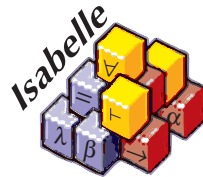


Future Prospects of Isabelle Technology

Makarius Wenzel
<http://sketis.net>

November 2017



- Prover IDE (PIDE) for Interactive Theorem Proving
 - Isabelle/jEdit PIDE
 - Isabelle/VSCoDe PIDE
- Isabelle document preparation
- Development environment for Isabelle/ML and SML

Abstract

In the past 3 decades, Isabelle has made a long way from a modest LCF-style proof assistant (with copy-paste of proof scripts written in ML) to the current Isabelle/PIDE editor-environment (with its timeless and stateless processing of proof documents). In this presentation, I will try to extrapolate this into the future: How far can we scale proof documents and libraries, e.g. via moving Isabelle into the “cloud”? How can we reduce system resource requirements on the client side? How can we upgrade interactive edits produced by a single author, towards versioned changesets by multiple or distributed authors? What are suitable frameworks for the next generation of Isabelle document preparation? What can we make out of Isabelle/ML as ultra-clean environment for functional programming? Etc. etc.

What is Isabelle?

Framework of domain-specific formal languages

Logic:

Isabelle/Pure: Logical framework and bootstrap environment

Isabelle/HOL: Theories and tools for applications

Programming:

Isabelle/ML: Tool implementation (Poly/ML)

Isabelle/Scala: System integration (JVM)

Proof:

