Bill J. Ellis
March 1, 2005

And the SPADEase-Planner

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

Hum 7 explored the nature of the SPADEase-Parser. This note continues this theme, for the SPADEase-Planner.

2 A big question

There are a few ways to explore the development of the SPADEase-Planner:

- **NuSPADE** - Directly employ NuSPADE as has been used for earlier projects.
- **Core NuSPADE planner** - Extract and employ the core planner aspect in NuSPADE. This would involve replacing the existing NuSPADE infrastructure while leaving the core planning essentially intact.
- **Mini-Planner** - The Mini-Planner[1] is the planner component seen inside PropGen. This can be extended and employed as the planner.

Some observations are made in deciding which path to pursue.

2.1 NuSPADE needs some refining

The NuSPADE planner has been shown to work on the kinds of problems expected to be seen by SPADEase-Planner. However, as indicated in Hum 1 there are various aspects of NuSPADE that require addressing. Further, as witnessed in Hum 8, NuSPADE can be complicated and difficult to extend. There is a strong argument that NuSPADE should be accessed as a single, unchanged, library component and new components should be created for new functionality.

2.2 Planner implementations

Both NuSPADE and Mini-Planner lack, or barley support, key functionality for a proof planner. Supporting the critic mechanism requires recording the goal structure. Maintaining a goal structure within a Prolog search causes problems for meta-variables. In particular, special care needs to be taken to instantiate meta-variables throughout the goal tree once a concrete value is discovered. Also, there are large problems in behaving correctly where meta-variables are present at splits in the goal tree.

2.3 Performance issues

The Mini-Planner, like NuSPADE, records a goal structure suitable for supporting the critics mechanism. However, unlike NuSPADE, the Mini-Planner makes little use of regular Prolog search. This tends to make the Mini-Planner a little more straight foreword to understand and develop, however this also tends to make the Mini-Planner slower in execution.

2.4 Don't disband the explicit goal tree

The critics of NuSPADE are explored immediately after a failed method is detected. There is not actually a need to store the whole goal tree for such critics. However, the explicit goal tree is especially valuable in debugging proof plans. Further, it is conceivable that new critics will require access to information about the current plan beyond the details at the current goal.

2.5 The planner is small

The core planner is a surprisingly small component. The vast majority of development time is spent refining methods and their interactions rather than the core planner. Thus, it is prudent to spend a little additional effort to produce a planner that eases the more lengthy method development process.

3 A little answer

Package NuSPADE as a library. Access this library from a refined version of the Mini-Planner. This leaves some technicalities to be considered.

[1] It is mini because, unlike NuSPADE, it is only a planner. It does not have its own meta-language, instead choosing to reuse the various predicates embedded within NuSPADE.

3.1 The needs file

The needs file curtails the visibility of rewrite rules during planning. Fewer rules leads to a smaller search space with the disadvantage of possibly missing a proof. In NuSPADE additional rules leads to a significant increase in processing time as wave-rules are generated on the fly. This problem is eliminated in SPADEase through a new wave-rule cache policy.

- Rippling is a well constrained proof technique. The addition of wave-rules should not impact adversely on the behaviour of the ripple methods.
- The trans method searches for rules that deconstruct the current goal. This is achieved by looking for rules of a particular form, rather than a collection of named deconstruction rules. Additional rules may lead to additional search space, however this searching will be controlled and useful.
- The simplify method, is constrained by a goal reducing measure. Additional rules may allow for greater scope in simplification, but this simplification is expected to be controlled and useful.

It is slightly surprising to note that, in theory at least, the availability of rules should have a minimal impact on the performance of the planner. The needs file exists primarily to describe the context required for proving goals and manage the costly repeated regeneration of wave-rules seen in NuSPADE.

3.2 Controled planning for a more constrained search

It seems the primary concern in the performance of the planner depends on the behaviour of the methods rather than the existence of rules. Method invocation is controlled in NuSPADE via Prolog backtracking over the asserted methods. This is not always ideal. Often a method will produce a new state that is better than the previous state[2]. However, there is no mechanism in NuSPADE to prevent backtracking to the previous state and exploring a search space that, essentially, has been explored in the better state. Not being able to curtail the search space in the presence of obvious better states is is a significant weakness in NuSPADE.

The Mini-Planner invokes methods as described by a controlling script. This script replaces the Prolog backtracking model of NuSPADE with an explicit list of methods to be invoked. Operations are permitted on this list, including the removal of methods to prune superseded search spaces. In practise, this curtailing of the search space is similar to the use of the cut operator in Prolog.

Maintaining the Mini-Planner method control script will give additional leverage in composing more constrained method invocations and consequently help constrain the search space.

3.3 When to give up searching?

In practise, NuSPADE offers four end scenarios:

1. Finding a proof plan and finishing with success.
2. Finding an error, reporting the error, and finishing with failure.
3. Complete an exhaustive search and terminate with failure.
4. Get lost in a huge search space, causing the user to intervene and terminate the process.

Ideally, only the first three scenarios should be present. However, the inability to control method invocations and slightly weak preconditions on some methods means that the final scenario is a real possibility. Certainly, efforts can be made to constrain the search. However, to address the final scenario a straight forward timer interrupt might be considered[3].

