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:

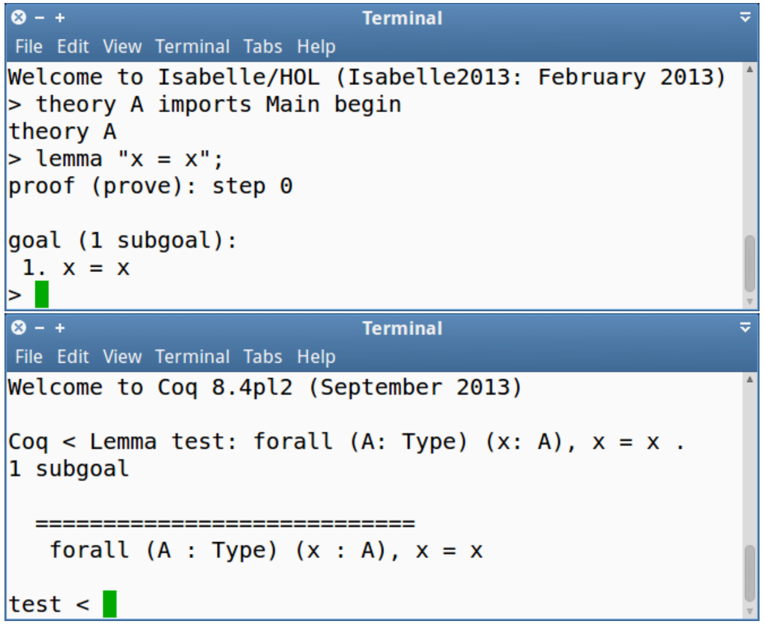
- renovate and reform interactive (and automated) theorem proving for new generations of users
- address paradigm shifts: multicore and pervasive parallelism
- document-oriented user interaction and tool integration

Ultimate challenge:

Introducing genuine interaction into ITP

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

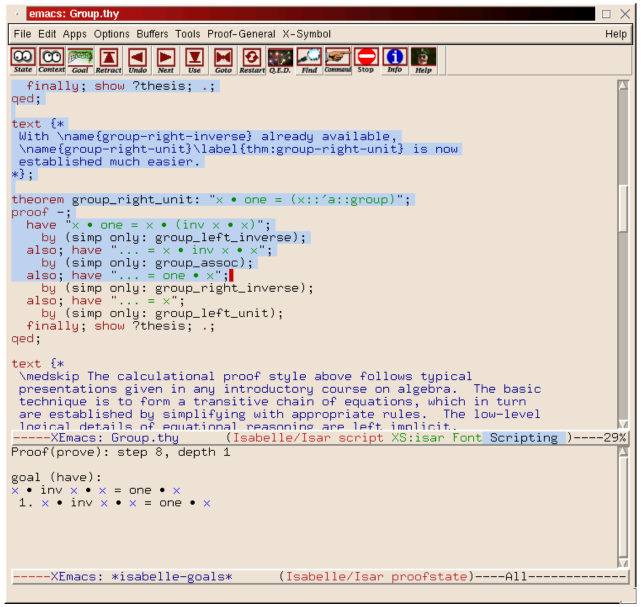
TTY loop (≈ 1979)

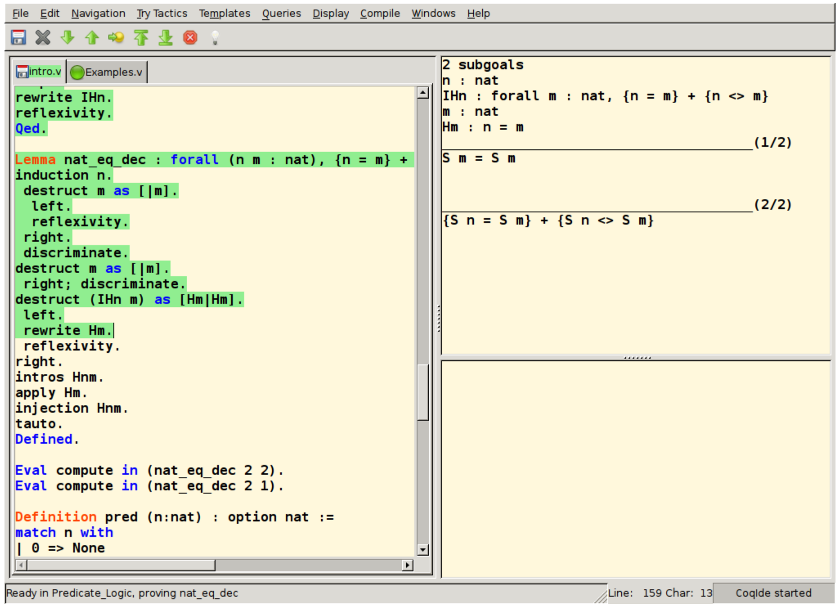


(Wikipedia: K. Thompson and D. Ritchie at PDP-11)

