Interactive Proof Documents
Theorem Provers for User Interfaces

Makarius

November 2008
Motivation
Aims

• Renovate and reform traditional “LCF-style” theorem proving for coming generations of users.
• Catch up with technological shifts, e.g. advanced user-interfaces, parallel computing.
• Support novel models for interactive proof checking.
Proof Documents
**Documents**

**Classical structure:** definition — statement — proof

**Example:** Isabelle/Isar

```plaintext
datatype foo = Foo | Bar foo

lemma fixes x :: foo shows P x
proof (induct x)
  case Foo
  then show P Foo ⟨proof⟩
next
  case (Bar x)
  note ⟨P x⟩
  then show P (Bar x) ⟨proof⟩
qed
```

Proof Documents
Scripts (1)

datatype foo = Foo | Bar foo  
lemma fixes x :: foo shows "P x"
proof (induct x)
case Foo
then
show "P Foo"
sorry
next
case (Bar x)
note ‘P x’
then
show "P (Bar x)"
sorry
qed
Scripts (2)

Problems with proofs scripts:

- No structure.
- Inefficient checking.
- Prover policies enforce bad habits.
Proof document structure

**General interactive provers:**
1. definitions / statements: sequential dependency
2. proofs: irrelevant $\rightarrow$ independent $\rightarrow$ parallel checking

**Isabelle/Isar:**
0. theories: graph structured (DAG)
1. definitions / statements: sequential
2. toplevel proofs: parallel
3. local proofs: hierarchical (tree)
Intermission: parallel proof checking


2. Value-oriented parallel computations (summer 2008)
   - type 'a future
   - fork: (unit -> 'a) -> 'a future
   - join: 'a future -> 'a
   - cancel: 'a future -> unit

3. Proof objects with holes (last week)
   - promise $a[\bar{x}]: \varphi$ where $FV \varphi = \{\}$ and $TV \varphi = \bar{x}$
   - fulfill $a = p$

4. Extended “LCF” inference kernel (fall 2008)
   - Thm.future: (unit -> thm) -> term -> thm
   - Thm.force_proof: thm -> unit

5. Extended Isar/VM (“goal package”) (summer 2008)

6. Extended Isabelle/Isar toplevel and interaction protocol
Interface Architecture
Editor versus Prover

Editor \rightarrow Prover

Interface Architecture
Example: Java IDE

Characteristics:

- Conceptually simple — no rocket science.
- It works well — mainstream technology.
- Provers are not implemented in Java!
- Even with Scala/JVM, the JVM is not ideal for LCF-style provers.
Example: Coq IDE

Characteristics:

+ Conceptually simple — no rocket science.
+ It works . . . mostly.
  - Many Coq power-users ignore it.
  - GTK/OCaml is a niche market; GTK/SML is unavailable.
− Need to duplicate editor implementation efforts.
**Mixed platforms**

**Realistic assumption:**
- Prover: SML (or OCaml or Haskell)
- Editor: Java/JVM (or . . . )

**Big problem:** How to integrate the two worlds?
- Separate processes: requires marshalling, serialization, protocols.
- Different implementation languages and programming paradigms.
- Different cultural backgrounds!
Example: PGIP framework

Characteristics:
+ Very general architecture of distributed components.
  – Significant effort for protocol definition.
  – Significant effort for implementation, integration, and maintenance.

Note: fundamental difference of API vs. protocol
Future Isabelle/Isar architecture (1)

Conceptual view:

Implementation view:
Future Isabelle/Isar architecture (2)

Characteristics:
+ Regular API, based on internal protocol.
+ Supports mixed environments: Scala/JVM vs. SML.
+ Conceptual advances in proof document model:
  parallel checking, asynchronous interaction.
  - Significant effort for design and initial implementation.
  - Provers need to be adapted to interface needs.
  - Biased towards particular platforms.
+ Focussed on particular platforms.
Actual Implementation
Isabelle process wrapper

- Type: JVM library implemented in Scala
- Features:
  - process management
  - message model
  - XML / YXML transfer
  - Isabelle symbol recoding
- Developed as integral part of Isabelle
  see ~/lib/classes/Pure.jar and ~/src/Pure
Isabelle / Netbeans

- Type: standalone Java application within the Netbeans framework
- Features:
  - basic “Proof General” functionality
  - uses old version of Isabelle process wrapper
  - implements first version of IAPP document model
- Developed by Holger Gast (Universität Tübingen) within a few weeks (including learning Netbeans)
Isabelle / jEdit

- Type: plugin written in Scala, for Java-based editor framework
- Features:
  - basic support for forthcoming proof document model
  - discontinues typical “Proof General” interaction
- Developed by Johannes Hölzl (TU München) within several weeks (undergraduate programming project)
Reasonable technologies

- Scala/JVM, not Java
- potentially Scala/.NET
- Swing
- abstract XML trees, not DOM etc.
- XML/XHTML rendering with CSS 2 (JavaDesktop)
- PDF rendering (JavaDesktop)
- jEdit (JavaDesktop)
- Netbeans framework
Demo?