

Asynchronous User Interaction and Tool Integration in Isabelle/PIDE

Makarius Wenzel
Univ. Paris-Sud, LRI

July 2014



Project **Paral-ITP**
ANR-11-INSE-001

Introduction

Motivation

General aims:

- renovate and reform interactive (and automated) theorem proving for new generations of users
- address paradigm shifts: multicore and pervasive parallelism
- document-oriented user interaction and tool integration

Motivation

General aims:

- renovate and reform interactive (and automated) theorem proving for new generations of users
- address paradigm shifts: multicore and pervasive parallelism
- document-oriented user interaction and tool integration

Ultimate challenge:

Introducing genuine interaction into ITP

- many conceptual problems
- many technical problems
- many social problems

TTY loop (\approx 1979)

```
Terminal
File Edit View Terminal Tabs Help
Welcome to Isabelle/HOL (Isabelle2013: February 2013)
> theory A imports Main begin
theory A
> lemma "x = x";
proof (prove): step 0

goal (1 subgoal):
  1. x = x
> █

Terminal
File Edit View Terminal Tabs Help
Welcome to Coq 8.4pl2 (September 2013)

Coq < Lemma test: forall (A: Type) (x: A), x = x .
1 subgoal

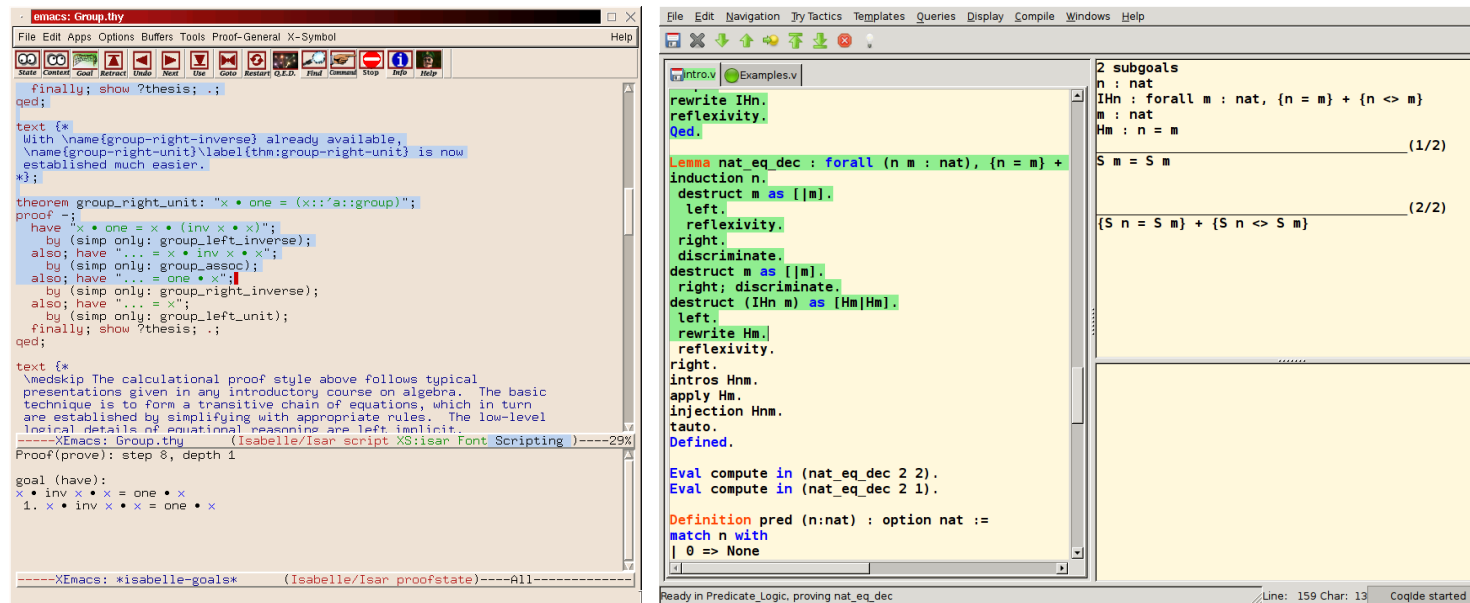
=====
forall (A : Type) (x : A), x = x
test < █
```



(Wikipedia: K. Thompson and D. Ritchie at PDP-11)

- user drives prover, via **manual copy-paste**
- inherently **synchronous and sequential**

Proof General and clones (\approx 1999)



- user drives prover, via automated copy-paste and undo
- inherently **synchronous and sequential**

PIDE: Prover IDE (\approx 2009)

Approach:

Prover supports asynchronous **document model** natively

Editor continuously sends source **edits** and receives markup **reports**

Tools may **participate** in document processing and markup

User constructs document content — assisted by
GUI rendering of cumulative **PIDE markup**