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

Proof General and clones (≈ 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

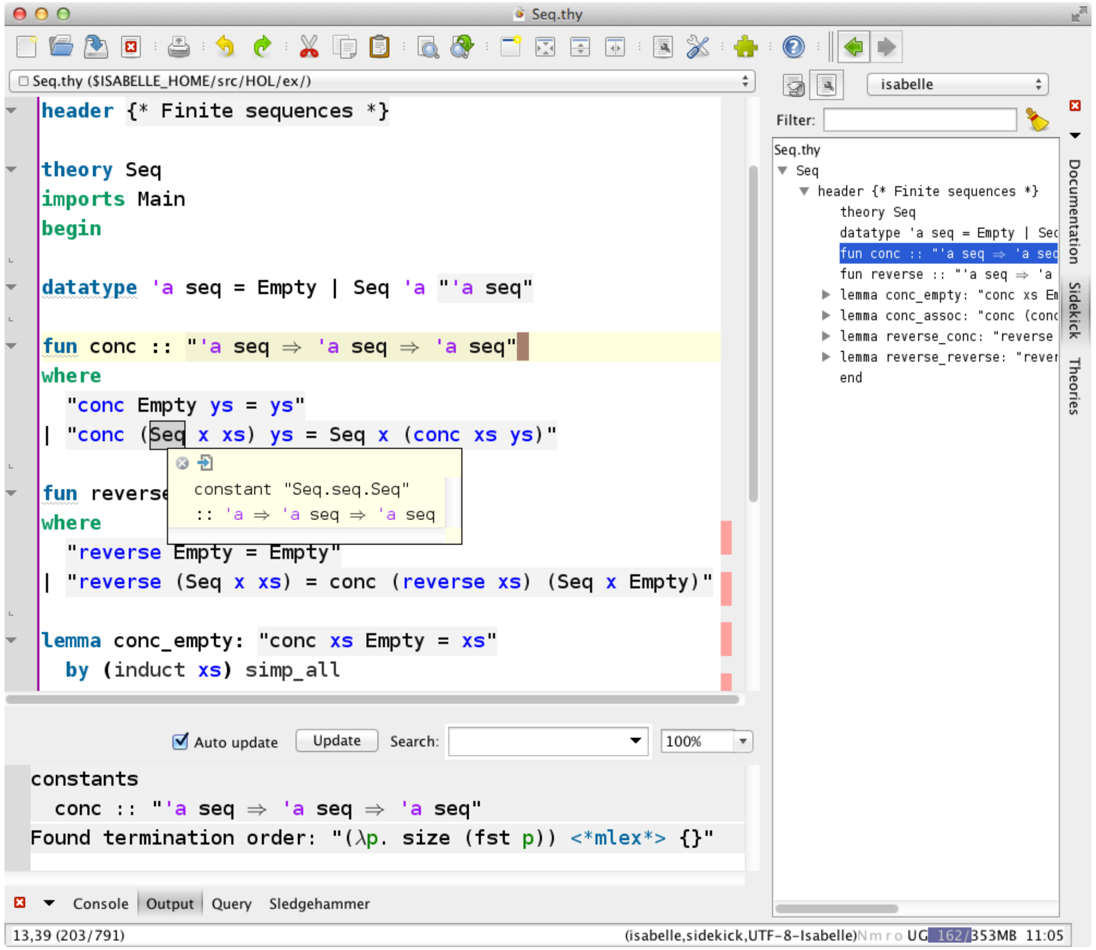
PIDE applications:

Isabelle/jEdit the default user-interface of Isabelle

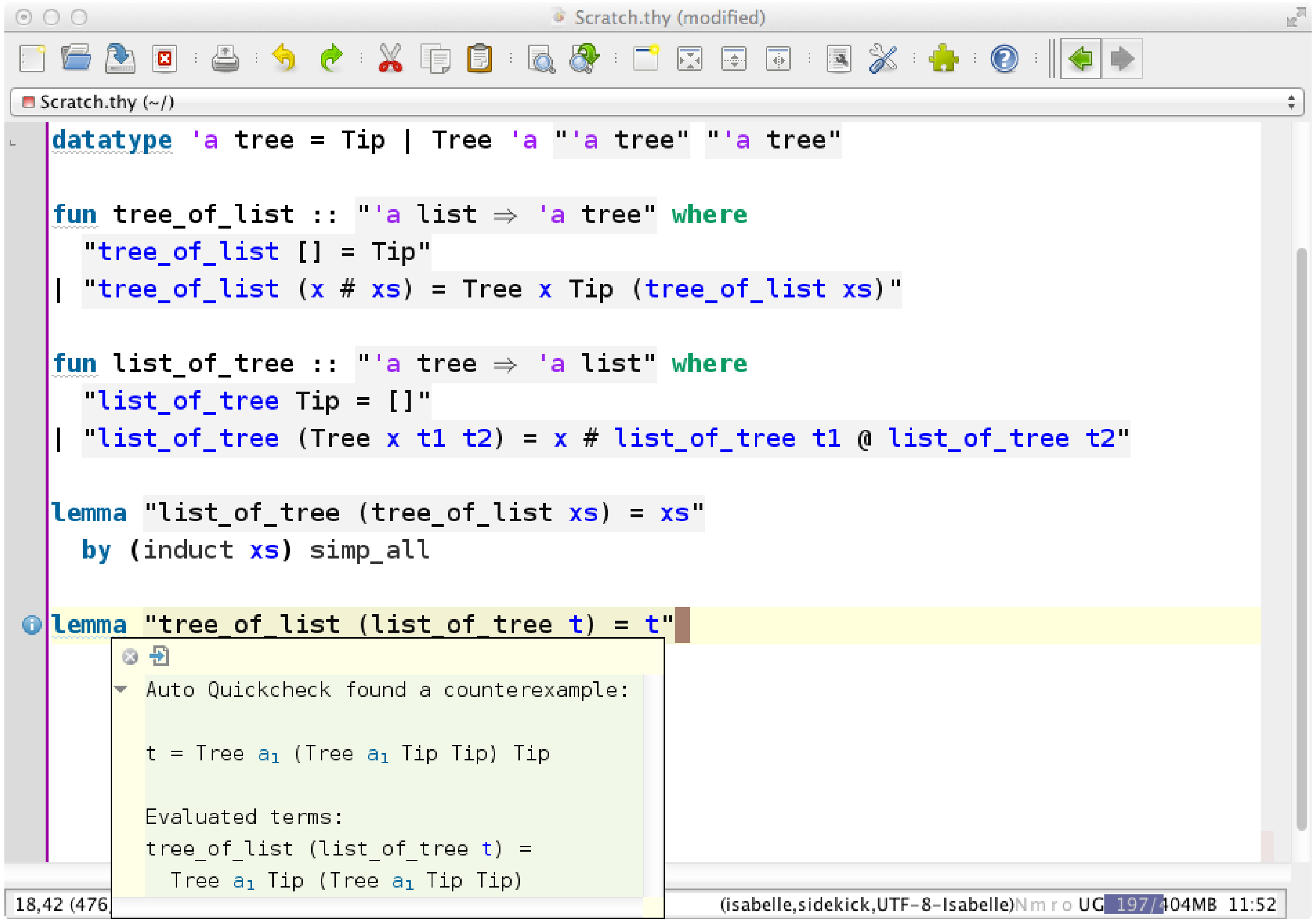
Isabelle/Eclipse by Andrius Velykis (for Isabelle2013)
`https://github.com/andriusvelykis/isabelle-eclipse`

Isabelle/Clide by Martin Ring and Christoph Lüth (subsequent talk)
`https://github.com/martinring/clide`

Isabelle/jEdit Prover IDE (2014)

