On prover interaction and integration with Isabelle/Scala

Makarius Wenzel
TU München

October 2009

GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

Abstract

After several decades, most proof assistants are still centered around tty-bound interaction in a tight read-eval-print loop. Even well-known Emacs modes for such provers follow this synchronous model based on single commands. There have also been some attempts at re-implementing prover interfaces in big IDE frameworks, while keeping the old interaction model. Can we do better than that?

Already 10 years ago, the Isabelle/Isar proof language has emphasized the idea of "proof document" (structured text) instead of "proof script" (sequence of commands), although the implementation was still emulating tty interaction in order to be able to work with the existing Proof General interface. After some recent reworking of Isabelle internals, in order to support parallel processing of theories and proofs, the original idea of structured document processing has surfaced again.

Isabelle2009 already provides some support for interactive proof documents with asynchronous / parallel checking, which awaits to be connected to a suitable editor framework (or "IDE"). The remaining problem is how to do that systematically,

without having to specify and implement complex protocols or middle-ware for prover integration.

This is the point where we introduce the new Isabelle/Scala layer, which is meant to expose certain aspects of Isabelle/ML to the outside world. The Scala language (by Martin Odersky) is sufficiently close to ML in order to model well-known prover concepts conveniently, but Scala also runs on the JVM and can access existing Java libraries without requiring additional glue code. By building more and more external system wrapping for Isabelle in Scala, we shall eventually reach the point where we can integrate the prover seamlessly into existing IDEs (say Netbeans). Although current experiments are focused on relatively simple editor functionality in jEdit, the emerging Isabelle/Scala library is targeting general tool integration.

For example, an Isabelle/Isar proof method might want to call into the Scala/JVM world to invoke some ATP or model finder that happens to be available there. Or some external tools might want to invoke Isabelle as a ”service”, by interacting directly with a stylized Scala API, instead of generating adhoc text files.

Introduction

Motivation

General aims:

- renovate and reform traditional "LCF-style" theorem proving for coming generations of users and tool developers
- catch up with technological shifts, e.g. advanced user-interfaces, parallel computing
- support novel models for interactive proof checking

Possible applications:

- web client, based on server-side prover component
- powerful proof editor, or "Prover IDE"
- integration with external tools via Scala/JVM