PIDE: Prover IDE (\approx 2009)

Approach:

Prover supports asynchronous **document model** natively

Editor continuously sends source **edits** and receives markup **reports**

Tools may **participate** in document processing and markup

User constructs document content — assisted by
GUI rendering of cumulative **PIDE markup**

PIDE applications:

Isabelle/jEdit the default user-interface of Isabelle

Isabelle/Eclipse by Andrius Velykis (for Isabelle2013)

<https://github.com/andriusvelykis/isabelle-eclipse>

Isabelle/Clide by Martin Ring and Christoph Lüth (subsequent talk)

<https://github.com/martinring/clide>

Isabelle/jEdit Prover IDE (2014)

```
header {* Finite sequences *}

theory Seq
imports Main
begin

datatype 'a seq = Empty | Seq 'a "'a seq"

fun conc :: "'a seq => 'a seq => 'a seq"
where
  "conc Empty ys = ys"
| "conc (Seq x xs) ys = Seq x (conc xs ys)"

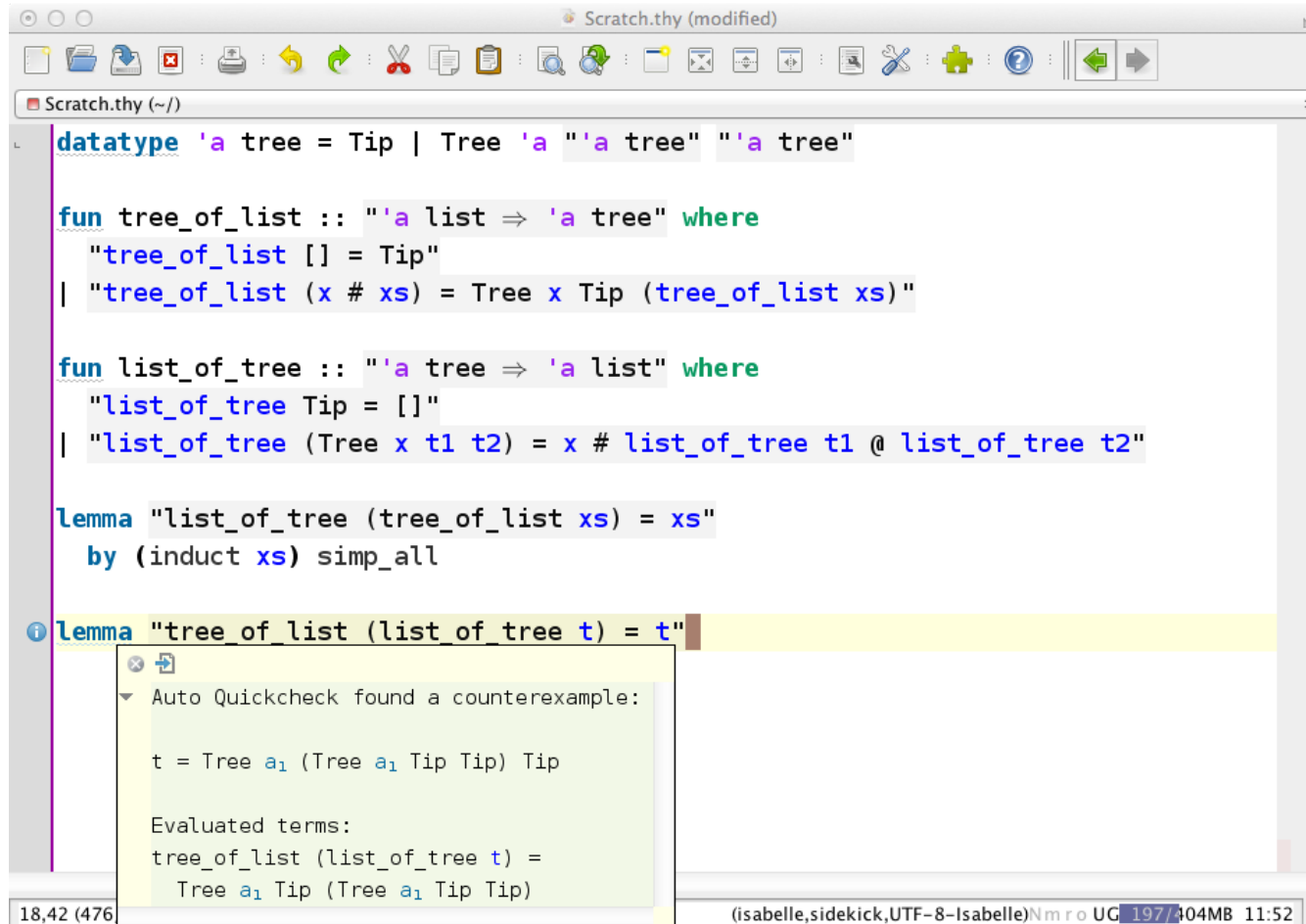
fun reverse
where
  "reverse Empty = Empty"
| "reverse (Seq x xs) = conc (reverse xs) (Seq x Empty)"

lemma conc_empty: "conc xs Empty = xs"
by (induct xs) simp_all

constants
  conc :: "'a seq => 'a seq => 'a seq"
Found termination order: "(λp. size (fst p)) < *mlex* > {}"
```

