

# The languages of Isabelle: Isar, ML, and Scala

Makarius Wenzel  
TU München

September 2009



# Abstract

More than 20 years ago, Isabelle was introduced by Larry Paulson as a "logical framework", to facilitate experimentation with various calculi according to general ideas of higher-order Natural Deduction. Over time the system has widened its scope, to become a general platform for building applications based on formal logic, with fully formal proof checking by a trusted kernel. We give an overview of the Isabelle platform of 2009 from the perspective of the main languages involved: Isar, Standard ML, and Scala/JVM.

Isabelle/Isar is presented to end-users as a language for human-readable proofs (and specifications). It is firmly rooted on the Pure logical framework, and imposes certain policies on core inferences by means of rich extra-logical infrastructure. Thus Isar enhances the original framework of primitive proof rules towards one of structured proof texts. The concrete proof language can be seen as an application of more general principles provided internally: the greater part of the Isar language is implemented as derived elements. Further "domain specific proof languages" can be implemented as well.

Isabelle/ML is both the implementation language and extension language of the

framework. Isabelle follows the original "LCF-approach" (Robin Milner, 1979). This means that the key notions are modeled as abstract datatypes in ML, and users may implement extensions on top without affecting soundness. Isabelle/ML is embedded into the Isar source language such that the proof context is available at compile time. Antiquotations within ML source allow robust references to formal entities. Using Poly/ML, the baseline performance for typical symbolic computations is very good; recent moves towards multicore support improve upon this further. In Isabelle2009, both theories and proofs are scheduled in parallel by the system, with reasonable scalability for current home machines (4-8 cores).

Isabelle/Scala has been recently added as a wrapper around the Isabelle/Isar/ML process. The idea is to reach out into the "real world", as represented by the JVM platform with its existing facilities for user interfaces, web services etc. Thus we overcome the traditional text-based interaction with the raw ML process, replacing it by an editor-oriented proof document model with rich markup provided by the prover in the background. The existing concepts of parallel proof checking are eventually generalized towards an asynchronous interaction model that is expected to increase end-user productivity significantly. This general API for "Isabelle system programming" in Scala has already been used for ongoing implementation of Isabelle/jEdit.

**Isabelle/Isar**

# Isabelle/Pure framework (Paulson 1989)

**Logical framework:** 3 levels of  $\lambda$ -calculus

$\alpha \Rightarrow \beta$       terms depending on terms  
 $\bigwedge x. B \ x$     proofs depending on terms  
 $A \Longrightarrow B$     proofs depending on proofs

**Rule composition:** via higher-order unification

*resolution:* mixed forward-back chaining

*assumption:* closing branches

**Note:** arbitrary nesting of rules

## Example: Natural Deduction rules

$$\frac{A \quad B}{A \wedge B}$$

$$A \implies B \implies A \wedge B$$

$$\frac{\begin{array}{c} [A] \\ \vdots \\ B \end{array}}{A \rightarrow B}$$

$$(A \implies B) \implies A \rightarrow B$$

$$\frac{\begin{array}{c} [n][P \ n] \\ \vdots \\ P \ 0 \quad P \ (Suc \ n) \end{array}}{P \ n}$$

$$P \ 0 \implies (\bigwedge n. P \ n \implies P \ (Suc \ n)) \implies P \ n$$

# Isabelle/Isar proof language (Wenzel 1999)

**Main idea:** Pure rules turned into proof schemes

**from** *facts*<sub>1</sub> **have** *props* **using** *facts*<sub>2</sub>  
**proof** (*resolution rule*)  
  *body*  
**qed** *assumption*\*

