Proof General

A Generic Interface for Proof Development

David Aspinall

LFCS, University of Edinburgh

http://www.dcs.ed.ac.uk/home/proofgen

Introduction

Background
Why Proof General?

Background

- 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*)
  - a computerized helper for developing machine proofs
- Terminology: *proof script*
  - input to proof assistant which constructs a machine proof
  - stored in a file

Why Proof General?

- Many proof assistants still have a primitive interface
  - It's easy to program!
  - Logicians unafraid of cryptic codes (?)
- But a modern interface has advantages:
  - Saves time for experts, providing short-cuts
  - Helpful to novices, providing hints
  - Opens the way to higher-level interactions
- A *generic* interface is attractive
  - Can hook-up to many proof assistants
  - Use the same mechanism for interaction

Proof General aims to be a generic interface, useful both to novices and to experts.

Architecture

Generic aspects of proof assistants

Choose Emacs

System architecture

Generic aspects of proof assistants

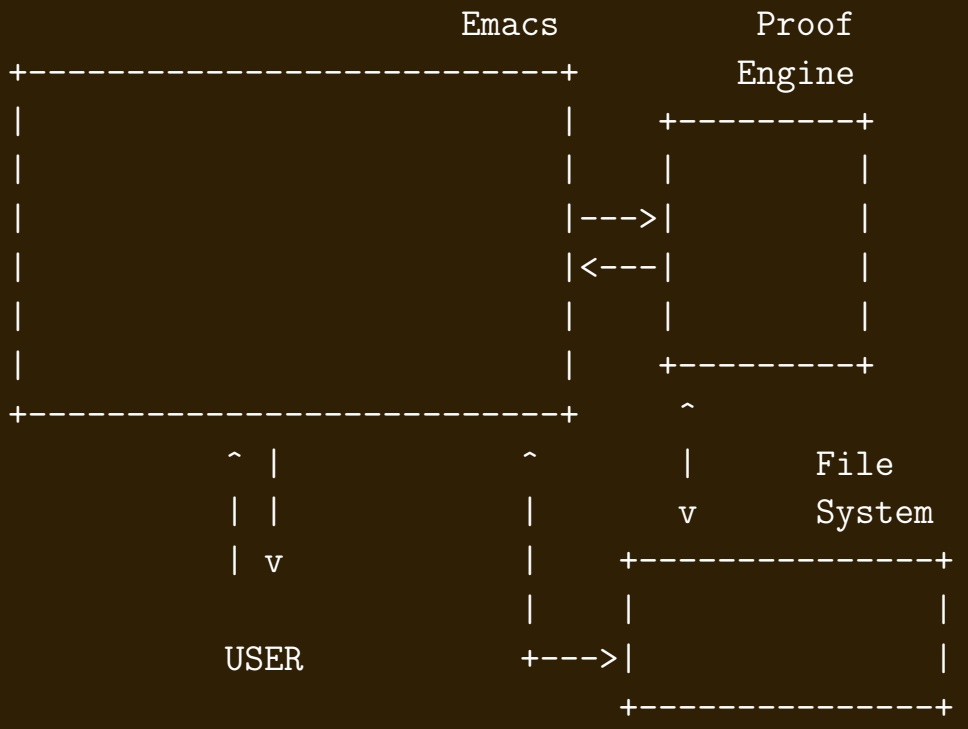
- Proof scripts have a common structure:
  - declarations and definitions, and
  - *goal* . . . *save* sequences
- Interaction has a common structure:
  - User issues declaration or definition
  - User enters *goal-directed proof* dialogue
    - ⋆ user gives proof step; system responds with subgoal list
    - ⋆ repeat
- 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
  - portability: runs on NT, Unix, Linux, . . .
  - extensive libraries
  - user familiarity, easy user-customization
  - interpreted scripting language for development: Emacs Lisp
- . . . and cons
  - hard to learn and over complicated
  - the original bloatware
  - interoperability limited (live in Emacs!)
  - single-threaded

System architecture



User Features

Simplified communication

Script management

Proof by Pointing

User friendliness

Other Emacs features

Simplified communication

- 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 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, magical 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

  $\phi \longrightarrow \psi$ instead of `phi --> 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 `rsh` or `ssh`

Implementation

Implementation notes
Instantiation mechanism
Example instantiation
Development model

Implementation notes

- Emacs Lisp
  - about 7000 loc for generic part
  - 30 – 500 loc for specific part for each assistant
- 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
  - use `define-derived-mode` for inheritance
  - slow, but built-ins and byte-code compilation improve matters
  - *docstrings* are wonderful

Instantiation mechanism

- 80 configuration settings, split up into:
  - Regexps to recognize proof script
  - Regexps to recognize prover messages
  - Commands to control prover
  - Hooks to configure behaviour
- Some important settings:

`proof-prog-name`	command to start prover
`proof-shell-annotated-prompt-regexp`	matches end of prover output
`proof-goal-command-regexp`	matches goal command in script
`proof-shell-start-goals-regexp`	matches start of goals output
`proof-shell-eager-annotation-start`	matches important messages

- 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-kill-goal-command         "Goal \"PROP no_goal_set\";"
 proof-showproof-command         "pr()"
 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
 "\\*\\*\\*\\|^.*Error:\\|^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    "\\(\\(.\\|\n\\)*No subgoals!\n\\)"
 proof-shell-eager-annotation-start    "^\\[opening \\|^###\\|^Reading")
(provide 'demoisa)
```

Development model

- Incremental genericity
  - generalize as necessary
  - occasionally extend and redesign core
- 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
- Frequent pre-release versions, quick response to bugs
- Extreme techniques . . . ?

Appraisal

Benefits of Proof General

Usage

Comparison

Benefits of Proof General

- A nice front-end for doing real work
- By construction, suggests a protocol for proof assistants

Usage

- Currently supported proof assistants, target users:

	User community	Other interfaces?
LEGO	30	no
Coq	80	yes
Isabelle	150	yes
Isabelle/Isar	30	no
Plastic	5	no

- Several other possible systems (HOL variants, Agda, . . .)
- Use in teaching
  - 1999 Types Summer School: 50 students learning LEGO, Coq, and Isabelle
  - CAFR MSc and PhD course at Edinburgh
- Version 3.0 announced 1st December
  - 150 visits in 2 days, 60 downloads

Comparison

- There's more sophistication elsewhere:
  - Graphical representations
    - proof-trees
    - direct manipulation
  - Structure editing
  - Integrated environments
- However, Proof General has complementary advantages:
  - familiarity, uniformity
  - portability, adaptability
  - easy instantiation
  - easy modification

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
  - Use standard markup mechanisms (MathML, XML, ATerms)
  - Focus on protocols, move away from Emacs
  - Middleware layer connects proof engine to front-end (CORBA?)
- Imaginary
  - Prover-independent syntax mechanisms
  - Logic and theory mappings, standard taxonomies
  - Distributed proof development, theorem proving agents

Credits

- People
  - Thomas Kleymann
  - Yyves Bertot and Project CROAP
  - Dilip Sequeira
  - Healfdene Goguen
  - Markus Wenzel
  - others . . .
- Funding
  - LFCS
  - EPSRC Grant for LEGO
  - EC BRA Types

The End

Version 1.9 of 1999/12/14 20:37:57, processed December 14, 1999