13,39 (203/791) (isabelle,sidekick,UTF-8-Isabelle)Nmr o UC 162 353MB 11:05

Automatically tried tools (2014)



The screenshot shows a window titled "Scratch.thy (modified)" with a toolbar and a text editor. The text editor contains the following code:

```
datatype 'a tree = Tip | Tree 'a "'a tree" "'a tree"

fun tree_of_list :: "'a list ⇒ 'a tree" where
  "tree_of_list [] = Tip"
| "tree_of_list (x # xs) = Tree x Tip (tree_of_list xs)"

fun list_of_tree :: "'a tree ⇒ 'a list" where
  "list_of_tree Tip = []"
| "list_of_tree (Tree x t1 t2) = x # list_of_tree t1 @ list_of_tree t2"

lemma "list_of_tree (tree_of_list xs) = xs"
  by (induct xs) simp_all

lemma "tree_of_list (list_of_tree t) = t"
```

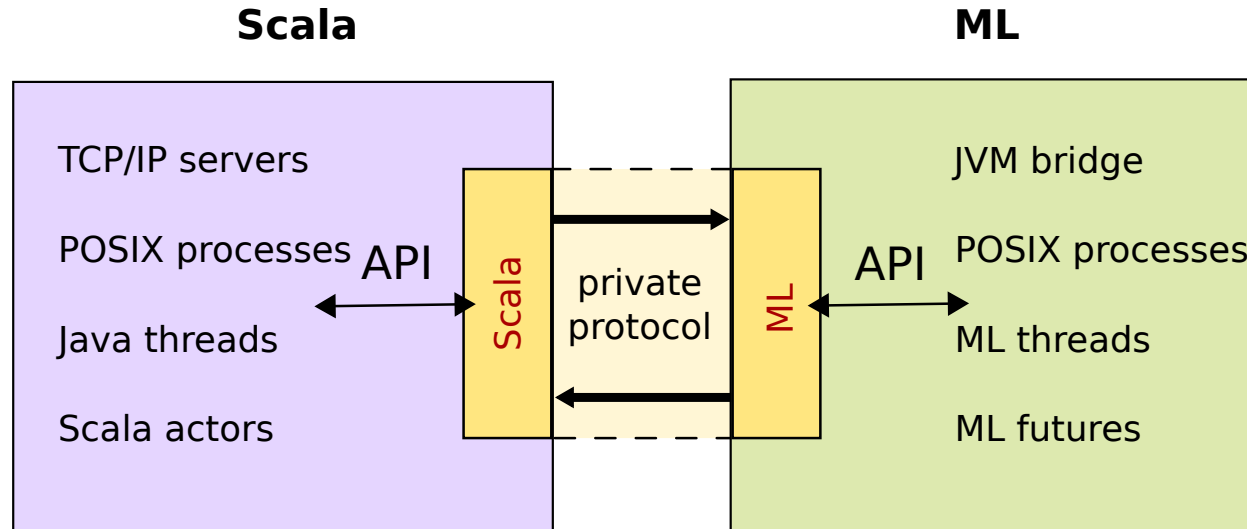
The last lemma is highlighted in yellow. A tooltip is visible over it, containing the following text:

Auto Quickcheck found a counterexample:
 $t = \text{Tree } a_1 (\text{Tree } a_1 \text{ Tip Tip}) \text{ Tip}$
Evaluated terms:
 $\text{tree_of_list (list_of_tree } t) =$
 $\text{Tree } a_1 \text{ Tip (Tree } a_1 \text{ Tip Tip)}$

The status bar at the bottom shows "18,42 (476)" on the left and "(isabelle,sidekick,UTF-8-Isabelle)Nm r o UG 197/104MB 11:52" on the right.

