Proof General

A Generic Tool for Proof Development

David Aspinall

LFCS, University of Edinburgh

http://www.lfcs.informatics.ed.ac.uk/proofgen
Introduction

Background
Why Proof General?
What is Proof General?
• Terminology: *machine proof*
  - formal machine representation of mathematical/logical proof

• Machine proofs useful in
  - specification, development, verification of software and hardware
  - teaching mathematical proof and formal logic
  - mathematical research

• Terminology: *proof assistant* (or *prover*)
  - an *interactive* computerized helper for developing machine proofs

• Terminology: *proof script*
  - user-level input to prover which constructs a machine proof
  - may contain procedural proofs (LCF style), or declarative proofs (Mizar style)
  - stored in a file, like a program
Why Proof General?

- Many proof assistants still have only a primitive interface
  - It’s easy to program!
  - Experts unafraid of cryptic command language

- But a modern interface has advantages:
  - Saves time for experts, providing short-cuts
  - Helps novices, providing hints
  - Opens the way to higher-level interactions

- A *generic* interface is attractive:
  - Saves time for implementors, can concentrate on logical bits
  - Helps users try different systems, using the same interactions
What is Proof General?

- A generic interface based on Emacs
- It provides many useful features, including:
  - script centred development
  - script management
  - proof by pointing
  - helpful toolbar and menus
  - coloured output and special fonts for maths, . . .
- It presently has support for Isabelle(/Isar), Coq, LEGO, Plastic, HOL98
- More support and development is on the way . . .

An idea: a generic tool to help proof development.
An attitude: be useful both to novices and to experts.
Generic aspects of proof assistants
Choose Emacs
System architecture
Generic aspects of proof assistants

- Interaction has a common structure
  - User makes declarations or definitions
  - User enters *proof dialogue*
    - user gives proof step; system responds (e.g. subgoal list)
    - repeat

- Proof scripts have a common structure, similarly:
  - declarations and definitions, and
  - *goal . . . save* sequences

- Primitive interfaces have common structure:
  - Command-line interface: *proof assistant shell*

How can we build a system to exploit these common structures?
Choose Emacs

- The world's best text editor also provides a user-interface toolkit!

Choosing emacs has pros

- user familiarity: Emacs already used to write scripts
- portability: runs on MS Windows, Unix, Linux, ...
- interpreted scripting language for development: Emacs Lisp
- extensive libraries, easy user-customization

...and cons

- hard to learn and over complicated
- the original bloatware
- interoperability limited (live in Emacs!)
- single-threaded
System architecture

Emacs

+-------------------------+ Engine
|                          |
| +---------------+      +-------------------------+
| |               |      |
| |   |            |      |
| | <---|          |      |
| |     |          |      |
| |     |          |      |
| |     |          |      |
| +-------------------------+      +-------------------------+

File

v System

v

USER

+------>

+-------------------------+
User Features

Script centered development
Script management
Proof by Pointing
User friendliness
Other Emacs features
Script centered development

- Hide irrelevant information
  - shell hidden
  - but still available for emergencies

- Buffer display model: two-of-three window panes
  - *script*
  - *goals* or *response*

- Script buffer centred around “latest” proof command

- Goals buffer centred around working subgoal

- Response buffer displays other relevant messages
  - urgent messages
  - result of non-proof step (search results, command feedback)

- Customizable to use three buffers and multiple windows
Script management

- Synchronizes editor with proof assistant
- Provides visual feedback
  
  
  blue background — processed text
  
  pink background — text being processed

- Highlighted text is *locked* to prevent accidental editing
- Connects with prover’s history mechanism, for *retraction*
  - undo individual steps within a proof
  - block-structure outside proof
- Connects with prover’s file handling
  - extend synchronization to multiple files
  - dependencies communicated or deduced automatically
- Avoids using cut-and-paste or “load file” commands
**Proof by Pointing**

- Click on subterm of goal
  - generates proof command to simplify/solve goal
  - inserts command into proof
  - executes it

- Support from proof assistant required!
  - annotations to markup term-structure
  - communication of position in AST
  - proof command generation

- Many possibilities
  - context-sensitive menus
  - other gestures (e.g. drag term to rearrange equation)
  - not yet implemented
User friendliness

- Toolbar
  - buttons to start proof, process step, undo step, finish proof, . . .
- Menus
  - change display modes, start/stop proof assistant, . . .
  - all commands available here
- Easy preference setting
- Online documentation
  - variety of formats
  - links to proof assistant documentation
- . . .and of course, speedy short-cut key sequences like
  C-c C-RET proof-goto-point
Other Emacs features

- Syntax highlighting
  - decoration of proof scripts and prover output
- Symbol fonts
  - glyphs for logical symbols, greek letters, etc
  \[\phi \rightarrow \psi\] instead of \[\text{phi} \rightarrow \text{psi}\]
- Tags
  - search for definitions and proofs amongst many files
- Item menu
  - navigate to definitions and proofs in current window
- Remote proof assistant
  - run prover on different machine using \texttt{rsh} or \texttt{ssh}
Implementation

Implementation notes
Instantiation mechanism
Example instantiation
Development model
Implementation notes

- Main implementation in Emacs Lisp
  - 7000 loc for generic parts
  - 30 – 500 loc per assistant for prover specific parts

- Support in proof assistant (optional)
  - output markup for robustness
  - file loading messages
  - proof by pointing machinery

- Emacs Lisp issues
  - fairly primitive, but has some CL macros (and CLOS emulation)
  - slow, but built-ins and byte-code compilation improve matters
  - easy to learn and use, docstrings are wonderful
Instantiation mechanism