**Solving sub-problems:** within *body*

**fix** *vars*  
**assume** *props*  
**show** *props* *<proof>*

**Abbreviations:**

**then**   ≡   **from** *this*  
  **..**    ≡   **proof** **qed**

## Example: Natural Deduction proofs

**have**  $A$  **and**  $B$   $\langle proof \rangle$   
**then have**  $A \wedge B$  ..

**have**  $A \rightarrow B$   
**proof**  
  **assume**  $A$   
  **show**  $B$   $\langle proof \rangle$   
**qed**

**fix**  $n :: nat$   
**have**  $P n$   
**proof** (*induct*  $n$ )  
  **show**  $P 0$   $\langle proof \rangle$   
  **fix**  $n$  **assume**  $P n$   
  **show**  $P (Suc\ n)$   $\langle proof \rangle$   
**qed**

# Application: inductive definitions and proofs

**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* ⟨*proof*⟩

**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* ⟨*proof*⟩

**qed**

## Application: calculational proofs

**also**<sub>0</sub> = **note** *calculation = this*  
**also**<sub>*n*+1</sub> = **note** *calculation = trans [OF calculation this]*  
**finally** = **also from** *calculation*

### Example:

**have**  $a = b$  *<proof>*  
**also have**  $b = c$  *<proof>*  
**also have**  $c = d$  *<proof>*  
**finally have**  $a = d$  .

### Notes:

- **derived** language elements
- Isar admits “domain specific” proof languages

# Isar language characteristics

- interpreted language of “proof expressions”
    - proof context
    - flow of facts towards goals
    - simple reduction to Pure logic
  - language framework
    - highly structured
    - highly extensible
    - non-computational
- language for **proofs**, not proof procedures

**Isabelle/ML**

# The LCF approach

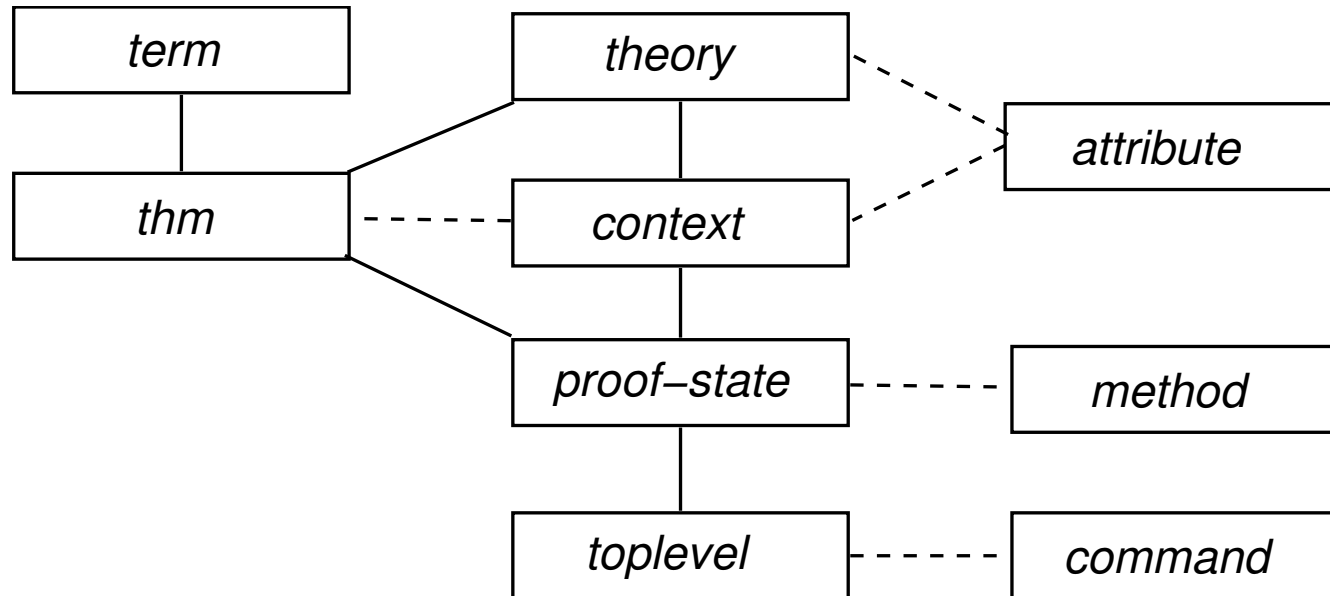
Design principles of the LCF system (Milner 1979):