PIDE architecture

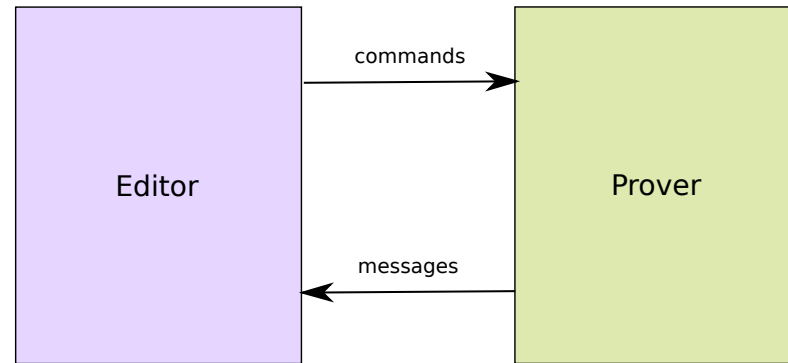
The connectivity problem



Design principles:

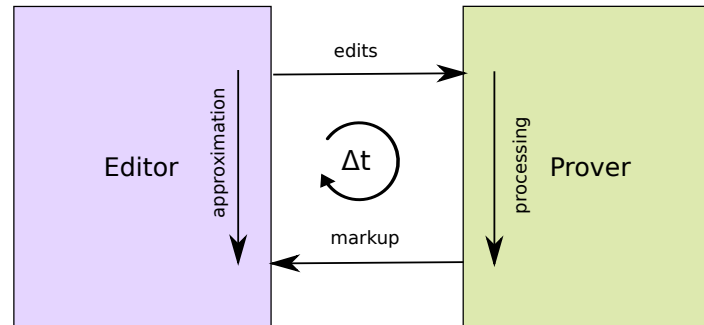
- **private** protocol for prover connectivity
(asynchronous interaction, parallel evaluation)
- **public** Scala API
(timeless, stateless, static typing)

PIDE protocol functions



- `type protocol_command = name -> input -> unit`
 - `type protocol_message = name -> output -> unit`
 - **outermost state** of protocol handlers on each side (pure values)
 - **asynchronous streaming** in each direction
- editor and prover as **stream-processing functions**

Approximative rendering of document snapshots



1. editor knows text T , markup M , and edits ΔT (produced by user)
2. apply edits: $T' = T + \Delta T$ (**immediately** in editor)
3. formal processing of T' : ΔM after time Δt (**eventually** in prover)
4. temporary approximation (**immediately** in editor):
 $\tilde{M} = \text{revert } \Delta T; \text{ retrieve } M; \text{ convert } \Delta T$
5. convergence after time Δt (**eventually** in editor):
 $M' = M + \Delta M$

Document content

Prover command transactions

- “small” toplevel state st : *Toplevel.state*
- command transaction tr as partial function over st
we write $st_0 \longrightarrow^{tr} st_1$ for $st_1 = tr\ st_0$
- general structure: $tr = read; eval; print$

Interaction view:

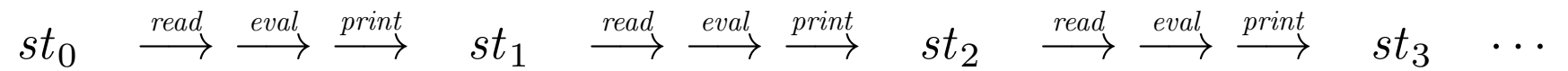
$tr\ st_0 =$

let $eval = read\ ()$ **in** — $read$ does not require st_0
let $st_1 = eval\ st_0$ **in** — main transition $st_0 \longrightarrow st_1$
let $() = print\ st_1$ **in** st_1 — $print$ does not change st_1

Important: purely functional transactions with managed output

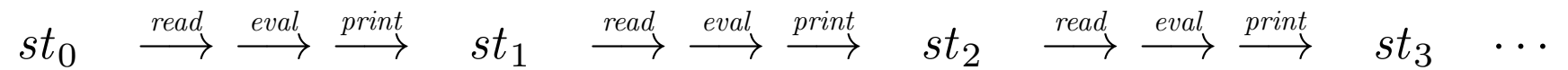
Command scheduling

Sequential R-E-P Loop:

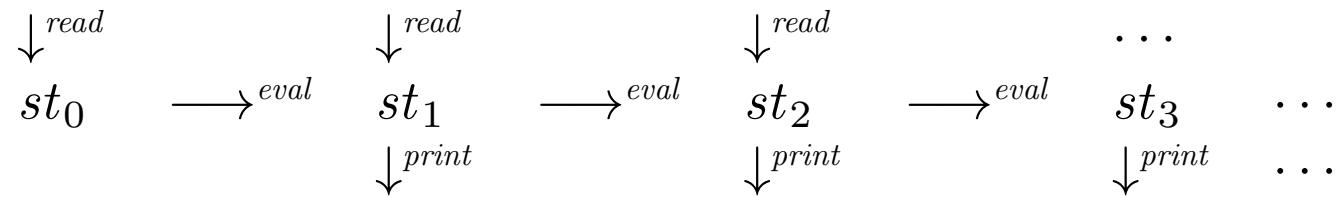


Command scheduling

Sequential R-E-P Loop:

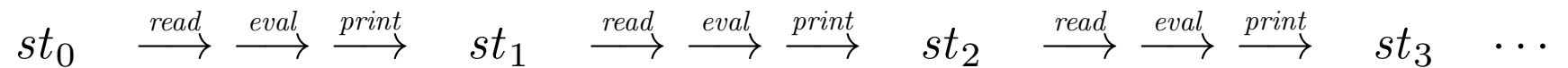


PIDE 2011/2012:

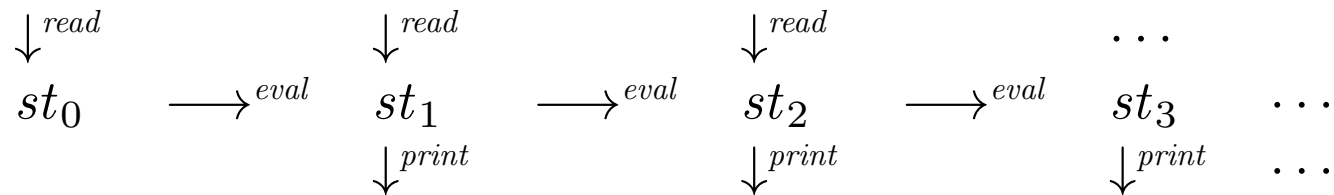


Command scheduling

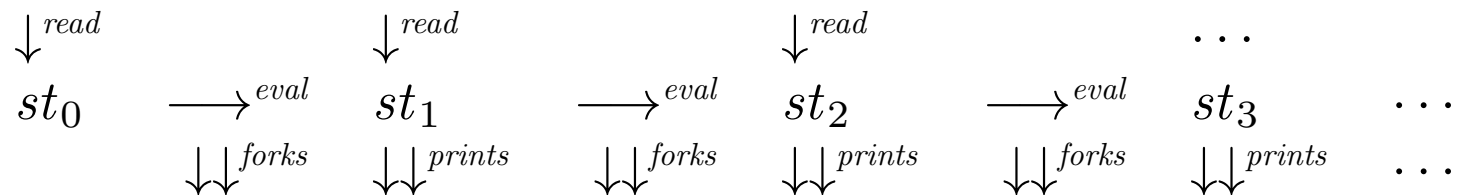
Sequential R-E-P Loop:



PIDE 2011/2012:



PIDE 2013/2014:



Document nodes

Global structure: directed acyclic graph (DAG) of theories

Local structure:

entries: linear sequence of command spans,
with static *command_id* and dynamic *exec_id*

perspective: visible and required commands,
according to structural dependencies

overlays: print functions over commands (with arguments)

Document nodes

Global structure: directed acyclic graph (DAG) of theories

Local structure:

entries: linear sequence of **command spans**,
with static *command_id* and dynamic *exec_id*

perspective: **visible** and **required** commands,
according to structural dependencies

overlays: **print functions** over commands (with arguments)

Notes:

- for each document version, the **command exec assignment** identifies results of (single) *eval st* or (multiple) *print st*
- the same execs may **coincide** for different versions
- non-visible / non-required commands remain **unassigned**

Document edits

Key operation: *update* \rightsquigarrow *assignment*

datatype *edit* = *Dependencies* | *Entries* | *Perspective* | *Overlays*

val *Document.update*: *version_id* \rightarrow *version_id* \rightarrow

(*node* \times *edit*) *list* \rightarrow *state* \rightarrow

(*command_id* \times *exec_id list*) *list* \times *state*

Notes:

- document update restructures **hypothetical execution**
 - **command exec assignment** is acknowledged quickly
 - actual **execution is scheduled** separately
- protocol thread remains reactive with reasonable latency

Execution management

Execution management in Isabelle/PIDE

Prerequisites:

- native threads in Poly/ML (D. Matthews, 2006 . . .)
- future values in Isabelle/ML (M. Wenzel, 2008 . . .)