Isabelle/Isar: Intelligible semi-automated reasoning

Document language: \LaTeX type-setting of proof text

Prover IDE (PIDE) for Interactive Theorem Proving

Prover 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

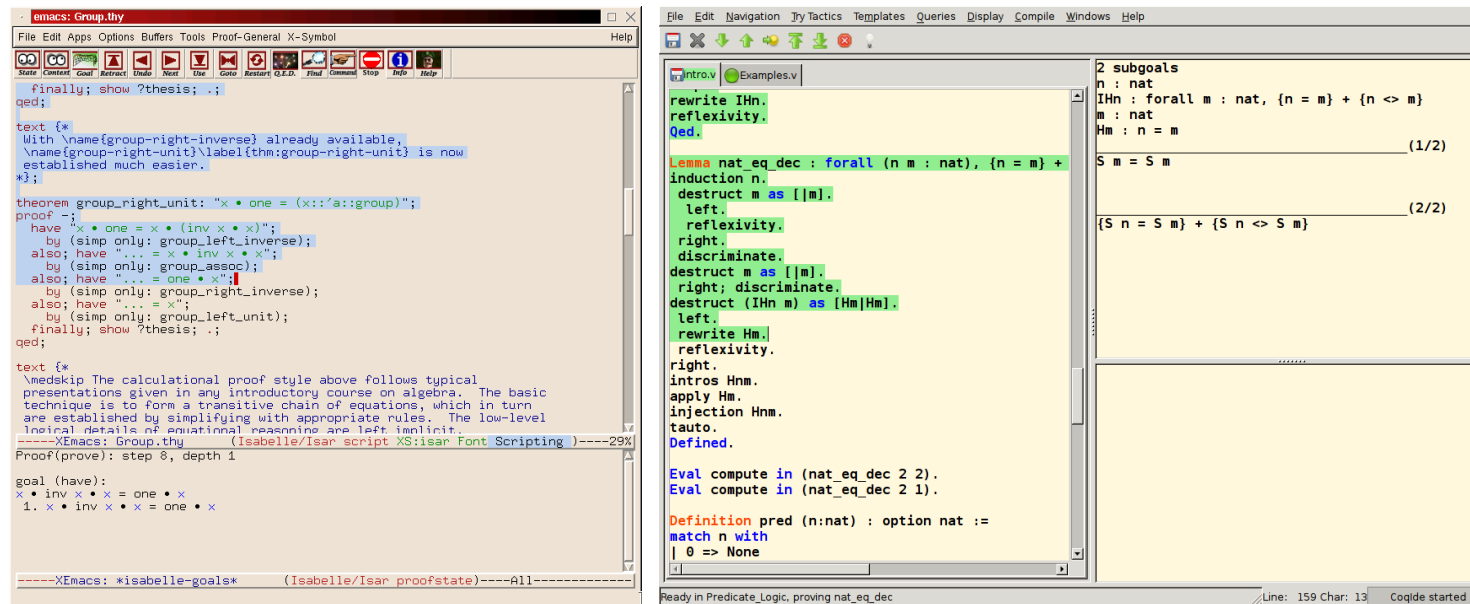
test < █
```



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

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

Proof General and clones (\approx 1999)



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

PIDE: Prover IDE (\approx 2008)

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**

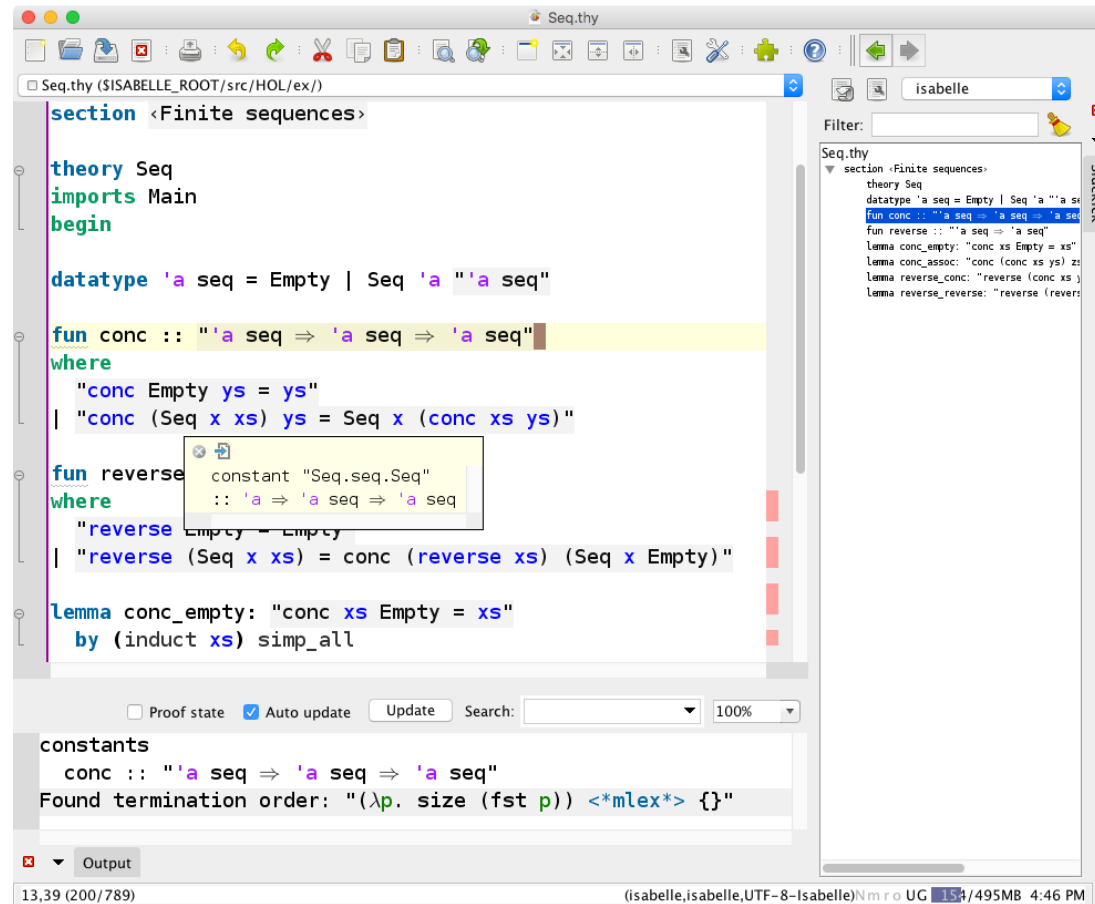
Challenge: introducing **genuine interaction** into ITP

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

Isabelle/jEdit PIDE

