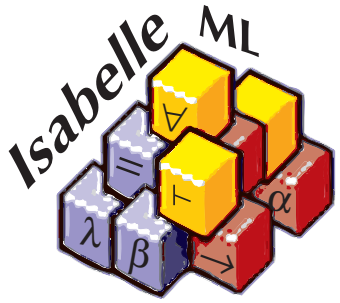
Isabelle/PIDE as IDE for ML

Makarius Wenzel
sketis.net

November 2016



Abstract

Isabelle is usually positioned as environment for interactive and automated theorem proving, but its Prover IDE (PIDE) may be used for regular program development as well. Standard ML is particularly important here, since it is the bootstrap language of Isabelle/ML (i.e. SML with many add-ons) and Isabelle/Pure (i.e. the logical framework).

The ML IDE functionality of Isabelle + Poly/ML is manifold:

- Continuous feedback from static analysis and semantic evaluation is already available for years, e.g. Isabelle2014 (August 2014). It is a corollary of how PIDE interaction works, and of the integration of the Poly/ML compiler into that framework. Source files are statically checked and semantically evaluated while the user is editing. The annotated sources contain markup about inferred types, references to defining positions of items etc.
- Source-level debugging within the IDE is new in Poly/ML 5.6, which is bundled with Isabelle2016 (February 2016). The Prover IDE provides the Debugger dockable to connect to running ML threads, inspect the stack frame with local ML bindings, and evaluate ML expressions in a particular run-time context. See also `http://sketis.net/2016/ml-debugging-within-the-prover-ide`.
- IDE support for the Isabelle/Pure bootstrap process is new technology for the coming release of Isabelle2016-1 (December 2016). The `ROOT.ML` file acts like a quasi-theory in the context of theory `ML_Bootstrap`: this allows continuous checking of all loaded ML files. The theory file is presented with a modified header to import Pure from the running Isabelle instance.

  It is also possible to modify standalone SML projects, to edit the sources freely in the ML IDE. For example, MetiTarski `https://bitbucket.org/lcpaulson/metitarski` can participate after some trivial changes of its `ROOT.ML` file.

  Overall, we move more and more to an integrated framework for development of formal-reasoning tools, but other applications are admissible as well.

Isabelle/PIDE

PIDE approach (2009)

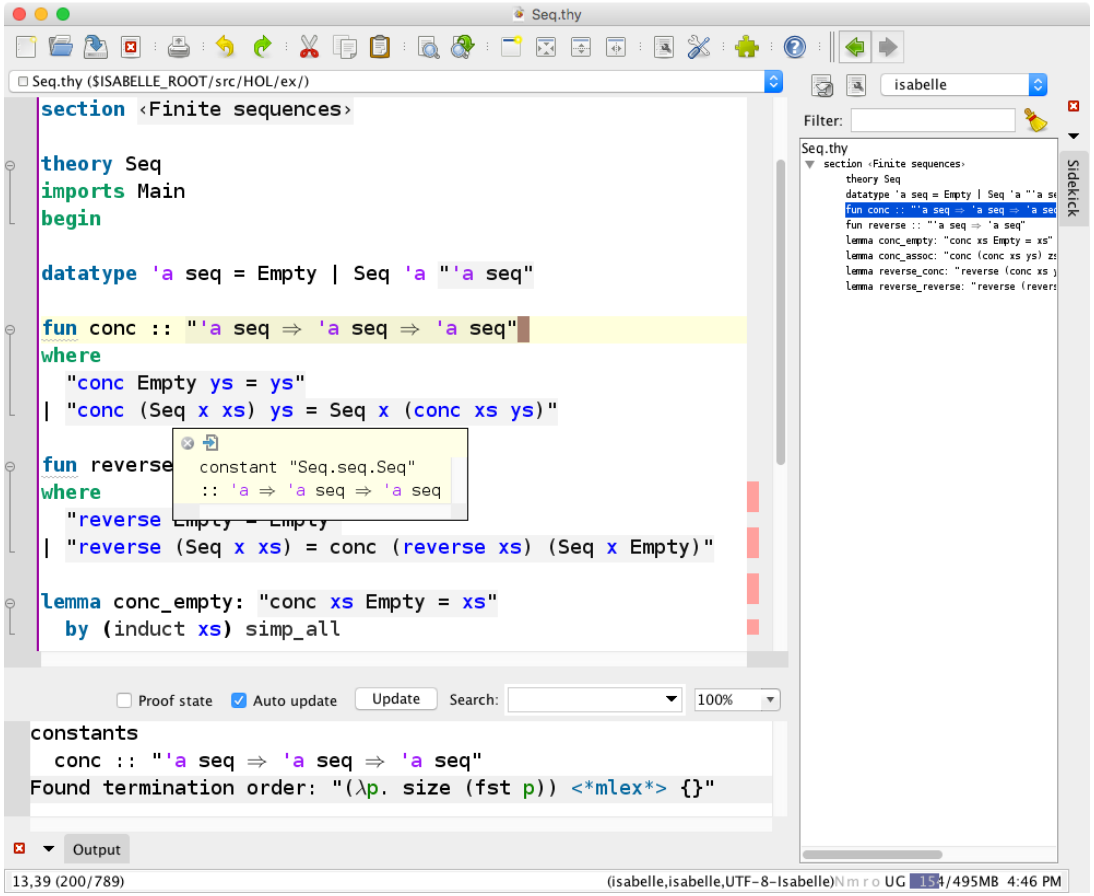
Prover supports asynchronous document model natively