Execution in PIDE 2013/2014:

Hypothetical execution: lazy execution outline with symbolic assignment of *exec_ids* to *eval* and *prints*

Execution frontiers: conflict avoidance of consecutive versions

Execution.start: $unit \rightarrow execution_id$

Execution.discontinue: $unit \rightarrow unit$

Execution.running: $execution_id \rightarrow exec_id \rightarrow bool$

Execution forks: managed future groups within execution context

Execution.fork: $exec_id \rightarrow (unit \rightarrow \alpha) \rightarrow \alpha \text{ future}$

Execution.cancel: $exec_id \rightarrow unit$

Asynchronous print functions

Asynchronous print functions

Observations:

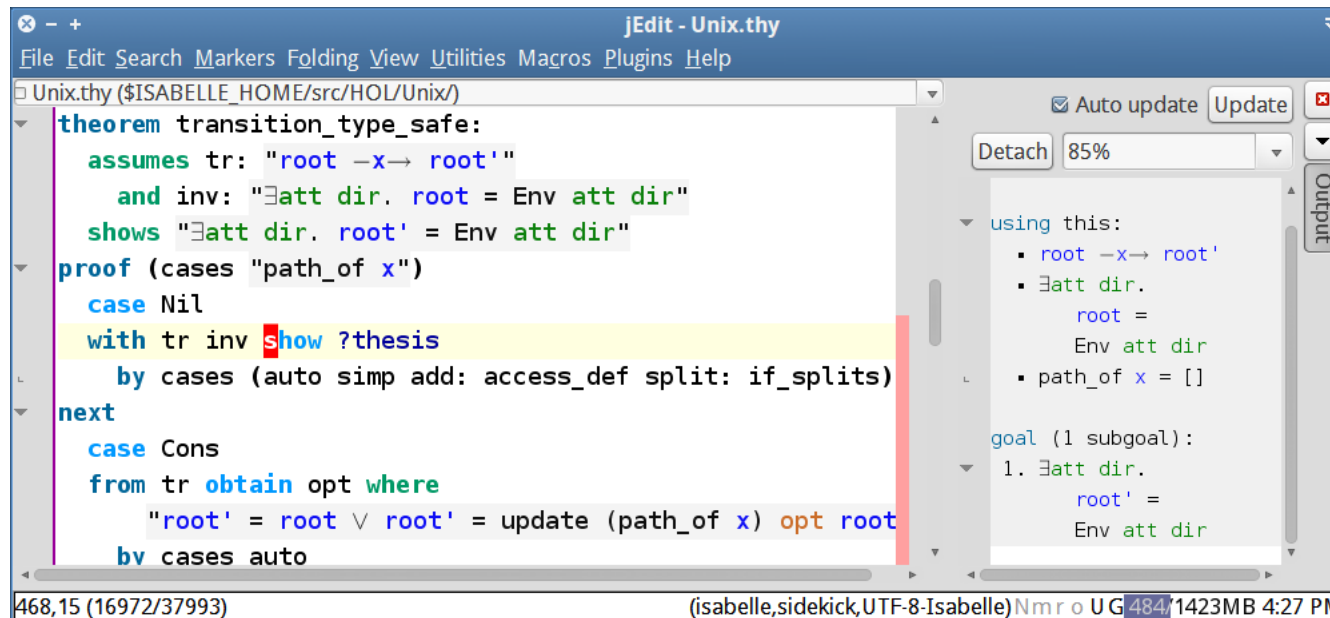
- cumulative *print* operations consume more time than *eval* (output of goals is slower than most proof steps)
- *print* depends on user perspective
- *print* may diverge or fail
- *print* augments results without changing proof state
- many different *prints* may be run independently

Approach:

- each command transaction is associated with several *exec_ids*: one *eval* + many *prints*
- document content forms **union of markup**
- print management via **declarative parameters**: startup delay, time-out, task priority, persistence, strictness wrt. eval state

Application: print proof state

- parameters: $\{pri = 1, persistent = false, strict = true\}$
- change of perspective invokes or revokes asynchronous / parallel prints spontaneously
- GUI panel follows cursor movement to display content



The screenshot shows the jEdit IDE with a file named Unix.thy. The main editor displays the following theorem and proof:

```
theorem transition_type_safe:
  assumes tr: "root -x→ root'"
  and inv: "∃att dir. root = Env att dir"
  shows "∃att dir. root' = Env att dir"
proof (cases "path_of x")
  case Nil
  with tr inv show ?thesis
  by cases (auto simp add: access_def split: if_splits)
next
  case Cons
  from tr obtain opt where
    "root' = root ∨ root' = update (path_of x) opt root"
  bv cases auto
```

The Output panel on the right shows the current proof state:

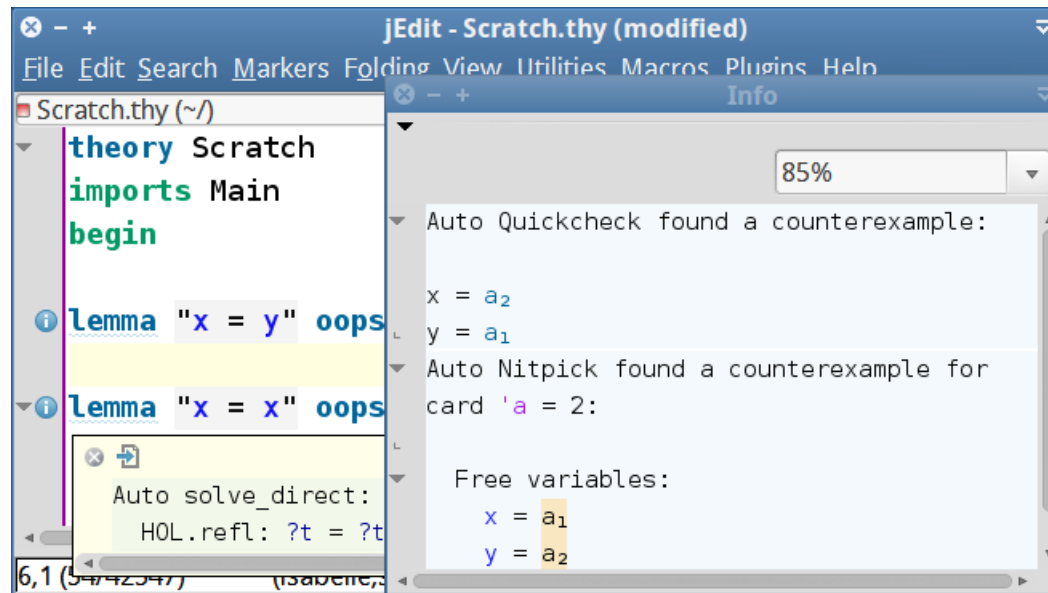
```
using this:
  • root -x→ root'
  • ∃att dir.
    root =
      Env att dir
  • path_of x = []

goal (1 subgoal):
  1. ∃att dir.
    root' =
      Env att dir
```

The status bar at the bottom indicates the cursor is at line 468, column 15 (16972/37993) in the file (isabelle.sidekick,UTF-8-Isabelle)Nmr o UG484, with a file size of 1423MB and the time is 4:27 PM.

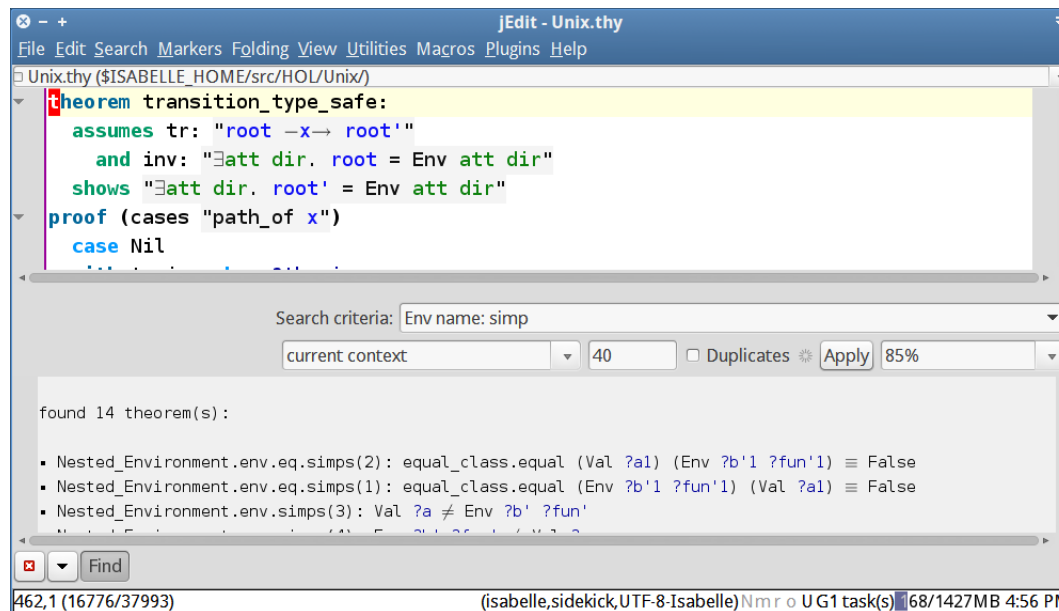
Application: automatically tried tools

- parameters: $\{delay = 1s, timeout = 4s, pri = -10, persistent = true, strict = true\}$
- long-running tasks with little output, e.g. automated (dis-)provers
- comment on existing document content via [information message](#)



Application: query operations with user input

- parameters: $\{pri = 0, persistent = false, strict = false\}$
- separate infrastructure to manage temporary **document overlays**
- stateful GUI panel with user input, system output, and control of corresponding command transaction (status icon, cancel button)



```
File Edit Search Markers Folding View Utilities Macros Plugins Help
Unix.thy ($ISABELLE_HOME/src/HOL/Unix)
theorem transition_type_safe:
  assumes tr: "root -x→ root'"
  and inv: "∃att dir. root = Env att dir"
  shows "∃att dir. root' = Env att dir"
proof (cases "path_of x")
case Nil
```

