

# **Asynchronous processing of proof documents – rethinking interactive theorem proving**

Makarius

November 2007

1. Motivation
2. Document processing
3. Main agents: provers, editors, users

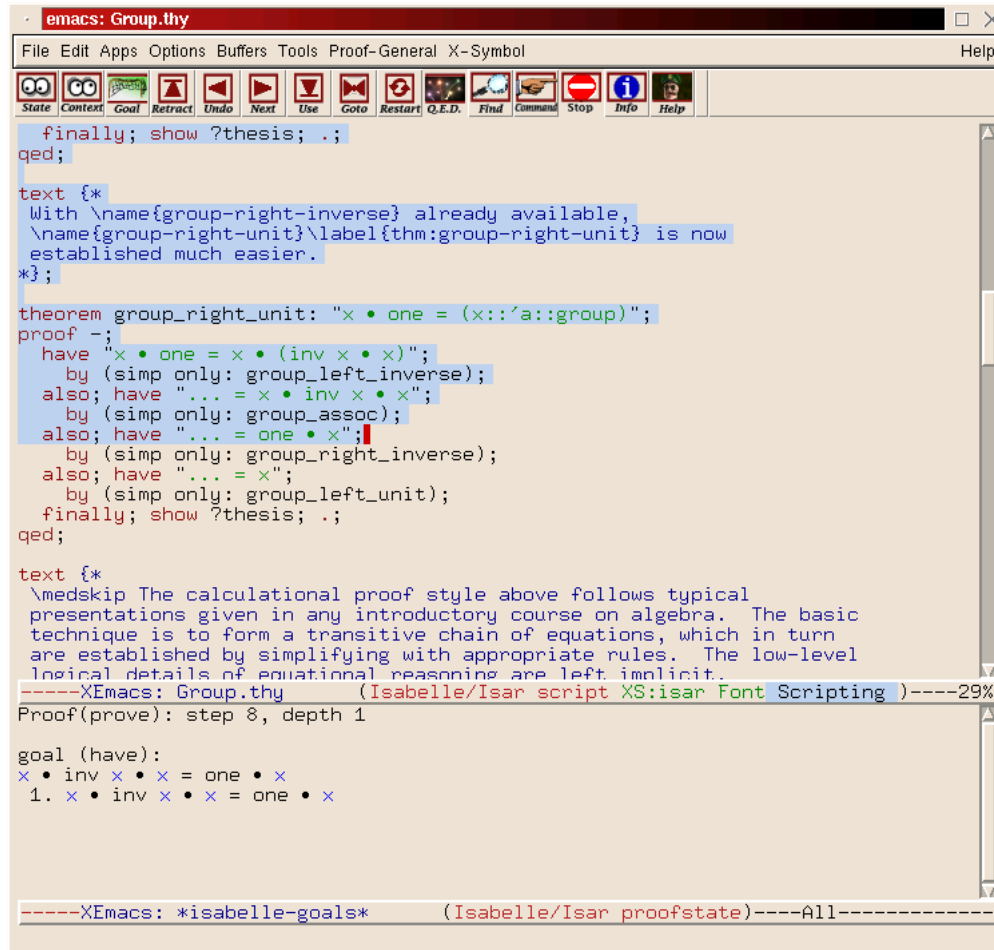
# Motivation

## General aims

- Support interactive development of larger formal theories
- Reduce requirements on front-end (editors, web clients etc.)
- Exploit parallel proof checking  
(multiprocessing is the elephant in the room)
- Exploit inherent structure of documents  
(implicit: proof irrelevance, explicit: Isar)

—→ Towards the next generation of interactive proof checking

# Example: Proof General



```
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)

## Example: Mizar

```
for A,B,C being set holds
  A c= B implies A /\ C c= B /\ C
proof
  let A, B, C be set;
  assume subset: A c= A;
  ::> *52
  thus A /\ C c= B /\ C
  proof
    let x be set;
    assume a1: x in A /\ C;
    then x in A;
  ::> *4
  end;
  ::> *70
end;

::> 4: This inference is not accepted
::> 52: Invalid assumption
::> 70: Something remains to be proved
```

Main characteristics:

- simultaneous checking of *proof text*
- always fully checked, potential omissions
- no proof state
- inline response
- mostly “batchmode”

# **Proof document processing**

# Isabelle language layers

**Primitive layer:** logic implementation

- Isabelle/Pure logical framework (as LCF-style kernel)
- Isabelle/ZF, HOL, HOLCF, . . . object-logics

**Primary layer:** Isabelle/Isar theory and proof language

**Presentation layer:**  $\text{\LaTeX}$  generated from formal theory sources

Example presentations:

- $\rightarrow$  This  $\leftarrow$  (slides)
- <http://isabelle.in.tum.de/dist/library/HOL/Unix/document.pdf>  
(proof document)

# The primary “document” model

Fundamental entities:

- *printed document*: result of processing a *session*
- *session*: graph of *theory* nodes
- *theory*: sequence of *commands* (transactions)
- *command*: theory specification element (definition, statement), or proof element etc.

Existing technology:

- sequential processing of command transactions
- synchronous reporting of success / error
- single *undo* / *redo*

Note 1: Relative state addressing, expressed as unary offsets!

Note 2: Proof General only uses *undo*.



# Example