- 80 configuration settings total; may only need half. Organized as:
  - Regexps to recognize proof script
  - Regexps to recognize prover messages
  - Commands to control prover
  - Hooks to configure behaviour

- Some important examples:

  proof-goal-command-regexp       matches goal command in script
  proof-shell-start-goals-regexp  matches start of goals output
  proof-prog-name                 command to start prover
  proof-shell-insert-hook         hook to tweak prover input

- One line to add autoloads, name, customizations for new prover

- Use define-derived-mode for new script, goals, response, shell

- With new “easy configure” mechanism, no Elisp necessary!
Example instantiation

(require 'proof-easy-config) ; easy configure mechanism
(proof-easy-config
  'demoisa "Isabelle Demo"
  proof-prog-name "isabelle"
  proof-terminal-char ?\;
  proof-comment-start "(*"
  proof-comment-end "*)"
  proof-goal-command-regexp ^Goal
  proof-save-command-regexp "~qed"
  proof-goal-with-hole-regexp "qed_goal "\(\(.*\)\)"
  proof-save-with-hole-regexp "qed "\(\(\.*)\)"
  proof-non-undoables-regexp "undo\|back"
  proof-goal-command "Goal \%s\;"
  proof-save-command "qed \%s\;"
  proof-undo-n-times-cmd "pg_repeat undo \%s;"
  proof-auto-multiple-files t
  proof-shell-cd-cmd "cd \%s"
  proof-shell-prompt-pattern "[ML=-#>]+?> "
  proof-shell-interrupt-regexp "Interrupt"
  proof-shell-start-goals-regexp "Level [0-9]"
  proof-shell-end-goals-regexp "val it"
  proof-shell-quit-cmd "quit();"
  proof-assistant-home-page
  "http://www.cl.cam.ac.uk/Research/HVG/isabelle.html"
  proof-shell-annotated-prompt-regexp "\(\val it = () : unit\n\)?ML>?
  proof-shell-error-regexp
  "\(\*\*\*\*\|\"uncaught exception \"|^Exception- "
  proof-shell-init-cmd
  "fun pg_repeat f 0 = () | pg_repeat f n = (f(); pg_repeat f (n-1));"
  proof-shell-proof-completed-regexp "\(\\|\"No subgoals!\n\)"
  proof-shell-eager-annotation-start "\(\[opening \"\|\"Reading"
  (provide 'demoisa)
Development model

- Successive generalization
  - generalize as needed
  - sometimes extend and redesign core, as needed
  - LEGO mode $\rightarrow$ Proof Mode $\rightarrow$ Proof General

- Developer/maintainer in each camp
  - Emacs and prover support for each prover
  - adds specific features, generalizes if useful elsewhere
  - serves as primary user/tester

- CVS server, access to whole repository for all developers

- Frequent pre-release versions, quick response to bugs

- Open source, user contributions welcomed
Introduction
Architecture
User Features
Implementation
Appraisal
Future
The End

Appraisal

Usage
Comparison
Benefits of Proof General
• Target users of currently supported proof assistants:

<table>
<thead>
<tr>
<th>User community</th>
<th>Other interfaces?</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEGO</td>
<td>30</td>
</tr>
<tr>
<td>Coq</td>
<td>80</td>
</tr>
<tr>
<td>Isabelle</td>
<td>200</td>
</tr>
<tr>
<td>Isabelle/Isar</td>
<td>20</td>
</tr>
<tr>
<td>Plastic</td>
<td>5</td>
</tr>
<tr>
<td>HOL98</td>
<td>200</td>
</tr>
</tbody>
</table>

• Other possible systems (HOL variants, Agda, VDM, ACL2, . . .)

• Use in teaching
  – 2000 EEF Foundations school in Deduction and Theorem Proving
  – 1999 Types Summer School: 50 learning LEGO, Coq, and Isabelle
  – MSc/PhD course in formal reasoning at Edinburgh

• Current version is 3.1, about 100 registered users as of May 2000.
Comparison

- There’s more sophistication elsewhere:
  - Graphical representations: proof-trees, direct manipulation
  - Structure editing, integrated environments, . . .

- However, Proof General has complementary aspects:
  - intended for day-to-day proof, not an experiment in HCI
  - draws on familiarity (text editor), uniformity (between systems)
  - scales to large proofs
  - portable, easy to adapt and extend

Proof General occupies a middle ground in interface technology
**Benefits of Proof General**

- A nice front-end for doing real work!
- Is being used by experts, doesn't get in their way (much)
- But is also used for teaching novices
- Replaying proofs is trivial
- By construction, it suggests a protocol for interactive proof
  - New project to design standard extensible protocol . . .
- Very easy to install; self-configuring
- Very easy to adapt to new systems, to get basic features

*Proof General achieves a lightweight, *useful* interface at little cost*
Future

- Evolutionary
  - More features — completion, favourites, theory browser
  - More proof assistants

- Revolutionary
  - Factor out script management, use for programming languages
  - Standardize markup mechanism (XML, MathML, OpenMath, ATerms)
  - Focus on protocols, move away from purely Emacs
  - Middleware layer connects proof engine to front-ends (CORBA)

- Imaginary
  - Prover-independent syntax mechanisms
  - Logic and theory mappings, standard taxonomies

**Working title of next project:** Proof General Kit
New Users, Developers Welcome!

- Enjoy using Proof General
- Add support for another prover
- Undertake a project
- Contribute to future design

Credits:

- Thomas Kleymann,
- Yyves Bertot and CtCoq
- Dilip Sequeira, Healfdene Goguen,
- Markus Wenzel, David von Oheimb, . . .

For more, visit [http://www.lfcs.informatics.ed.ac.uk/proofgen](http://www.lfcs.informatics.ed.ac.uk/proofgen)