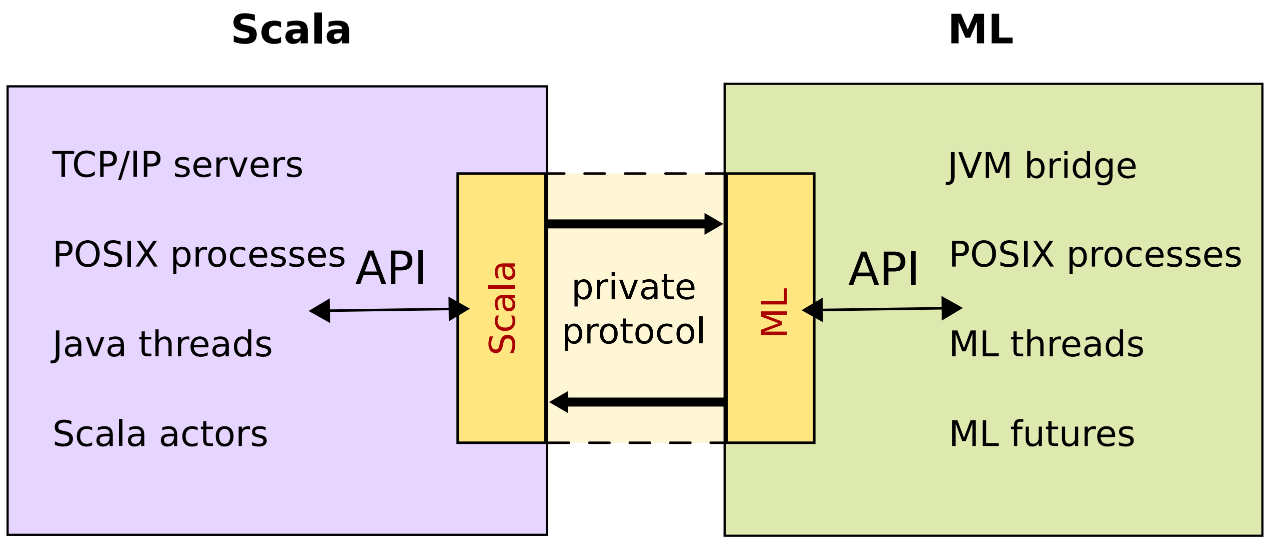


Automatically tried tools (2014)



PIDE architecture

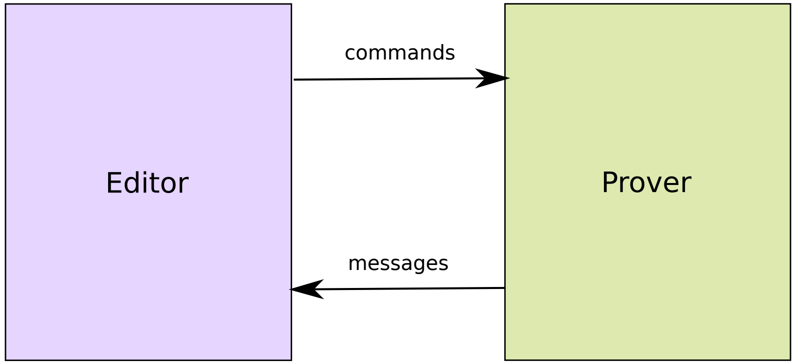
The connectivity problem



Design principles:

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

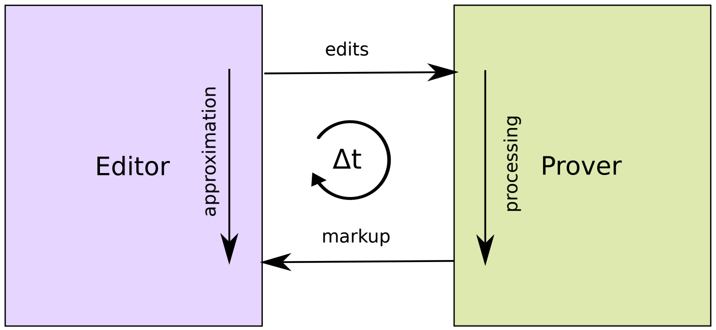
PIDE protocol functions



- `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

$\longrightarrow$ editor and prover as stream-procession 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):
   $\tilde{M} = \mathit{revert}\ \Delta T; \mathit{retrieve}\ M; \mathit{convert}\ \Delta T$
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:

$$st_0 \quad \xrightarrow{read} \xrightarrow{eval} \xrightarrow{print} \quad st_1 \quad \xrightarrow{read} \xrightarrow{eval} \xrightarrow{print} \quad st_2 \quad \xrightarrow{read} \xrightarrow{eval} \xrightarrow{print} \quad st_3 \quad \cdots$$