Isabelle/jEdit Prover IDE (October 2017)

- asynchronous interaction
- continuous checking
- parallel processing



Building blocks

jEdit: <http://www.jedit.org>

- sophisticated **text editor** implemented in Java

Scala: <http://www.scala-lang.org>

- higher-order functional-object-oriented programming on JVM

PIDE:

- general framework for Prover IDEs based on Scala
- with **parallel and asynchronous** document processing

Isabelle/jEdit:

- main example application of the PIDE framework
- default user-interface for Isabelle
- **filthy rich client**: requires 4–8 GB memory, 2–4 CPU cores

Timeline

Parallel Isabelle

- 2005 “free lunch is over”: **multicore** invasion into consumer market
- 2006–2008 Isabelle + Poly/ML support multicore hardware in **batch mode**

Isabelle/PIDE/jEdit

- 2008–2010: **experimental** Isabelle/jEdit Prover IDE
- October 2011: **first release** of Isabelle/jEdit 1.0
- October 2017: Isabelle/jEdit 9.0

Isabelle/VSCoDe

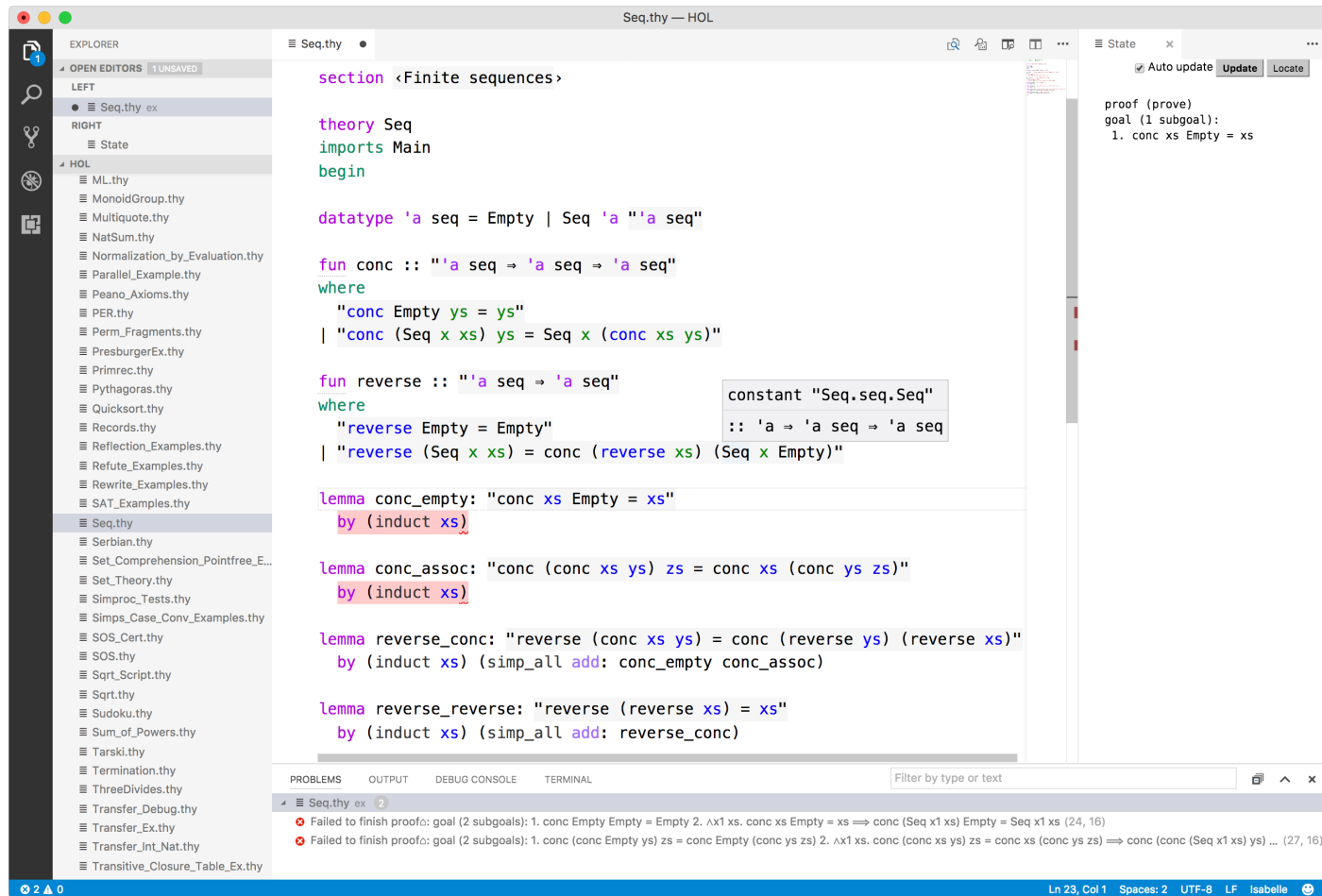
- Early 2017: **experimental** Isabelle/VSCoDe Prover IDE
- October 2017: **first release** Isabelle/VSCoDe 1.0

Isabelle/VSCode PIDE

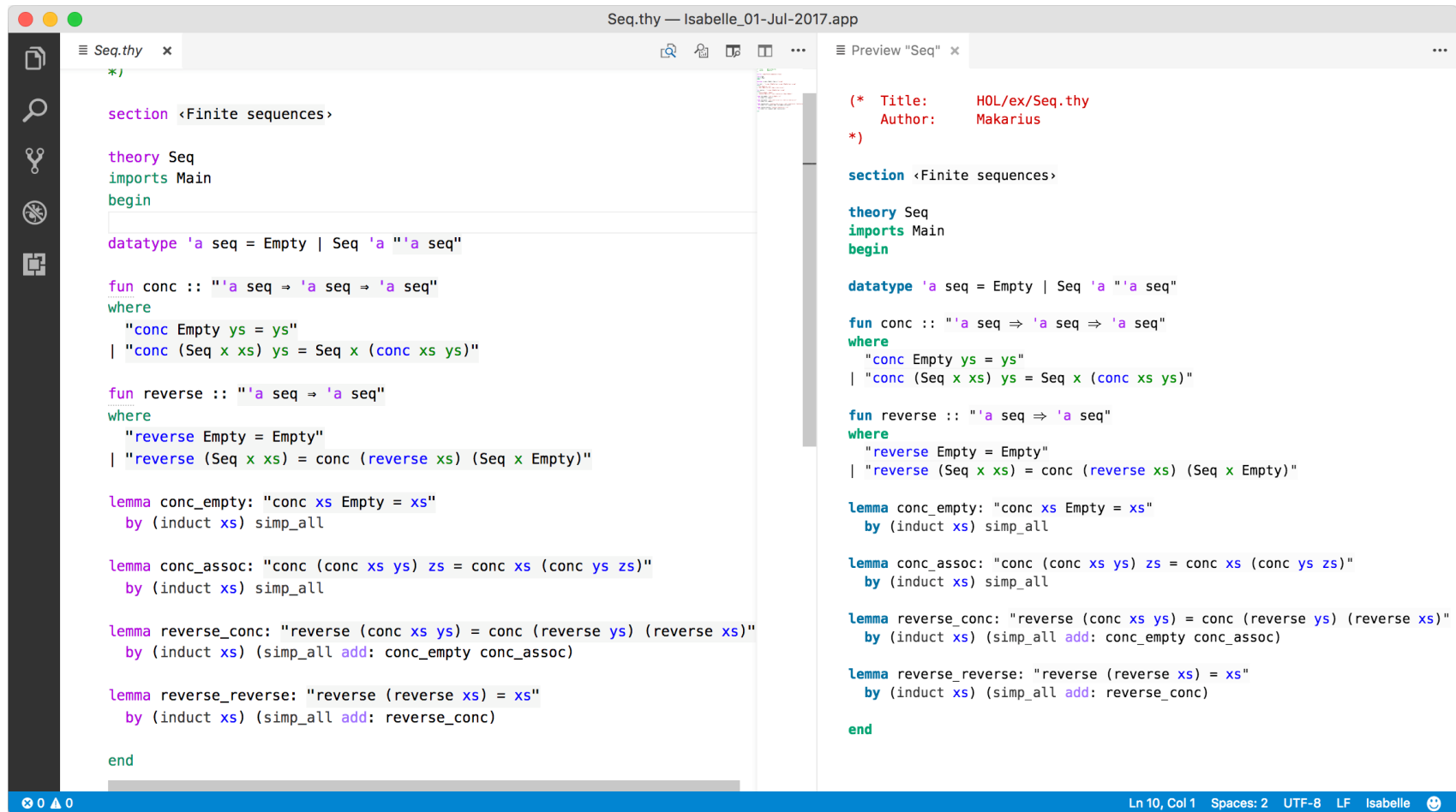
Building blocks

- Visual Studio Code editor platform:
 - recent open-source project by Microsoft
 - “Code editing. Redefined. Free. Open Source. Runs everywhere.”
 - based on [Electron](#) application framework
 - with [Node.js](#), [Chromium](#) browser, [V8](#) JavaScript engine
 - IDE for [TypeScript](#) in TypeScript (typed JavaScript)
- Isabelle/Scala/PIDE: slightly reworked for multiple front-ends
- Isabelle extension: via VSCode marketplace

Isabelle/VSCode: document-oriented interaction



Isabelle/VSCode: document preview



The screenshot displays the Isabelle/VSCode PIDE interface. The main editor window shows the source code of a file named `Seq.thy`. The code defines a theory `Seq` for finite sequences, including a datatype `'a seq`, a concatenation function `conc`, and a reversal function `reverse`, along with several lemmas. A right-hand pane, titled "Preview 'Seq'", shows a rendered version of the same code, highlighting the document preview feature. The status bar at the bottom indicates the current position is line 10, column 1, with 2 spaces, UTF-8 encoding, LF line endings, and the Isabelle language.