- 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**  $\langle\langle \dots \rangle\rangle$
- antiquote:  $@\{name\ args\}$

## Examples:

**ML**  $\langle\langle val Ps: term list = Thm.premis-of @\{thm mp\} \rangle\rangle$

**ML**  $\langle\langle val thm = @\{lemma A \implies B \implies A\ by\ assumption\} \rangle\rangle$

## Example: ML with antiquotations

```
ML << val goal = Goal.init @{cprop A ∧ B → B ∧ A} >>
```

```
ML <<  
  val results = goal |>  
  (rtac @{thm impI} 1 THEN  
   etac @{thm conjE} 1 THEN  
   rtac @{thm conjI} 1 THEN  
   atac 1 THEN  
   atac 1) >>
```

```
ML <<  
  val thm =  
    (case Seq.pull results of  
     NONE => error Proof failed  
     | SOME (result, -) => Goal.finish result) >>
```

# Example: ML generated from HOL (Florian Haftmann)

**datatype** *form* = *T* | *F* | *And form form* | *Or form form*

**ML**  $\langle\langle$

```
fun eval-form @{code T} = true  
  | eval-form @{code F} = false  
  | eval-form (@{code And} (p, q)) =  
    eval-form p andalso eval-form q  
  | eval-form (@{code Or} (p, q)) =  
    eval-form p orelse eval-form q;
```

```
eval-form (@{code And} (@{code T}, @{code T}))  
 $\rangle\rangle$ 
```

# Intermission: parallel proof checking

**Question:** Who survives the multicore crisis?

## **LCF-style theorem provers:**

- full programmability (in ML)
  - fully general parallelization problem
- explicit proof construction
  - requires significant runtime resources
- practical proof irrelevance
  - great potential for **implicit parallelism**

# Status of parallelization efforts

## **Poly/ML 5.2.1** (by David Matthews)

- native POSIX threads in ML
- sequential garbage collection (typically 5–10%)

## **Isabelle2009:**

- value-oriented parallelism in ML (futures)
- parallel inference kernel
- parallel scheduling of theories and proofs
- performance: 4 cores (factor 3.0), 8 cores (factor 4.5)

## **Ongoing work:**

- improved scalability (16 cores in August 2009)
- parallel and distributed GC in Poly/ML (!?)

**Isabelle/Scala**

# 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:

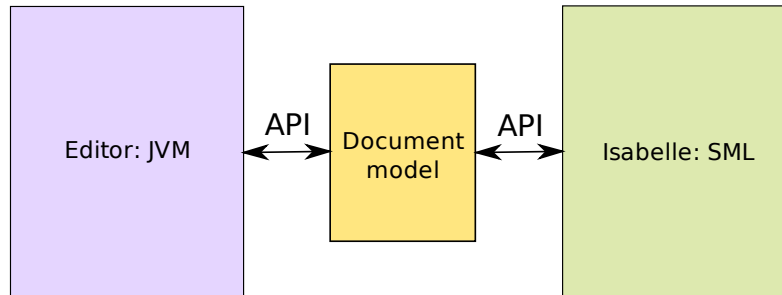
- web client, based on server-side prover component
- powerful proof editor, or “Prover IDE”
- advanced document preparation, with rich semantic information

# Layers for Isabelle system programming

1. ML compiler and runtime system —  
Standard ML with tight integration into `lsar`
2. Posix system glue based on `bash` and `perl` —  
works uniformly on Linux, Mac OS, Windows (via Cygwin)
3. Scala/JVM wrapping — platform independent `.jar`
  - integral part of Isabelle sources, `.ML` and `.scala` side-by-side
  - Isabelle/ML conventions carry over to Isabelle/Scala  
(forExampleWeDoNotUseMixedCaseIdentifiers)
  - some library modules:
    - robust Isabelle process management
    - mathematical symbols
    - structured result messages (YXML markup)
    - editor-oriented document model