3.4 Middle-out reasoning

NuSPADE supports middle-out reasoning by the explicit registering and patching of variables within proof methods. This approach is designed for sections of plans with no splits and for a single point of meta-variable introduction. This mechanism is specialised and not robust. The Mini-Planner does not support any form of middle-out reasoning.

The main problem with both NuSPADE and Mini-Planner is the creation and use of an explicit goal tree. This goal tree exists separate to the Prolog search and thus variables are not automatically unified and instantiated across plans.

[2] For example, simplification almost always produces a new state that entirely supersedes the previous state.

[3] This is a crude but effective method to ensure termiunation. Alternative stratigies involving certianly reducing measures, such as counting expanded goals, might be preferable.

3.4.1 Record the variables

Both NuSPADE and Mini-Planner holds information alongside the hypotheses and conclusions. This is a convenient location to discuss the values of meta-variables. A goal will start of the form:

```
ROOT: []: ground(Hs): ground(Conc)
```

Note it is assumed that the hypotheses will always be ground. Middle-out reasoning is only considered for the conclusion. A method may introduce some meta-variables:

```
GOAL-1: []: ground(Hs): ground(Conc)/\A/\A/\B
```

When this method is stored, the meta-variables can be identified and reduced to atoms. The ultimate goal is to translate these to a ground term.

```
GOAL-1: [vars([a,b,c]), translate([])]: ground(Hs): ground(Conc)/\a/\a/\b/\c
```

This goal is restored to its natural Prolog form by introducing meta-variables accordingly and recording the association to their atom names. Note that these Prolog variables will have no direct relationship to the above Prolog variables. They are new variables.

```
GOAL-2: [translate([]), map([a=X, b=Y,c=Z]))]: ground(Hs): ground(Conc)/\X/\X/\Y/\Z
```

The goal can be transformed by the method. The changes to the meta-variables will be propagated over into their copies in the information list.

```
GOAL-2: [translate([]), map([a=true, b=(Q/\R), c=Z]))]: ground(Hs): ground(Conc)/\true/\true/\(Q/\R)/\Z
```

Those mappings that are not ground will produce new variables.

```
GOAL-2: [vars([d,e,f]), translate([]), map([a=true, b=(d/\e), c=f]))]: ground(Hs): ground(Conc)/\true/\true/\(d/\e)/\f/\V
```

Any remaining variables in the goal will also produce new variables (this corresponds to middle-out reasoning within middle-out reasoning, something neither NuSPADE nor Mini-Planner supports. In practise, this can occur in the exception freedom plans with a double invocation of the trans method).

```
GOAL-2: [vars([d,e,f,g]), translate([]), map([a=true, b=(d/\e), c=f]))]: ground(Hs): ground(Conc)/\true/\true/\(d/\e)/\f/\g
```

The mappings are translations to the variables of the previous goal.

```
GOAL-2: [vars([d,e,f,g]), translate([a=true, b=(d/\e), c=f])]: ground(Hs): ground(Conc)/\true/\true/\(d/\e)/\f/\g
```

Restore the goal again.

```
GOAL-3: [translate([a=true, b=(d/\e), c=f]), map([d=U, e=I, f=O, g=P])]: ground(Hs): ground(Conc)/\true/\true/\(U/\I)/\O/\P
```

Suppose the methods grounds everything this time.

```
GOAL-3: [translate([a=true, b=(d/\e), c=f]), map([d=false, e=true, f=1=1, g=true])]: ground(Hs): ground(Conc)/\true/\true/\(false/\true)/\1=1/\true
```

No pending variables.

```
GOAL-3: [vars([]), translate([a=true, b=(d/\e), c=f]), map([d=false, e=true, f=1=1, g=true])]: ground(Hs): ground(Conc)/\true/\true/\(false/\true)/\1=1/\true
```

Move mappings to translations.

```
GOAL-3: [vars([]), translate([a=true, b=(d/\e), c=f, d=false, e=true, f=1=1, g=true])]: ground(Hs): ground(Conc)/\true/\true/\(false/\true)/\1=1/\true
```

Note that the final form of translate describes terminating rewrite rules (left to right) that can reduce any goal with recorded meta-variables into its eventual instantiated form. Backtracking may produce different variables and different translations, but the goals seen in reaching a proof should contain translations that directly affect the variables seen.

3.4.2 Man, that's complicated

The above seems rather complicated. However, it does not seem unnecessarily complicated. Further, the implementation should be fairly straight forward, using the various existing term manipulation predicates available as part of NuSPADE and PropGen.

3.5 Extracting plans

NuSPADE is capable of tracing backwards from a discovered plan and assembling a proof plan. This feature will need to be implemented for the Mini-Planner. The Mini-Planner does support the pruning of unfinished goals in branches leading to proof. It is thought that this code can be reused for extracting plans.

4 Executive summary

The fundamental characteristics of the SPADEase-Planner are considered. It is decided to build upon the Mini-Planner (as seen in PropGen) and exploit NuSPADE as a library of useful predicates. Key features to implement for SPADEase-Planner include the interface to SPADEase-Parser, middle-out reasoning support and the extraction of discovered plans.