```
datatype foo = Foo | Bar foo
```

```
lemma
```

```
  fixes x :: foo
```

```
  shows P x
```

```
proof (induct x)
```

```
  case Foo
```

```
  then show P Foo  $\langle$ proof $\rangle$ 
```

```
next
```

```
  case (Bar x)
```

```
  note  $\langle$ P x $\rangle$ 
```

```
  then show P (Bar x)  $\langle$ proof $\rangle$ 
```

```
qed
```

## Basic observations

- Checking specifications can take considerable time, but the result is determined syntactically.
- Checking proofs takes 95% of the time, but proofs are irrelevant (in Isabelle/Pure).
- Checking terminal justifications takes 95% of proof time, but Isar structure does not really care.

# Principles of “asynchronous” proof processing

- Commands (transactions) are explicitly identified (unique labels)
- States (after successful transactions) may be addressed explicitly
- Structural dependencies are observed, e.g.
  - sequential composition of consecutive transactions (default)
  - parallel composition of independent branches
  - nesting due to logical block structure
- Fine-grained result state of transactions, e.g.
  - *unprocessed*
  - *syntax-checked*
  - *proof-checked*
- Dynamic message model, e.g. progress reports

## Example: irrelevant proofs

**lemma** *[simp]: attributes (Val (att, text)) = att*

**by** *(simp add: attributes-def)*

**lemma** *[simp]: attributes (Env att dir) = att*

**by** *(simp add: attributes-def)*

**lemma** *[simp]: attributes (map-attributes f file) = f (attributes file)*

**by** *(cases file) (simp-all add: attributes-def map-attributes-def split-tupled-all)*

**lemma** *[simp]: map-attributes f (Val (att, text)) = Val (f att, text)*

**by** *(simp add: map-attributes-def)*

**lemma** *[simp]: map-attributes f (Env att dir) = Env (f att) dir*

**by** *(simp add: map-attributes-def)*

# Example: derived specifications

## inductive

$transition :: file \Rightarrow operation \Rightarrow file \Rightarrow bool$   
 $(- \dashrightarrow - [90, 1000, 90] 100)$

## where

*read:*

$access\ root\ path\ uid\ \{Readable\} = Some\ (Val\ (att,\ text)) \implies$   
 $root\ -(Read\ uid\ text\ path) \rightarrow root\ |$

*write:*

$access\ root\ path\ uid\ \{Writable\} = Some\ (Val\ (att,\ text')) \implies$   
 $root\ -(Write\ uid\ text\ path) \rightarrow update\ path\ (Some\ (Val\ (att,\ text)))\ root\ |$

*chmod:*

$access\ root\ path\ uid\ \{\} = Some\ file \implies$   
 $uid = 0 \vee uid = owner\ (attributes\ file) \implies$   
 $root\ -(Chmod\ uid\ perms\ path) \rightarrow update\ path$   
 $(Some\ (map-attributes\ (others-update\ (K-record\ perms))\ file))\ root\ |$

...

$\langle monotonicity\ proof \rangle$

$\langle main\ proof \rangle$

# Example: sub-structured proofs

**theorem** *transition-uniq:*

**assumes**  $root'$ :  $root -x \rightarrow root'$  **and**  $root''$ :  $root -x \rightarrow root''$

**shows**  $root' = root''$  **using**  $root''$

```
proof cases
  case read
    with  $root'$  show ?thesis by cases auto
next
  case write
    with  $root'$  show ?thesis by cases auto
next
  case chmod
    with  $root'$  show ?thesis by cases auto
next
  ...
qed
```

**Main agents: provers, editors, users**

# Provers

- Attempt to cover broad range of existing provers: Isabelle, Mizar, Coq, Matita, etc.
- Define general principles, but do not set particular features in stone
- Implementation options:
  1. full version: native support of asynchronous checking (including parallel processing etc.)
  2. restricted version: fit unchanged systems into the model
  3. mixed version: additional support by “middle ware”



# Editors

- Open-mindedness to cover broad range of editing environments: web interfaces, Emacs, jEdit, etc.
- Down-scaled demands on specific features:
  - No locking of text regions (only highlighting)
  - Undo/redo follows editor view, not prover
  - Even less “structure editing”
  - Replace responses (warnings, errors) by in-text annotation
  - Abolish proof state buffer!?
- Convergence of exiting efforts on building post-Proof-General interfaces?

# Users

- User types: address beginners and experts alike
- User empowerment: more freedom in out-of-order editing, top-down development, composing outlines, multiple views
- User groups: support collaborative editing (How?)

Further issues:

- Generalizing multi-threaded / multi-viewed document processing towards multi-user processing.
- Integration with (centralized or distributed) repositories.