```
Seq.thy — Isabelle_01-Jul-2017.app

Seq.thy x
*)

section <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 :: "'a seq ⇒ 'a seq"
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

lemma conc_assoc: "conc (conc xs ys) zs = conc xs (conc ys zs)"
  by (induct xs) simp_all

lemma reverse_conc: "reverse (conc xs ys) = conc (reverse ys) (reverse xs)"
  by (induct xs) (simp_all add: conc_empty conc_assoc)

lemma reverse_reverse: "reverse (reverse xs) = xs"
  by (induct xs) (simp_all add: reverse_conc)

end

Preview "Seq" x

(* Title:      HOL/ex/Seq.thy
   Author:     Makarius *)

section <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 :: "'a seq ⇒ 'a seq"
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

lemma conc_assoc: "conc (conc xs ys) zs = conc xs (conc ys zs)"
  by (induct xs) simp_all

lemma reverse_conc: "reverse (conc xs ys) = conc (reverse ys) (reverse xs)"
  by (induct xs) (simp_all add: conc_empty conc_assoc)

lemma reverse_reverse: "reverse (reverse xs) = xs"
  by (induct xs) (simp_all add: reverse_conc)

end

Ln 10, Col 1  Spaces: 2  UTF-8  LF  Isabelle
```

Spin-off: IDE Language Server Protocol

Website: <http://langserver.org>

Maintainer: Microsoft

Purpose: “language smartness” for many editors

- JSON-RPC
- protocol for completion, goto-definition, scope-highlighting, . . .
- front-ends: VSCode, Eclipse, Neovim, . . .
- back-ends: JavaScript, TypeScript, C#, Go, Java, Isabelle/Isar, Isabelle/ML, . . .

Potential of the VSCode/Electron platform

Technology

- High-quality HTML rendering within **just one browser**
- Perspective for viable multiplatform support (without X11 on Linux)
- Generic GUI integration for:
 - Debugger
 - Version Control System (Git, Mercurial, . . .)

Ecosystem

- Project managed by developers at Microsoft
- Young and active community
- Many emerging projects and extensions

