Proof General / Eclipse: A Generic Interface for Interactive Proof

Daniel Winterstein[1], **David Aspinall**[1] **and Christoph Lüth**[2]
[1]University of Edinburgh, U.K. and [2]Universität Bremen, Germany

1 Introduction

Inspite of the considerable achievements of the formal proof programme, take-up of these systems by mathematicians and programmers remains poor. At least one reason for this is the lack of good development tools. The *Proof General* (PG) project is an ongoing attempt to redress this issue. Here we present PG/ECLIPSE, an interface/development environment that marks a new phase in the project.

2 The PG/ECLIPSE Interface

PG/ECLIPSE provides a rich range of features, many of which will be familiar to users of modern programming IDEs.

- Proof scripts (and projects) are presented to the user via a specialised text-editor and associated viewers (e.g. the document outline view) which support various actions.
- The central feature of PG/ECLIPSE is *script management* (c.f. [Bertot, 1998]), which is a form of step debugging tailored to linear scripts.
- *Symbol support* can make a huge difference to how readable a proof script is. This is possibly an area where TP interfaces have something to offer to programming interfaces. PG/ECLIPSE provides support for using mathematical symbols – including the use of typing shortcuts to enter symbols, and a symbol table editor, allowing users to adapt and extend the use of symbols to fit their own needs.
- *Theory navigation:* As a theory grows, finding specific definitions and proofs can become increasingly difficult and time consuming. This alone can make a large difference to the usefulness of a theory. PG/ECLIPSE provides support for theory navigation. As proof script files are read, their structure is analysed and the theories, theorems and lemmas they contain are indexed. This index is then used to provide several ways to navigate files.
- A *Content Assistant* which suggests completions for keywords/phrases.
- Good documentation and help is key to both the uptake of theorem provers and the reuse of proof scripts – but is often neglected. PG/ECLIPSE provides tools for documenting prover commands, prover settings and proof scripts, with integrated help displays. Help for theory elements (e.g. theorems) is created simply by writing a preceding comment (an idea taken from the Java approach to documenting code (c.f. [Friendly, 1995]).
- One of the great potentials for provers is in mathematical/scientific education. Although there are excellent computer algebra systems used in education, these neither perform nor teach proofs, which are central to mathematical work. PG/ECLIPSE introduces a teaching tool (based around an embedded web-browser), designed for delivering teaching material that interacts with a prover.
- *Interface scripting* Often a script will run better under certain settings. PG/ECLIPSE allows a theory developer to encode these settings in the proof script file. It defines a small set of commands that can be used to script the interface itself.

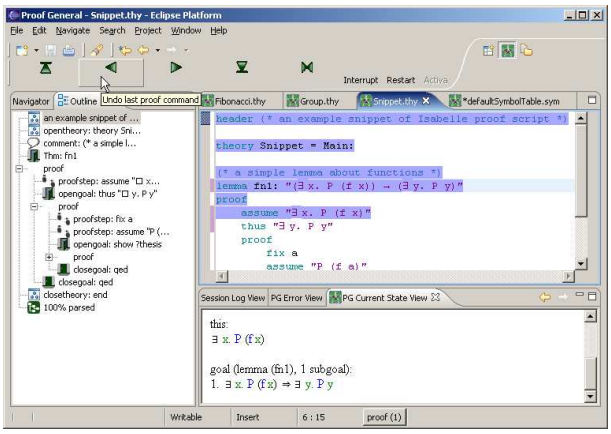

Figure 1: The PG/ECLIPSE display. The main window shows the proof script; view windows below show the prover output. A Problems view (not shown here) lists outstanding problems, such as syntax errors or unfinished proof-goals. To the left of the editor window is an outline view of the proof's structure. Above the editor, a toolbar triggers proof or undo steps by sending instructions to the prover (left-to-right, the buttons are: *undo all*, *undo*, *interrupt*, *step forward*, *process all*, and – most useful of all – *go to cursor location*).

3 Design principles for prover interfaces

Here we describe two design principles specific to theorem prover interfaces.

3.1 Proof scripting is a form of *programming*

It has often been noted that theorem proving has a great many similarities with programming. However this insight has not really been used in prover interface design. Since programming interfaces serve a much larger community, they have been much more actively developed. We therefore borrow ideas from programming interfaces where possible. One consequence of this was the decision to use Eclipse as a base. Eclipse is an open-source state-of-the-art IDE, originally intended for Java programming [Eclipse Foundation, 2003].

