Proof Tool Integration with Proof General

David Aspinall[⋆]

[⋆]School of Informatics
University of Edinburgh
Scotland
David.Aspinall@ed.ac.uk

Christoph Lüth[†]

[†] Deutsches Forschungszentrum für künstliche Intelligenz (DFKI)
Bremen
Germany
Christoph.Lueth@dfki.de

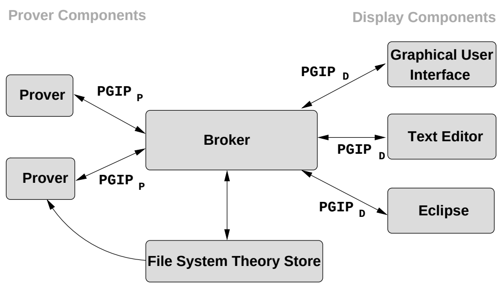
Abstract

We give a brief overview of the *Proof General Kit* software framework for proof development. The goal of the framework is to enable flexible environments for managing formal proofs across their life-cycle: creation, maintenance and exploitation. The framework connects together different kinds of component, exchanging messages using a common communication infrastructure and protocol called *PGIP*. We describe the main component of our framework, *Proof General Eclipse*, and our plans to turn it into an integration platform for proof tools.

1 A kit of components for directing proof

The *Proof General* system (Aspinall, 2000) provides a popular and generic interface for interactive proof assistants, whose central metaphor is *script management*. Script management sends lines of proof one-by-one to a proof assistant in an interactive dialogue, colouring the proof document to indicate the state of processing and to keep synchronization with the proof assistant. Proof General is almost 10 years old by now, and is held back by its Emacs-based architecture. In late 2003, we started work on building a new component-based architecture for interactive proof called *Proof General Kit*, pictured below.

Components connect to a central *broker* middleware, which is responsible for managing proof components and controlling the progress of proof. Components communicate using the *PGIP* protocol. PGIP comprises three sub-protocols, corresponding to the different types of components illustrated.



- The *prover protocol* PGIP$_{\mathrm{P}}$ defines messages exchanged between proof tools (primarily interactive theorem provers) and the broker. This captures the control mechanisms of script management, including command processing, state synchronization and state examination.
- The *display protocol* PGIP$_{\mathrm{D}}$ defines messages exchanged between displays and the broker. This captures the interaction mechanisms of script management, including requests to load files, process proof steps, and object querying. Messages to display to the user are also part of this protocol fragment. A message model specifies different modes and locations for message appearance.
- The *inter-broker protocol* PGIP$_{\mathrm{I}}$ defines messages exchanged between different brokers, for example allowing running the prover on a remote machine.

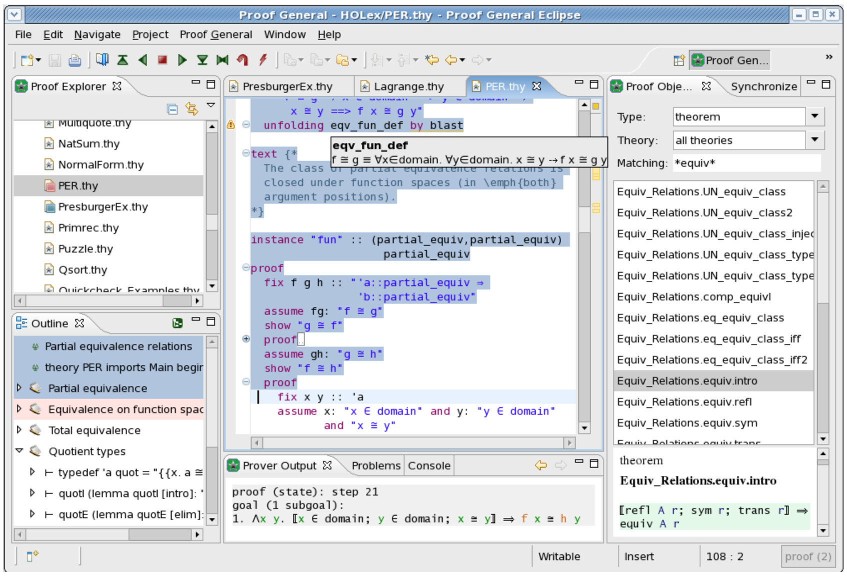
The syntax of PGIP messages is defined by an XML schema written in RELAX NG. Every message is wrapped in a `<pgip>` packet which uniquely identifies its origin and contains a sequence number and possibly a referent identifier and sequence number.

To help control script management and provide user-level navigation, documents have PGIP mark-up added by a parser in the theorem prover. For example, the statement of proof is labelled with an `<opengoal>` element, the proof commands or declarations with it are tagged with `<proofstep>`, and the end of the proof is labelled with `<closegoal>`. Attributes within these elements can indicate the names and contents of statements being proved, as well as declarations and definitions. The markup is imposed on the native proof script text without changing it in any way, and is never directly seen by the user.

2 Interactive proof development in Eclipse

The main UI in our architecture is Proof General Eclipse, a plugin for the Eclise IDE. It provides features including:

- Script management, including synchronization of multiple files and a navigator which indicates file state.
- Editing features: outlining and folding of proofs, tokenised symbol sequences, syntax highlighting, script formatting, proof views.
- Platform features: problems display and markers on text; bookmarks and tasks lists; CVS or Subversion integration; system consoles.