Future prospects for PIDE

- frontend: [high-quality HTML](#) presentation (e.g. via VSCode)
- backend: [headless PIDE](#) for “cloud” (e.g. via SSH or WebSocket)
- separation of edits/markup protocol vs. [display protocol](#) (e.g. for web client)
- PIDE edits vs. [Mercurial changesets](#):
 - multi-version editing
 - semantic annotations for changesets
- full-scale library editing / browsing (e.g. with PostgreSQL database server for PIDE markup)

Isabelle document preparation

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: $\text{@}\{name\} [options] arguments \dots\}$

short form:

1. cartouche argument: $\backslash\langle^name\rangle\langle argument\rangle$
2. no argument: $\backslash\langle^name\rangle$
3. standard name: $\langle argument\rangle$

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

HTML output

Status quo:

- static HTML with minimal CSS
- imitation of Isabelle/jEdit syntax highlighting
- limited semantic markup via Isabelle/jEdit `isabelle.preview`

Future prospects:

- support for document markup / markdown structure
- support for document antiquotations
- high-quality HTML / CSS rendering
- \LaTeX -math quality via MathJax
- interactive HTML presentation (e.g. `reveal.js`)

\LaTeX output

Status quo:

- \LaTeX sources and `pdflatex` runs via `isabelle build`
- pretty-printing of Isabelle symbols and tokens

Future prospects:

- `pdflatex` runs within PIDE
- \LaTeX errors and warnings with source positions
- use of semantic markup for document output

Development environment for Isabelle/ML and SML

The glorious past of SML

Website:

- <http://sml-family.org>
- <http://sml-family.org/history>

Classic documents:

- The Definition of Standard ML (SML'90)
<https://github.com/SMLFamily/The-Definition-of-Standard-ML>
- The Definition of Standard ML, revised (SML'97)