3.2 Separation of proof engine and interface

This is a form of modularisation. Modularisation is especially important in this domain, where – inspite of the relatively small size of the community – there are a range of target applications and a diverse wealth of systems. Each of these needs a good interface. Hence separating prover and interface should facilitate the development of TPs, by giving TP developers easy access to sophisticated interfaces. It would also benefit prover interface and application designers, who need not be tied to one proof system.

The Proof General Kit

We enforce this clear separation by specifying an API (the PG/KIT) for all prover interactions *and* all knowledge of prover behaviour. This API consists of two linked parts: An abstract model for prover behaviour, and a protocol for communication within proof sessions.

The model is based on abstracting the common behaviour of many interactive proof systems. It acts as a clearly specified "virtual layer" that must be emulated in each prover to cooperate properly with PG/ECLIPSE.

The communication protocol is called *PGIP*, for *Proof General Interactive Proof*. The basic principle for representing proof scripts in PGIP is to keep the prover's native language, and mark up the script with PGIP tags (which give the information needed by an interface). Designing PGIP as a wrapper for native languages has two main consequences:

Firstly, the user employs the prover's native language (PGIP mark-up is only used internally). This is necessary, since the wide variety of logics supported by modern provers make it very hard to come up with one language which efficiently supports all of these. It is also pragmatic, as users can continue to work in the proof language they are familiar with, and old proof scripts can still be used.

Secondly, the PGIP-annotation of the proof script (and subsequent user input) is done by the prover (or a component coming with the prover), and not by the interface, which knows nothing about the prover's language. Parsing requests and responses form part of the PGIP protocol.

4 Related work

We can only give a brief overview here of the closest related projects.

The Theorema theorem prover and the TeXMacs systems both have excellent support for entering and viewing mathematical expressions [Buchberger, 2001][Audebaud, 2003]. Various systems already implement script management (e.g. the previous PG system). However, these offer less functionality than PG/ECLIPSE, and do not provide a satisfactory solution to the problem of handling different provers. The most closely related work is that done within the PG/KIT framework: a PGIP Emacs mode, and a PGIP-based 'proof desktop', where theories and proofs are built up using graphical actions such as drag-and-drop. There is also a *broker* component in that can act as a middle-man between a collection of PGIP-equipped provers and interfaces. This should lead to increasingly flexible ways to develop proofs.

5 Future work

There are many possible lines for future development. Firstly, we want to use the Eclipse framework to further explore the analogy between theory development and software engineering. We can also go beyond adapting program development tools. One promising line of work is on using interactive proof planning to construct proof scripts (c.f. [Fleuriot, 2003]). Interactive planning could also be a valuable idea to transfer to IDE based programming. The closest existing analogue is the idea of 'programming templates'; fragments of code which can be used to help build up programs (e.g. a standard widget initialisation method). Proof planning can be considerably more powerful.

5.1 State of the project

This work is described in more depth at http://proofgeneral.inf.ed.ac.uk/Kit. An alpha-release of PG/ECLIPSE is now available (also from this webiste), and the latest version of Isabelle (available from www.cl.cam.ac.uk/Research/HVG/Isabelle) supports PGIP. Developers interested in using Proof General for other provers should contact the authors.

References

[Audebaud, 2003] P.Audebaud & L.Rideau. Texmacs as authoring tool for formal developments. In *User Interfaces for Theorem Provers UITP'03*, 2003.

[Bertot, 1998] Y.Bertot & L.Théry. A generic approach to building user interfaces for theorem provers. *Journal of Symbolic Computation*, 1998.

[Buchberger, 2001] B.Buchberger. Theorema: A short introduction. *Mathematica Journal*, 8:247–252, 2001.

[Eclipse Foundation, 2003] Eclipse Foundation. Eclipse platform technical overview. www.eclipse.org, 2003.

[Fleuriot, 2003] L.Dixon & J.Fleuriot. Isaplanner: A prototype proof planner in isabelle. In *19th International Conference on Automated Deduction*, 2003.

[Friendly, 1995] L.Friendly. The design of distributed hyperlinked programming documentation. In *International Workshop on Hypermedia Design '95*, 1995.