Isabelle/Isar: proof documents

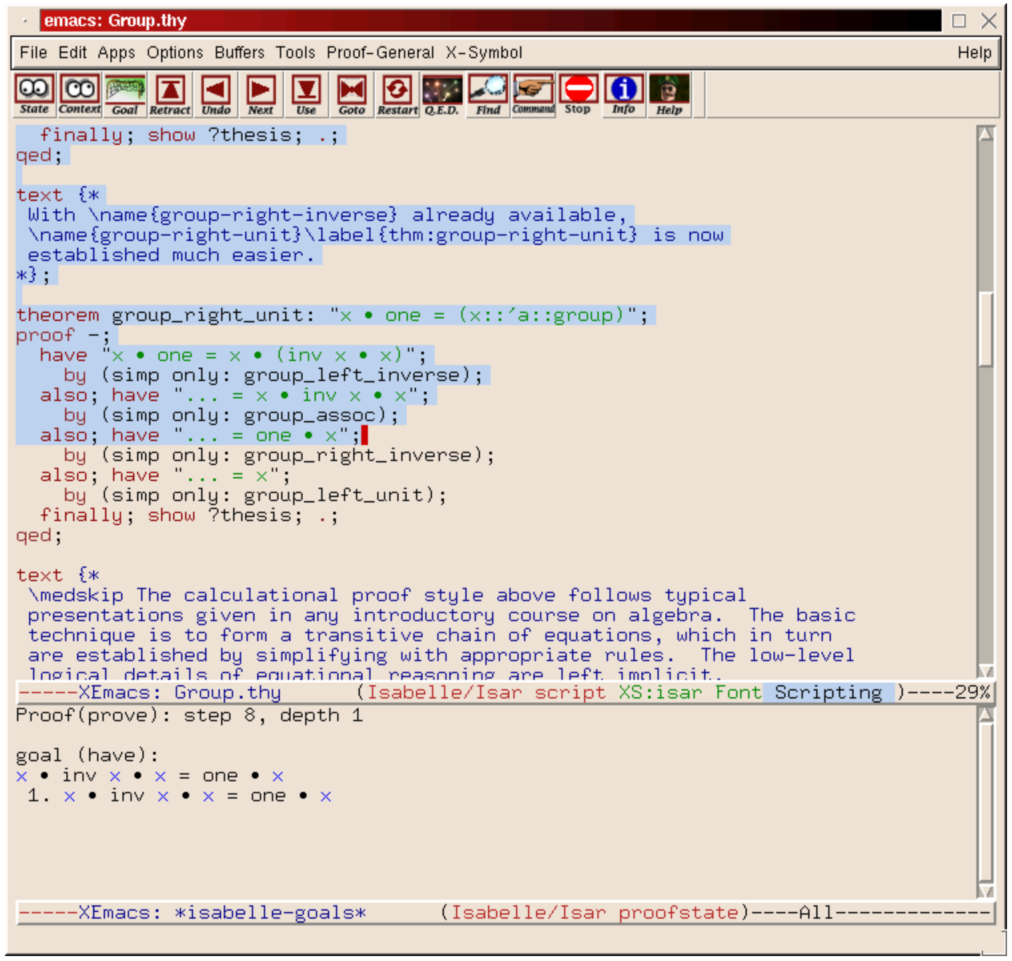
theory C **imports** $A\ B$ **begin**

inductive $path$ **for** $rel :: \alpha \Rightarrow \alpha \Rightarrow bool$ **where**
  $base$: $path\ rel\ x\ x$
$|$ $step$: $rel\ x\ y \Longrightarrow path\ rel\ y\ z \Longrightarrow path\ rel\ x\ z$

theorem $example$: **fixes** $x\ z :: \alpha$ **assumes** $path\ rel\ x\ z$ **shows** $P\ x\ z$
using $assms$
proof $induct$
  fix x **show** $P\ x\ x$ $\langle proof \rangle$
next
  fix $x\ y\ z$ **assume** $rel\ x\ y$ **and** $path\ rel\ y\ z$
  moreover assume $P\ y\ z$
  ultimately show $P\ x\ z$ $\langle proof \rangle$
qed

end

Proof General: proof scripts


emacs: Group.thy
File Edit Apps Options Buffers Tools Proof-General X-Symbol Help
State Context Goal Retract Undo Next Use Goto Restart Q.E.D. Find Command Stop Info Help

```
  finally; show ?thesis; .;
qed;

text {*
 With \name{group-right-inverse} already available,
 \name{group-right-unit}\label{thm:group-right-unit} is now
 established much easier.
*};

theorem group_right_unit: "x • one = (x::'a::group)";
proof -;
  have "x • one = x • (inv x • x)";
    by (simp only: group_left_inverse);
  also; have "... = x • inv x • x";
    by (simp only: group_assoc);
  also; have "... = one • x";
    by (simp only: group_right_inverse);
  also; have "... = x";
    by (simp only: group_left_unit);
  finally; show ?thesis; .;
qed;

text {*
 \medskip The calculational proof style above follows typical
 presentations given in any introductory course on algebra.  The basic
 technique is to form a transitive chain of equations, which in turn
 are established by simplifying with appropriate rules.  The low-level
 logical details of equational reasoning are left implicit.
-----XEmacs: Group.thy       (Isabelle/Isar script XS:isar Font Scripting )----29%
Proof(prove): step 8, depth 1

goal (have):
x • inv x • x = one • x
 1. x • inv x • x = one • x

-----XEmacs: *isabelle-goals*       (Isabelle/Isar proofstate)----All-------------
```

Main characteristics:

- sequential checking of *proof scripts*
- one frontier between checked/unchecked
- one proof state
- one response
- mostly synchronous (interface may block)