Proof General - HOLex/PER.thy - Proof General Eclipse
File Edit Navigate Project Proof General Window Help
Proof Gen...
Proof Explorer
Multiquote.thy
NatSum.thy
NormalForm.thy
PER.thy
PresburgerEx.thy
Primrec.thy
Puzzle.thy
Qsort.thy
Quickcheck_Examples.thy
Outline
Partial equivalence relations
theory PER imports Main begin
Partial equivalence
Equivalence on function spac
Total equivalence
Quotient types
typedef 'a quot = "{{x. a ≅
quotI (lemma quotI [intro]: '
quotE (lemma quotE [elim]:
PresburgerEx.thy
Lagrange.thy
PER.thy
Proof Obje...
Synchronize
Type: theorem
Theory: all theories
Matching: *equiv*
Equiv_Relations.UN_equiv_class
Equiv_Relations.UN_equiv_class2
Equiv_Relations.UN_equiv_class_injec
Equiv_Relations.UN_equiv_class_type
Equiv_Relations.UN_equiv_class_type
Equiv_Relations.comp_equivI
Equiv_Relations.eq_equiv_class
Equiv_Relations.eq_equiv_class_iff
Equiv_Relations.eq_equiv_class_iff2
Equiv_Relations.equiv.intro
Equiv_Relations.equiv.refl
Equiv_Relations.equiv.sym
Equiv_Relations.equiv.trans
Prover Output
Problems
Console
Writable
Insert
108 : 2
proof (2)

```
      x ≅ y ==> f x ≅ g y"
  unfolding eqv_fun_def by blast

text {*
  The class of partial equivalence relations is
  closed under function spaces (in \emph{both}
  argument positions).
*}

instance "fun" :: (partial_equiv,partial_equiv)
                   partial_equiv
proof
  fix f g h :: "'a::partial_equiv ⇒
                'b::partial_equiv"
  assume fg: "f ≅ g"
  show "g ≅ f"
  proof
  assume gh: "g ≅ h"
  show "f ≅ h"
  proof
    fix x y :: 'a
    assume x: "x ∈ domain" and y: "y ∈ domain"
          and "x ≅ y"
```

```
eqv_fun_def
f ≅ g ≡ ∀x∈domain. ∀y∈domain. x ≅ y → f x ≅ g y
```

```
proof (state): step 21
goal (1 subgoal):
1. ⋀x y. ⟦x ∈ domain; y ∈ domain; x ≅ y⟧ ⟹ f x ≅ h y
```

theorem
Equiv_Relations.equiv.intro

```
⟦refl A r; sym r; trans r⟧ ⟹
equiv A r
```

3 From script management to proof tool integration

With PGIP markup, Proof General provides a means for manipulating proof documents and attaching arbitrary meta-data. In previous work we have proposed literate extensions of this document mechanism, primarily to assist in constructing human-oriented texts at the same time as machine-oriented texts (Aspinall et al., 2006). A new extension we are currently planning is to allow generic mechanisms for proof texts to specify the use of different proof tools to prove particular lemmas, or even eventually the transfer of partial developments between different proof systems. We believe that providing generic, lightweight means for individually justified embeddings and transformations could be very useful in practice, and technically much easier than a fully formalised globally-sound logical framework which imposes a heavy burden. Indeed, this approach is already being followed with work on connecting resolution provers and model checkers to existing interactive theorem provers (e.g., Meng and Paulson (2004); Shankar (2000)). Rather than repeat ad hoc and different proof language extensions anew in each different theorem prover, we want to provide for these kind of connections in a uniform way, turning Proof General into an integration platform for different proof tools.

The basic idea is simple. We can extend our proof document model by a simple annotation of which prover can check particular parts of the script, such that scripts can become heterogeneous. We extend the proof document mechanism with ways to transfer proof obligations between systems, given some support from the underlying proof systems to transfer theorem statements (and theory contexts) into a standard format such as TPTP. Within the document, we exploit this by adding another form of document generating instruction in a literate markup scheme. Previously we considered document generating instructions to generate human-oriented texts (i.e., LaTeX articles) from fragments of theorem prover output. By a similar scheme we generate system texts (i.e., proof fragments) by calling out to other systems. When the proof script is processed by Proof General, the output of different systems can be checked and cached into the master document. As Eclipse already provides an integration platform for tools, Proof General Eclipse is a natural place to implement this idea.

References

David Aspinall. Proof General: A generic tool for proof development. In S. Graf and M. Schwartzbach, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1785, pages 38–42. Springer, 2000.

David Aspinall, Christoph Lüth, and Burkhart Wolff. Assisted proof document authoring. In Michael Kohlhase, editor, *Mathematical Knowledge Management MKM 2005*, LNAI 3863, pages 65– 80. Springer, 2006.

Jia Meng and Lawrence C. Paulson. Experiments on supporting interactive proof using resolution. In David A. Basin and Michal Rusinowitch, editors, *IJCAR*, volume 3097 of *LNCS*, pages 372–384. Springer, 2004.

Natarajan Shankar. Combining theorem proving and model checking through symbolic analysis. In Catuscia Palamidessi, editor, *CONCUR*, volume 1877 of *LNCS*, pages 1–16. Springer, 2000.