Command scheduling

Sequential R-E-P Loop:

$$st_0 \quad \xrightarrow{read} \xrightarrow{eval} \xrightarrow{print} \quad st_1 \quad \xrightarrow{read} \xrightarrow{eval} \xrightarrow{print} \quad st_2 \quad \xrightarrow{read} \xrightarrow{eval} \xrightarrow{print} \quad st_3 \quad \cdots$$

PIDE 2011/2012:

$$\begin{array}{ccccccccc}
\downarrow^{read} & & \downarrow^{read} & & \downarrow^{read} & & \cdots & \\
st_0 & \longrightarrow^{eval} & st_1 & \longrightarrow^{eval} & st_2 & \longrightarrow^{eval} & st_3 & \cdots \\
 & & \downarrow^{print} & & \downarrow^{print} & & \downarrow^{print} & \cdots
\end{array}$$

Command scheduling

Sequential R-E-P Loop:

$$st_0 \quad \xrightarrow{read} \xrightarrow{eval} \xrightarrow{print} \quad st_1 \quad \xrightarrow{read} \xrightarrow{eval} \xrightarrow{print} \quad st_2 \quad \xrightarrow{read} \xrightarrow{eval} \xrightarrow{print} \quad st_3 \quad \cdots$$

PIDE 2011/2012:

$$\begin{array}{cccccccc} \downarrow^{read} & & \downarrow^{read} & & \downarrow^{read} & & \cdots & \\ st_0 & \longrightarrow^{eval} & st_1 & \longrightarrow^{eval} & st_2 & \longrightarrow^{eval} & st_3 & \cdots \\ & & \downarrow^{print} & & \downarrow^{print} & & \downarrow^{print} & \cdots \end{array}$$

PIDE 2013/2014:

$$\begin{array}{cccccccc} \downarrow^{read} & & \downarrow^{read} & & \downarrow^{read} & & \cdots & \\ st_0 & \longrightarrow^{eval} & st_1 & \longrightarrow^{eval} & st_2 & \longrightarrow^{eval} & st_3 & \cdots \\ & \downarrow\downarrow^{forks} & \downarrow\downarrow^{prints} & \downarrow\downarrow^{forks} & \downarrow\downarrow^{prints} & \downarrow\downarrow^{forks} & \downarrow\downarrow^{prints} & \cdots \end{array}$$

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: $update \leadsto assignment$

$$
\begin{array}{l}
\textbf{datatype } edit = Dependencies \mid Entries \mid Perspective \mid Overlays \\
\textbf{val } Document.update\colon version\_id \to version\_id \to \\
\quad (node \times edit)\ list \to state \to \\
\quad (command\_id \times exec\_id\ list)\ list \times state
\end{array}
$$

Notes:

- document update restructures hypothetical execution
- command exec assignment is acknowledged quickly
- actual execution is scheduled separately

$\longrightarrow$ 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

$Execution.start\colon unit \to execution\_id$
$Execution.discontinue\colon unit \to unit$
$Execution.running\colon execution\_id \to exec\_id \to bool$

Execution forks: managed future groups within execution context

$Execution.fork\colon exec\_id \to (unit \to \alpha) \to \alpha\ future$
$Execution.cancel\colon exec\_id \to unit$

Asynchronous print functions

Asynchronous print functions

Observations:

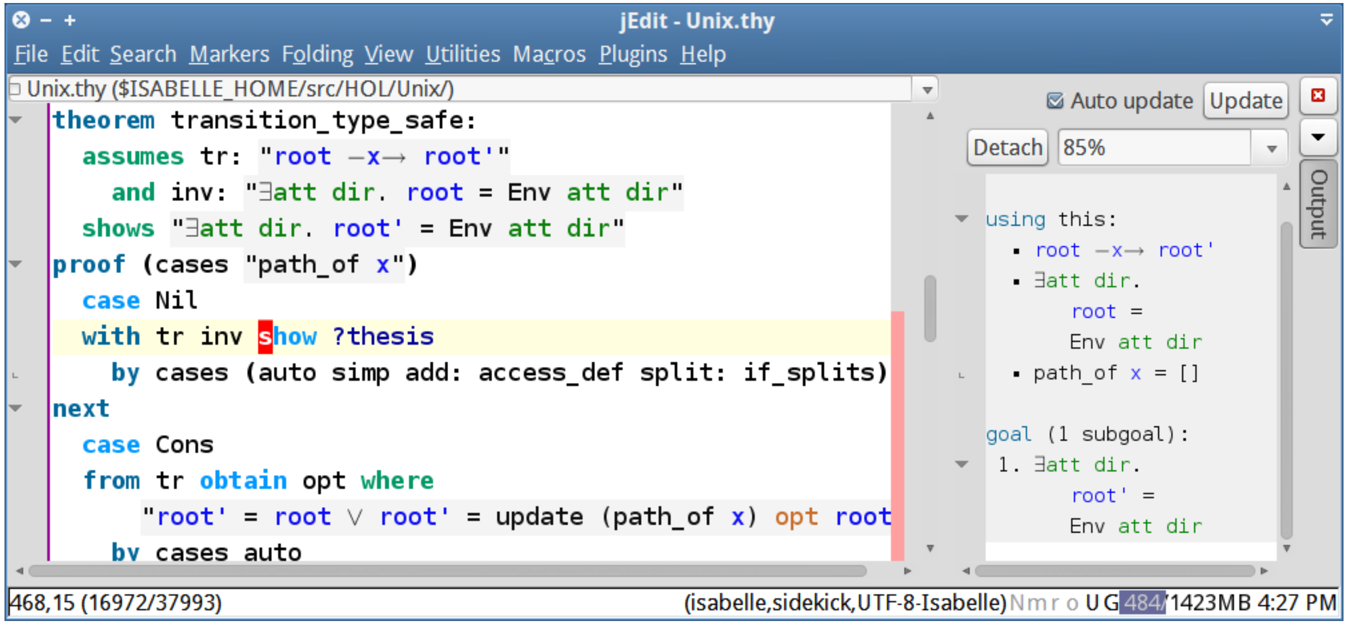
- cumulative $print$ operations consume more time than $eval$ (output of goals is slower than most proof steps)
- $print$ depends on user perspective
- $print$ may diverge or fail
- $print$ augments results without changing proof state
- many different $print$s may be run independently

Approach:

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

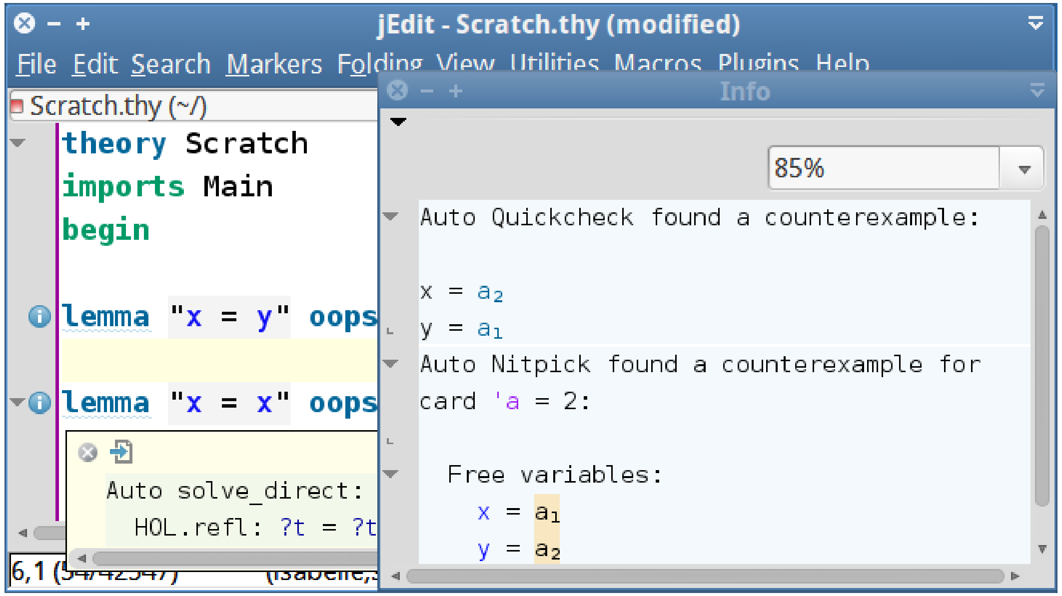
Application: print proof state

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



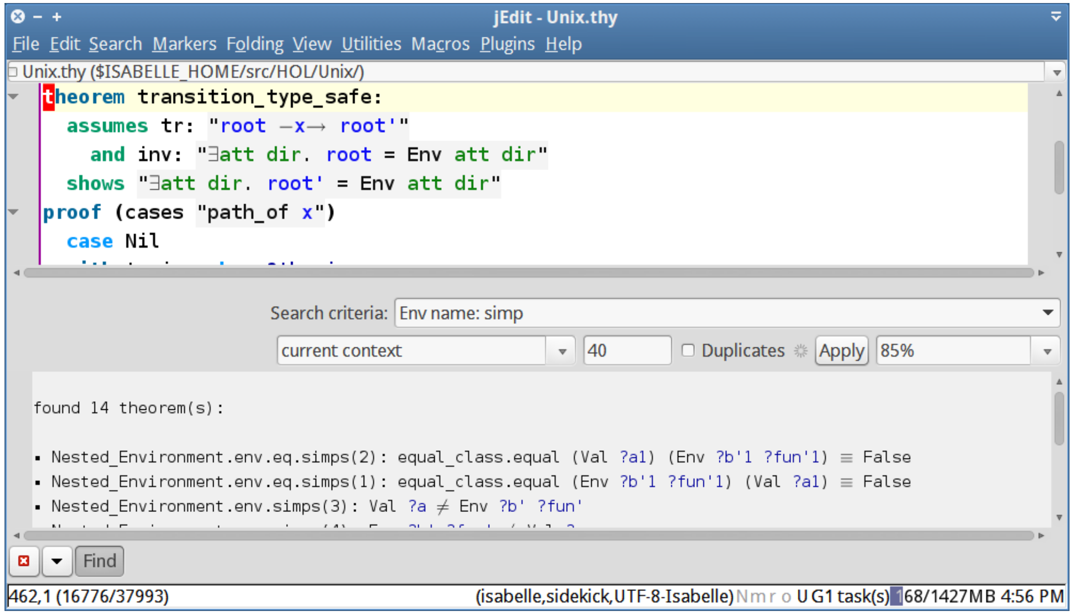
Application: automatically tried tools

- 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



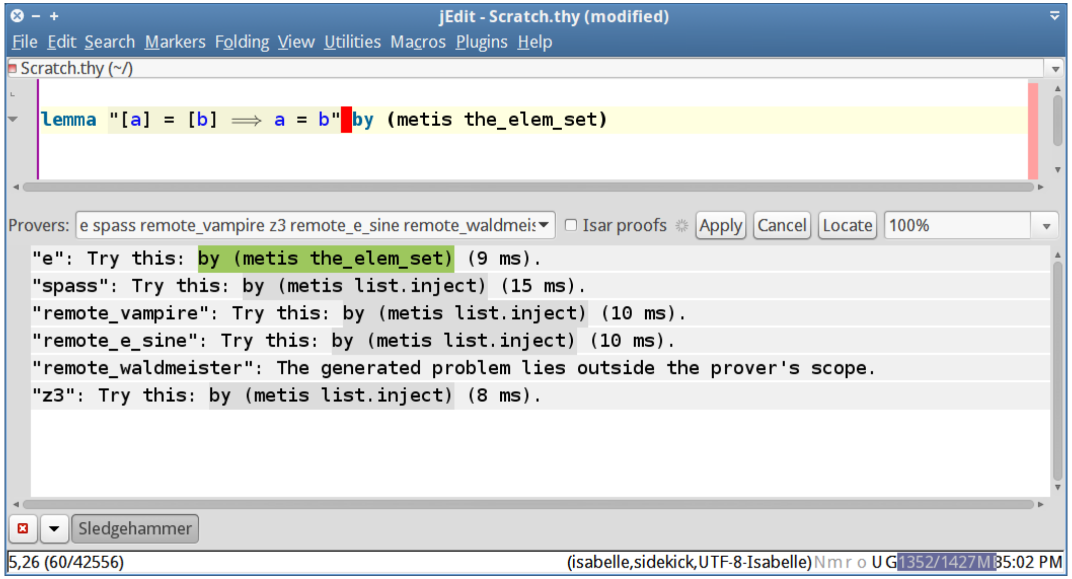
Application: query operations with user input

- 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)



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



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.

$\longrightarrow$ Try out Isabelle/PIDE today and provide feedback on usability!
`http://isabelle.in.tum.de`
`http://isabelle.in.tum.de/website-Isabelle2014-RC0`