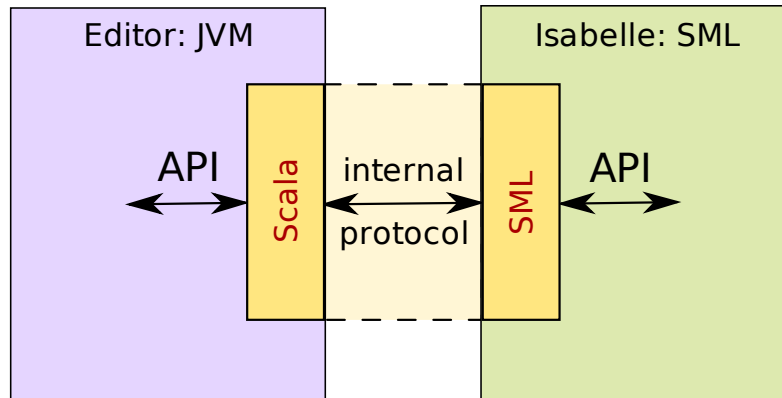
# Application: emerging interface architecture

## Conceptual view:



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

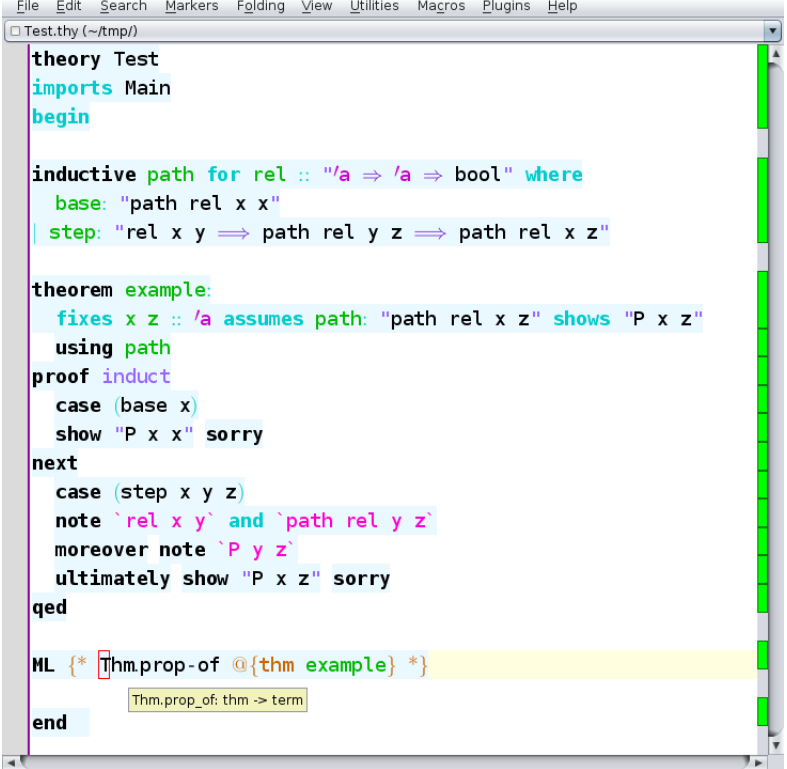
## Implementation view:



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

# Application: Isabelle/jEdit (under construction)

- jEdit plugin written in Scala
- “IDE” both for Isar and ML
- discontinues “locked region” of Proof General
- asynchronous proof document model (parallel proof checking)



```
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
```

22.7 (422/456) (isabelle:none,UTF-8-Isabelle) - - - UE 1.2/1.4Mb 5:21 PM

# Conclusion

# Language summary

## Isabelle/Isar:

- framework for domain specific proof languages
- rich logical and extra-logical infrastructure
- implicit parallelism

## Isabelle/ML:

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

## Isabelle/Scala:

- reasonably efficient
- access to mainstream libraries and frameworks
- robust Isabelle system integration for future tool environments