<https://github.com/SMLFamily/The-Definition-of-Standard-ML-Revised>

Best-known implementations:

SML/NJ <http://www.smlnj.org>

MLton <http://www.mlton.org>

Moscow ML <http://mosml.org>

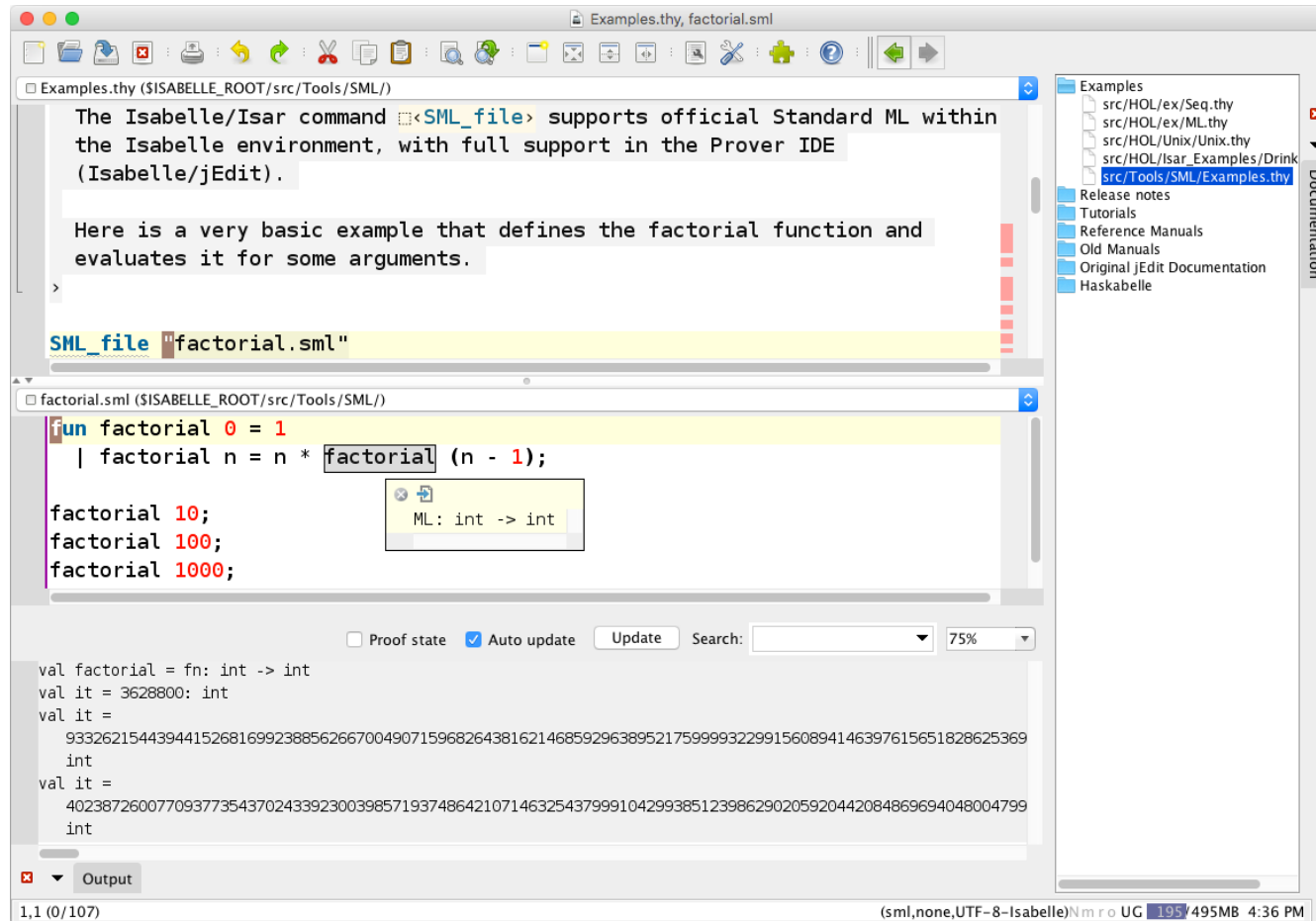
Poly/ML (since 1985, David Matthews)

- SML'97 with basis library
- interrupts from SML'90
- fast compiler that produces fast machine code (x86, x86_64)
- incremental runtime compilation (“eval”)
- runtime heap compaction (substructure sharing)
- dumped heap images and loadable modules
- linkable object files
- foreign language interface (libffi)
- **multicore** support (threads and locks)
- parallel garbage collection
- source-level **debugger**
- **IDE support**

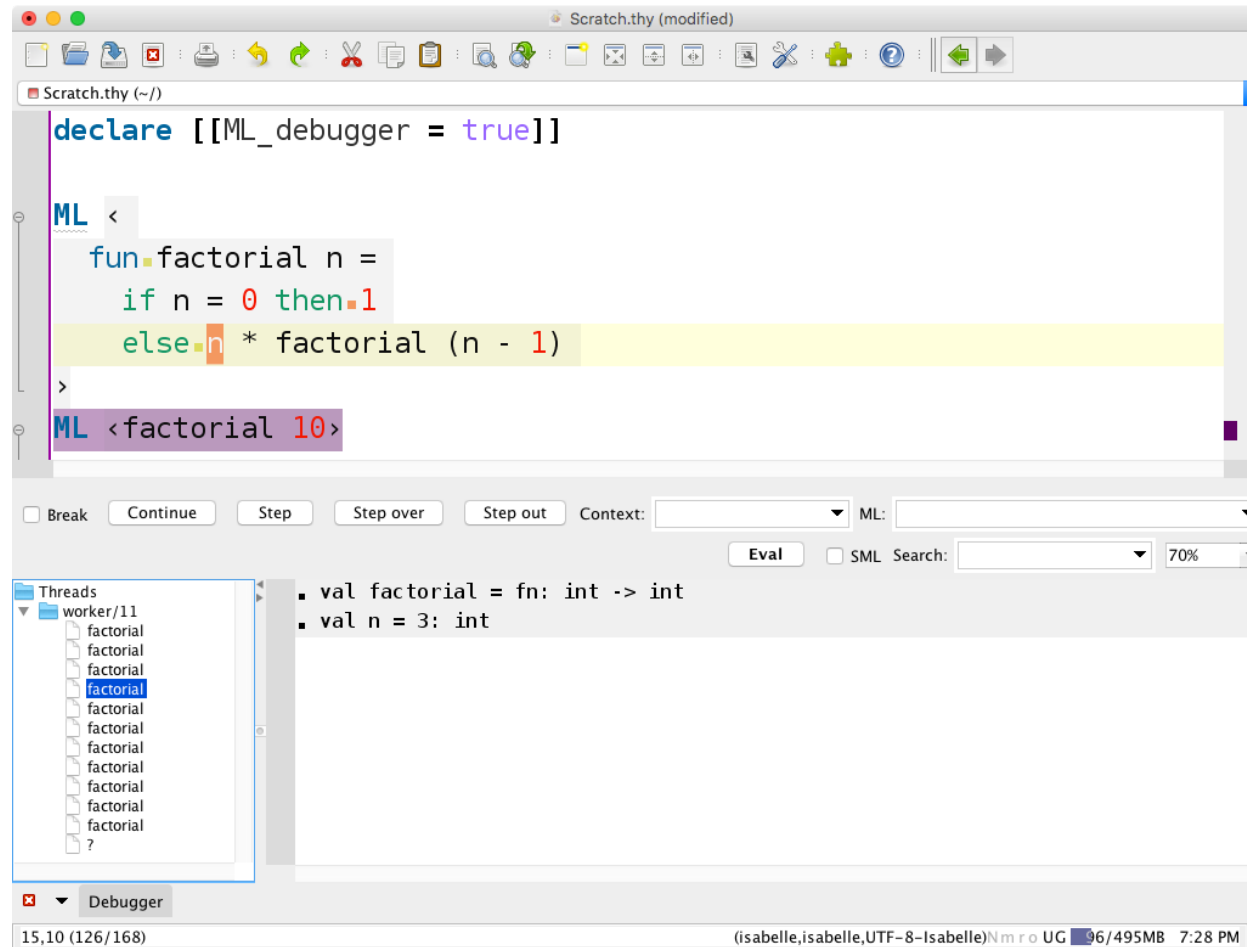
Isabelle/ML

- based on Poly/ML (exclusively)
- embrace-and-extend version of SML'90 + SML'97
- advanced Isabelle/ML library (**not** SML'97 basis library)
- **parallel evaluation** with futures, promises, lazy values
- **immutable data** managed within logical context
- compilation within logical context
- statically checked antiquotations
- rational numbers with literals
- mathematical symbols
- **PIDE support**: static phase, dynamic phase, debugger
- self-application: Isabelle/Pure can load itself within PIDE

Isabelle/ML: IDE



Isabelle/ML: debugger



Future prospects for ML

- Stop using ML on the command-line!
- Update teaching materials and textbooks for Isabelle/ML IDE
- Update examples for idiomatic Isabelle/ML:
 - `stateless` programming via context data
 - `pretty printing` with IDE layout and markup
 - proper `mathematical symbols` instead of ASCII art (or Unicode)
 - embedded languages via `nested cartouches`
 - reuse `library` to implement your own theorem prover

Isabelle footprint:

- download size: 165–195 MB
- disk space (without HOL): 520 MB
- disk space (with HOL): 750 MB