Editor continuously sends source edits and receives markup reports

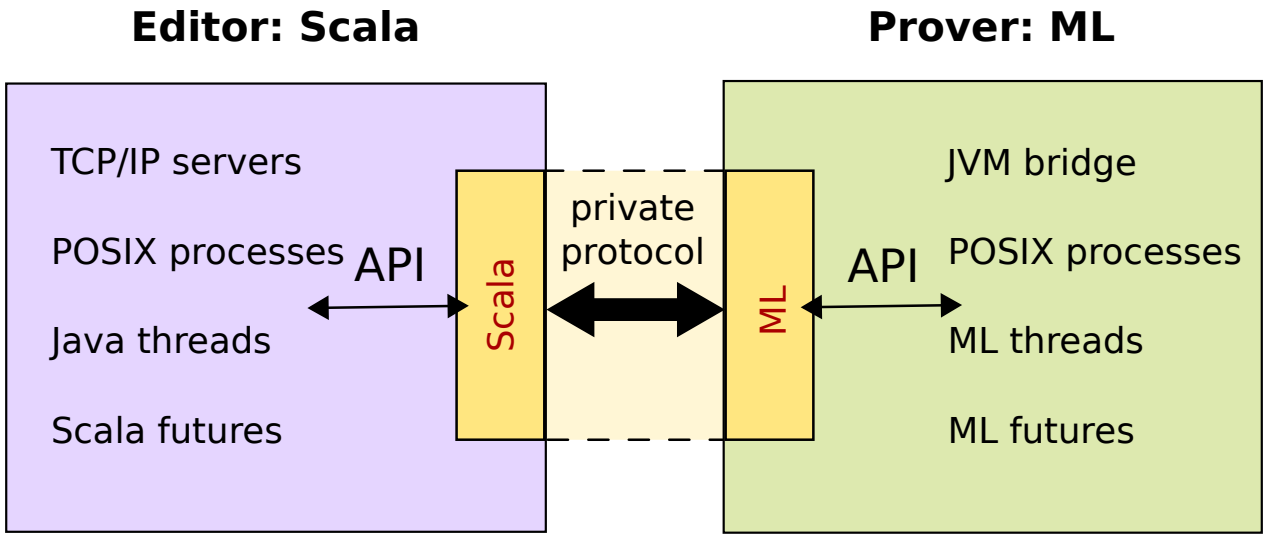
Tools may participate in document processing and markup

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

Isabelle/jEdit Prover IDE (2016)



PIDE architecture



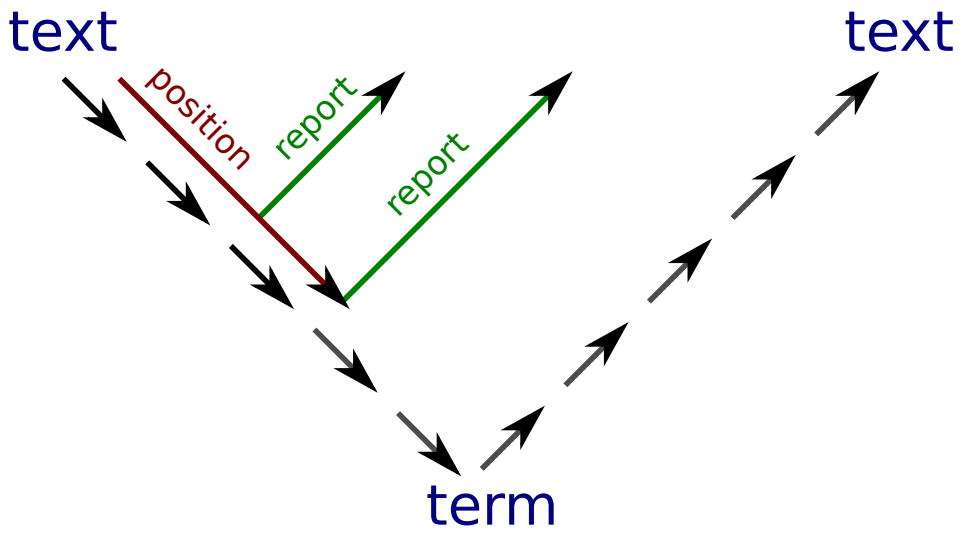
Design principles:

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

PIDE markup reports

Problem: round-trip through several sophisticated syntax layers

Solution: execution trace with markup reports



Example: semantic markup for domain-specific formal languages

Isabelle document language

Document structure

Markup

- section headings (6 levels like in HTML):
  chapter, **section**, **subsection**, . . . , **subparagraph**
- text blocks: **text**, **txt**, **text_raw**
- free-form LaTeX macros (rare)

Markdown

- implicit paragraphs and lists: itemize, enumerate, description

Document antiquotations

full form: @{*name* [*options*] *arguments* ...}

short form:

1. cartouche argument: \<^*name*>⟨*argument*⟩
2. no argument: \<^*name*>
3. standard name: ⟨*argument*⟩

Notable examples:

- *cartouche*, *theory_text*: self-presentation of Isar
- *bold*, *emph*, *verbatim*, *footnote*: text styles (with proper nesting)
- *noindent*, *smallskip*, *medskip*, *bigskip*: spacing
- *cite*: formal BibTEX items
- *path*, *file*, *dir*, *url*, *doc*: system resources

Isabelle/ML IDE

Continuous feedback from static analysis and semantic evaluation

- available for years, e.g. Isabelle2014 (August 2014)
- corollary of PIDE interaction and Poly/ML compiler integration
  - static checking (syntax) e.g. warnings, errors, inferred types, references to defined items
  - dynamic evaluation (semantics) e.g. toplevel output

  $\longrightarrow$ Isabelle/PIDE as semantic editor for Isabelle/ML and Standard ML

Example: inlined ML source

ML ⟨

$$fun\ factorial\ 0 = 1$$
$$\mid factorial\ n = n * factorial\ (n - 1)$$

⟩

ML ⟨$factorial\ 42$⟩
ML ⟨$factorial\ 10000\ div\ factorial\ 9999$⟩

Example: external ML source

ML_file $ackermann.ML$

- **ML_file** better suited for big modules
- more editor support (mode `isabelle-ml`): keywords, Sidekick, folding

Example: big ML projects within theory hierarchy

`~~/src/HOL/HOL.thy` `~~/src/Tools/quickcheck.ML`

Example: official Standard ML

See `~~/src/Tools/SML/Examples.thy`.

Source-level debugging within the IDE

- new in Poly/ML 5.6, which is bundled with Isabelle2016 (February 2016)
- `Debugger` dockable (`isabelle-debugger`):
  - connect to running ML threads
  - inspect stack frame with local ML bindings
  - evaluate ML expressions in run-time context

  See also *jedit* chapter 5 or `http://sketis.net/2016/ml-debugging-within-the-prover-ide`.

Example: ML debugger

context notes $[[ML\_debugger]]$
begin

ML ⟨

$$\begin{array}{l} \quad fun\ factorial\ 0 = 1 \\ \quad\quad |\ factorial\ n = n * factorial\ (n - 1) \end{array}$$
⟩

end

ML ⟨$factorial\ 10$⟩

IDE support for the Isabelle/Pure bootstrap process

Entry points (independent!)

- `~~/src/Pure/ROOT0.ML` – pre-bootstrap
- `~~/src/Pure/ROOT.ML` – main bootstrap
- `~~/src/Pure/Pure.thy` – final Pure setup (Isar commands)
- `~~/src/Pure/ML_Bootstrap.thy` – theory context for ML bootstrap

Bootstrap state

- `~~/src/Pure/Concurrent/thread_data.ML` – physical thread data
- `~~/src/Pure/Concurrent/thread_data_virtual.ML` – Isabelle context data

Standalone SML projects

Example: MetiTarski

```
hg clone https://bitbucket.org/lcpaulson/metitarski
hg import -R metitarski --no-commit
metitarski-change
```

- original: `https://bitbucket.org/lcpaulson/metitarski`
- minimal change: `metitarski-change`
- main directory: `metitarski`
- editable project sources: `metitarski/src/ROOT.ML`

Conclusions

Conclusions

Self-application of Isabelle/PIDE to Isabelle/Pure

Revival of Standard ML as Isabelle/ML (based on Poly/ML)

Demonstration for other Domain-specific Formal Languages

Try it yourself!

Current release: February 2016

`http://isabelle.in.tum.de`

Next release: December 2016

`http://isabelle.in.tum.de/website-Isabelle2016-1-RC2`