upon the Prolog database.

### 2.3 Project

Create an industrial strength, scalable proof planner, by integrating SPADEase with an external database system. As the fundamental objective for this is to improve the strength and performance of SPADEase other efficiency considerations may be explored. Interesting challenges that might be considered include:

- Selecting efficient data structures for proof planning. For example, proof planning might be more efficiently represented as a graph rather than a tree (as the same goal state may be generated more than once). Further, an efficient, general, approach to annotation would be valuable.

- Developing cache strategies (loading frequently accessed information into the Prolog database).

- Exploiting and refining the currently limited support for sub-planning within SPADEase. (This involves the separate planning of selected conjectures to avoid re-planning the same conjecture more than once. This is in principle similar to the separate proof of lemmas - without the usual intent that the lemmas would be useful for other problems. The decomposition into sub-goals is for efficiency alone.)

## 3 Sparking IsaPlanner

### 3.1 Overview

IsaPlanner is the cutting edge proof planner at Edinburgh University. SPADEase is a SPARK based proof planner, including tools for processing SPARK verification conditions. Currently, there is no means to export SPARK problems into IsaPlanner.

There are many communities using IsaPlanner to explore and refine proof plans. Adapting IsaPlanner to accept SPARK problems would allow us to transport our interesting verification problems to these communities. Further, given the strength of the IsaPlanner tool, it could be a good platform for developing new SPARK heuristics.

### 3.2 Project

Use IsaPlanner to proof plan SPARK verification problems. This will involve understanding the behaviour of IsaPlanner and modifying the existing SPADEase translation programs accordingly. Once IsaPlanner is able to plan SPARK problems, experiments can be conducted on the strength on IsaPlanner's methods. Based on these experiments new methods can be introduced. Interesting challenges that might be considered include:

- Integrating IsaPlanner within the Spark Approach, for example by generating and maintaining SPARK proof review files to reflect the proof progress within the sparked IsaPlanner.

- Exporting discovered IsaPlanner proof plans for execution in the SPADE proof checker.

- Isabelle, and hence IsaPlanner, support interactive theorem proving. SPARK problems tend to suggest a different style for visualising and interacting with an interactive proof. IsaPlanner may be adapted as an improvement over the SPADE proof checker.

## 4 Proof management

### 4.1 Overview

While the SPARK Approach includes a collection of powerful individual tools, there is no top level management system integrating these tools. While there are plans for a SPARK aware integrated development environment, it is unlikely that this will address the particular problems of proof management.

Proof management involves recording and maintaining proofs and the associations between proof objects. Through effective proof management it should be possible to automatically identify all verification conditions that are dependent on individual lemmas or assumptions.

Unfortunately, the SPARK tools offer limited support in extracting the key details of completed proofs. The Simplifier does report its actions, however it is difficult to identify specific rules used. The Checker has slightly more robust rule reporting. There is not an automated mechanism for proving rules, making it difficult to manage the proofs of lemmas.

In short, SPARK would indeed benefit from an overarching proof management system. However, significant work is required on the existing SPARK tools to reach the stage where the important data can be retrieved. Only then can the proof management task begin.

## 4.2 Project

Proof management would make for an interesting project. However, this requires access to connected proof data structures. The overhead in extracting these data-structures from SPARK is likely to dominate the project. Unfortunately, this task is not particularly interesting, nor valuable as the natural evolution of the SPARK tools is likely to address these problems eventually.