Efficient document processing

```
theory C imports A B begin

inductive path for rel :: α ⇒ α ⇒ bool where
  base: path rel x x
| step: rel x y ⟹ path rel y z ⟹ path rel x z ⟨internal proof⟩

theorem
  example: fixes x z :: α assumes path rel x z shows P x z
    using assms
    proof induct
      fix x show P x x ⟨proof⟩
    next
      fix x y z assume rel x y and path rel y z
      moreover assume P y z
      ultimately show P x z ⟨proof⟩
    qed

end
```

Main characteristics:

- continous checking of *structured text*
- no “locked region”
- essentially state-less
- feedback as source annotations
- asynchronous (and parallel)

General aims

- Overcoming the TTY model (read-eval-print loop)
- Overcoming two-buffer (or three-buffer) model
  (Proof General and clones)
- Overcoming Emacs — use jEdit (or Netbeans, or ???)

- Towards parallel and asynchronous interative proof checking
- Towards systematic interaction with external tools
  (via Scala/JVM)

Isabelle/ML

The LCF approach

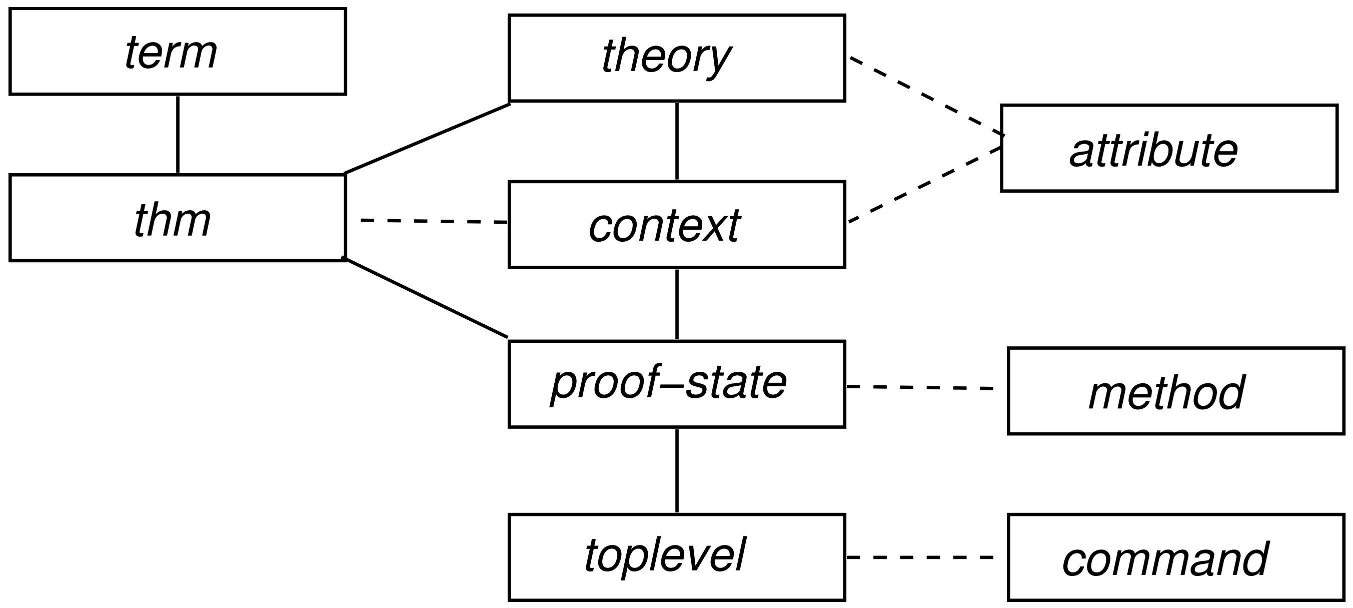
Design principles of the LCF system (Milner 1979):

- Implementation language: LISP
- Meta-language: ML
- Object-language: the logic, accessible via abstract types in ML ("correctness-by-construction")

Later variations on LCF:

1. HOLs: ML as implementation language
2. Coq: separate language for definitions and proofs
3. Isabelle: generic "framework"
   (a) logical framework
   (b) system framework ("logical operating system")

Formal entities in Isabelle/ML



basic values: *term*, *thm*

state information: *theory*, *context*, *proof-state*, *toplevel*

main operations: *attribute*, *method*, *command*

Isabelle source language layers

1. Isar implemented in ML
2. ML embedded into Isar
3. Logical entities embedded into ML via antiquotations

Basic notation:

- quote: **ML** $\langle\langle \ldots \rangle\rangle$
- antiquote: $@\{name\ args\}$

Examples:

ML $\langle\langle$ *val Ps: term list = Thm.prems-of* $@\{thm\ mp\}$ $\rangle\rangle$
ML $\langle\langle$ *val thm =* $@\{lemma\ A \Longrightarrow B \Longrightarrow A\ by\ assumption\}$ $\rangle\rangle$

User-space system extensions (in ML)

- (semi)automated proof tools
  e.g. *induct*, *cases*, *simp*, *blast*, *auto*, *metis*
- specification mechanisms ("definitional packages")
  e.g. **inductive**, **datatype**, **primrec**, **function**
- module concepts, e.g. **locale**, **class**
- code generators, e.g. Haskell, SML, OCaml
- etc.

Note: shipment as precompiled "logic images" e.g. Isabelle/HOL, Isabelle/HOL-Nominal, Isabelle/HOLCF

Some Isabelle/ML system infrastructure

- Universal *context* (logical, extra-logical, execution context)
- Managed evaluation and parallel programming
  (efficient *future values* as ML library, for 4–8 cores)
- ML threads with high-level concurrency library
- system shell-out that works with threads and signals

- *Missing:* “mainstream libraries”
  for XML, TCP/IP services, web programming, PDF rendering

Isabelle/Scala

Scala — http://www.scala-lang.org

What is Scala anyway?

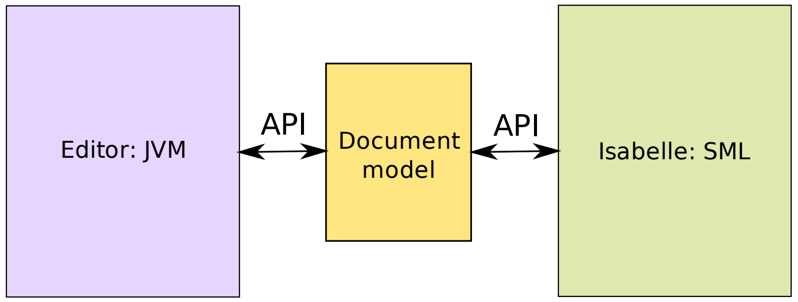
- *The Next Big Thing* in the JVM world
- Nice integration of the best of
  - object-oriented programming (many steps beyond Java)
  - higher-order functional programming (many improvements over ML and Haskell, despite some compromises)
- Native support for "domain specific languages"

Isabelle/Scala:

- Integral part of Isabelle/Pure sources, `.ML` and `.scala` side-by-side
- basic Isabelle/System services (e.g. math symbols, YXML markup)
- robust Isabelle process management
- editor-oriented document model (replaces TTY)

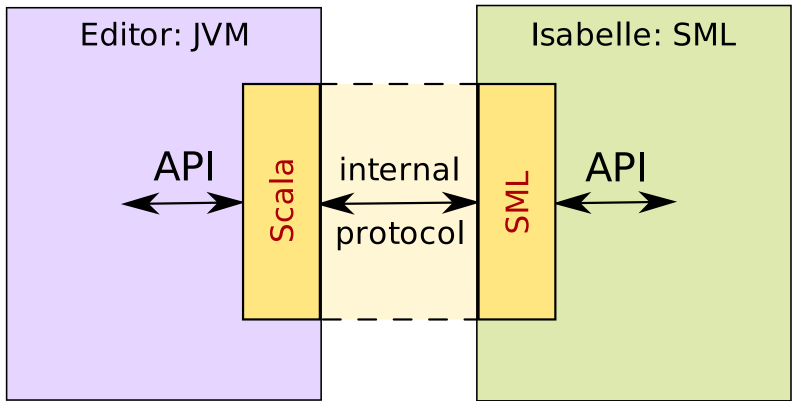
Emerging interface architecture

Conceptual view:



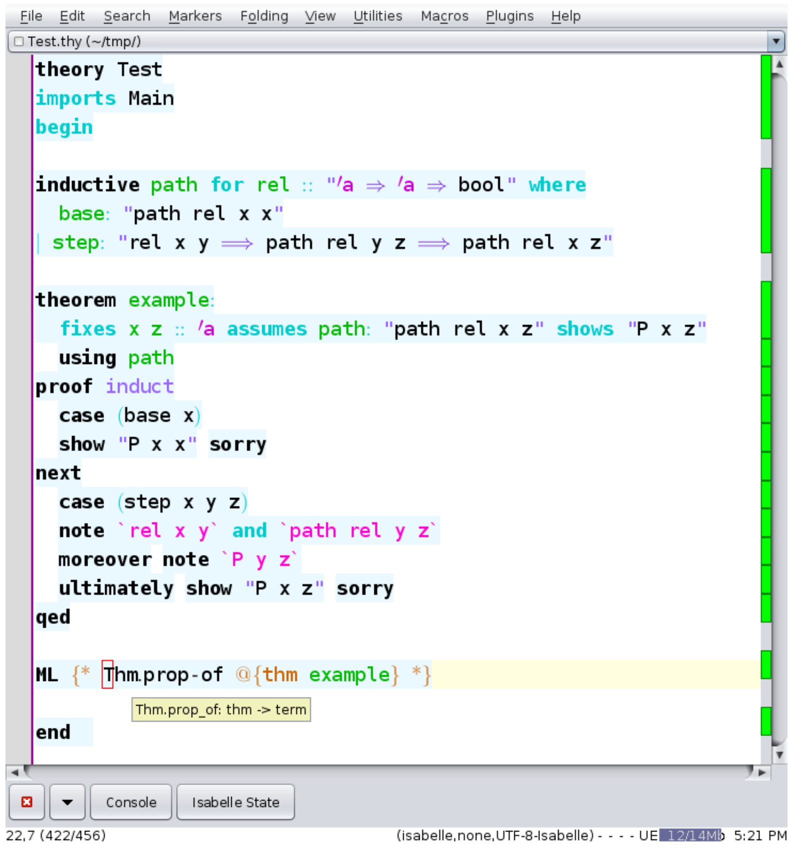
- bridge SML — Scala/JVM
- support GUIs, IDEs, application servers etc.
- advanced document model: parallel checking, asynchronous interaction

Implementation view:



- public API, private protocol
- integral part of future Isabelle distributions

Application: Isabelle/jEdit (under construction)


File Edit Search Markers Folding View Utilities Macros Plugins Help
Test.thy (~/tmp/)

```
theory Test
imports Main
begin

inductive path for rel :: "'a ⇒ 'a ⇒ bool" where
  base: "path rel x x"
| step: "rel x y ⟹ path rel y z ⟹ path rel x z"

theorem example:
  fixes x z :: 'a assumes path: "path rel x z" shows "P x z"
  using path
proof induct
  case (base x)
  show "P x x" sorry
next
  case (step x y z)
  note `rel x y` and `path rel y z`
  moreover note `P y z`
  ultimately show "P x z" sorry
qed

ML {* Thm.prop-of @{thm example} *}
        Thm.prop_of: thm -> term
end
```

- jEdit plugin written in Scala
- “IDE” both for Isar and ML
- free editing
- continous checking
- visual feedback

Further possibilities

- Asynchronous assistants for interactive proof development
  for example:
  - auto quickcheck
  - auto refute
  - auto nitpick
  - auto sledgehammer
- Callbacks from ML into JVM
  for example:
  - Nitpick (Isabelle/ML) invokes Kodkod (inside Isabelle/Scala)
  - some Isar proof method *foobar* calls proof tool `class Foobar`

Conclusion

Summary

Isabelle/ML: (based on Poly/ML)

- well-defined, well-understood
- very efficient (except for 32-bit words and floats)
- explicit and implicit parallelism
- tight integration with Isar and logical languages

Isabelle/Scala:

- reasonably efficient
- culturally similar to ML
- emphasizes immutability and concurrency
- access to mainstream libraries and frameworks (JVM)
- robust Isabelle system integration for future tool environments