Search criteria: Env name: simp
current context 40 Duplicates 85%

found 14 theorem(s):

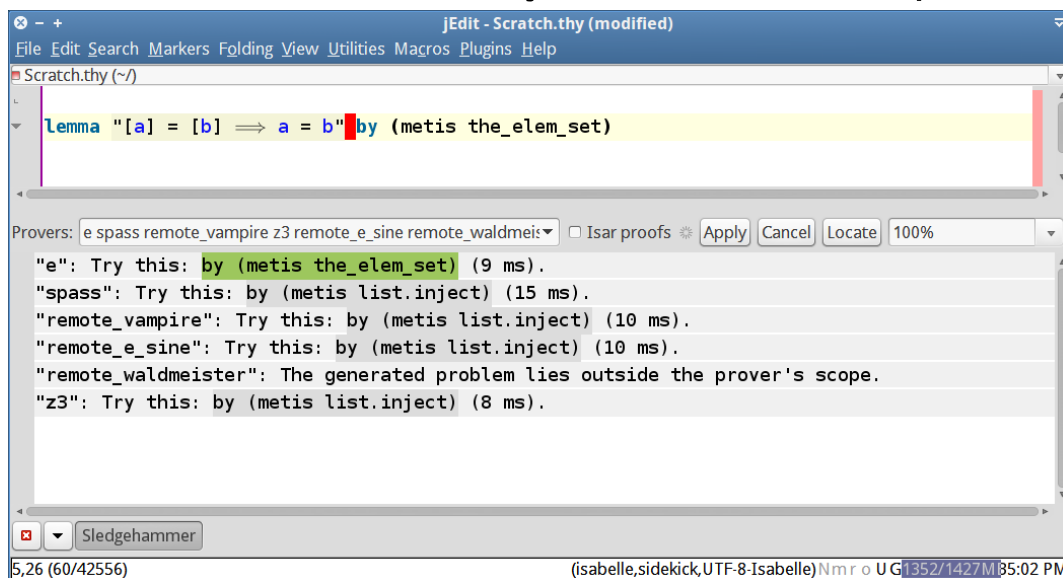
- Nested_Environment.env.eq.simps(2): equal_class.equal (Val ?a1) (Env ?b'1 ?fun'1) ≡ False
- Nested_Environment.env.eq.simps(1): equal_class.equal (Env ?b'1 ?fun'1) (Val ?a1) ≡ False
- Nested_Environment.env.simps(3): Val ?a ≠ Env ?b' ?fun'

Find

462,1 (16776/37993) (isabelle,sidekick,UTF-8-Isabelle) Nmr o U G1 task(s) 68/1427MB 4:56 PM

Application: Sledgehammer

- heavy-duty query operation, with long-running ATPs and SMTs in the background (local or remote)
- progress indicator (spinning disk)
- clickable output
- implementation: trivial corollary of above concepts



The screenshot shows the Sledgehammer application window titled "jEdit - Scratch.thy (modified)". The main text area contains the lemma: `Lemma "[a] = [b] ==> a = b" by (metis the_elem_set)`. Below the text area, there is a "Provers" section with a dropdown menu showing "e spass remote_vampire z3 remote_e_sine remote_waldmeis" and buttons for "Apply", "Cancel", and "Locate". The output area displays the results for each prover:

```
"e": Try this: by (metis the_elem_set) (9 ms).
"spass": Try this: by (metis list.inject) (15 ms).
"remote_vampire": Try this: by (metis list.inject) (10 ms).
"remote_e_sine": Try this: by (metis list.inject) (10 ms).
"remote_waldmeister": The generated problem lies outside the prover's scope.
"z3": Try this: by (metis list.inject) (8 ms).
```

At the bottom of the window, there is a status bar showing "5,26 (60/42556)" and "(isabelle,sidekick,UTF-8-Isabelle)Nm r o U G 1352/1427 M 85:02 PM".

Conclusions

Lessons learned

- Substantial reforms of LCF-style theorem proving are possible, with **big impact** on infrastructure, but **little impact** on existing tools.
 - **Parallel processing** is relatively easy, compared to the difficulties of asynchronous **user interaction** and **tool integration**.
 - Real-world frameworks like JVM/Swing impose technical side-conditions and challenges, notably for multi-platform support.
- Try out Isabelle/PIDE today and provide feedback on usability!
<http://isabelle.in.tum.de>
<http://isabelle.in.tum.de/website-Isabelle2014-RC0>