Further Scaling of Isabelle Technology

Makarius Wenzel
http://sketis.net

February 2018

Abstract

Over 32 years, Isabelle has made a long way from a small experimental proof assistant to a versatile platform for proof document development and maintenance. There has always been a challenge to keep up with the natural growth of applications, notably the Archive of Formal Proofs (AFP). Can we scale this technology further, towards really big libraries of formalized mathematics? Can the underlying Scala/JVM and Poly/ML platforms cope with the demands? Can we eventually do 10 times more and better? I will revisit these questions from the perspectives of

1. Editing: Prover IDE,
2. Building: batch-mode tools and background services,
3. Browsing: HTML views and client-server applications.

Introduction

Isabelle — a 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

Isabelle technology

Established:

- Isabelle/ML: based on Poly/ML (by David Matthews)
- Isabelle/Scala: based on Java VM (by Oracle)
- Isabelle/jEdit: based on jEdit text editor (Java)
- Mercurial — source control management

Emerging:

- SQLite — db files (e.g. session build info)
- PostgreSQL — db server (e.g. nightly test data since 2002)

Experimental / Potential:

- VSCode — text editor based on Node.js + Chromium
- Scala.js (e.g. in HTML browsing or Node.js tools)

Isabelle applications: Archive of Formal Proofs

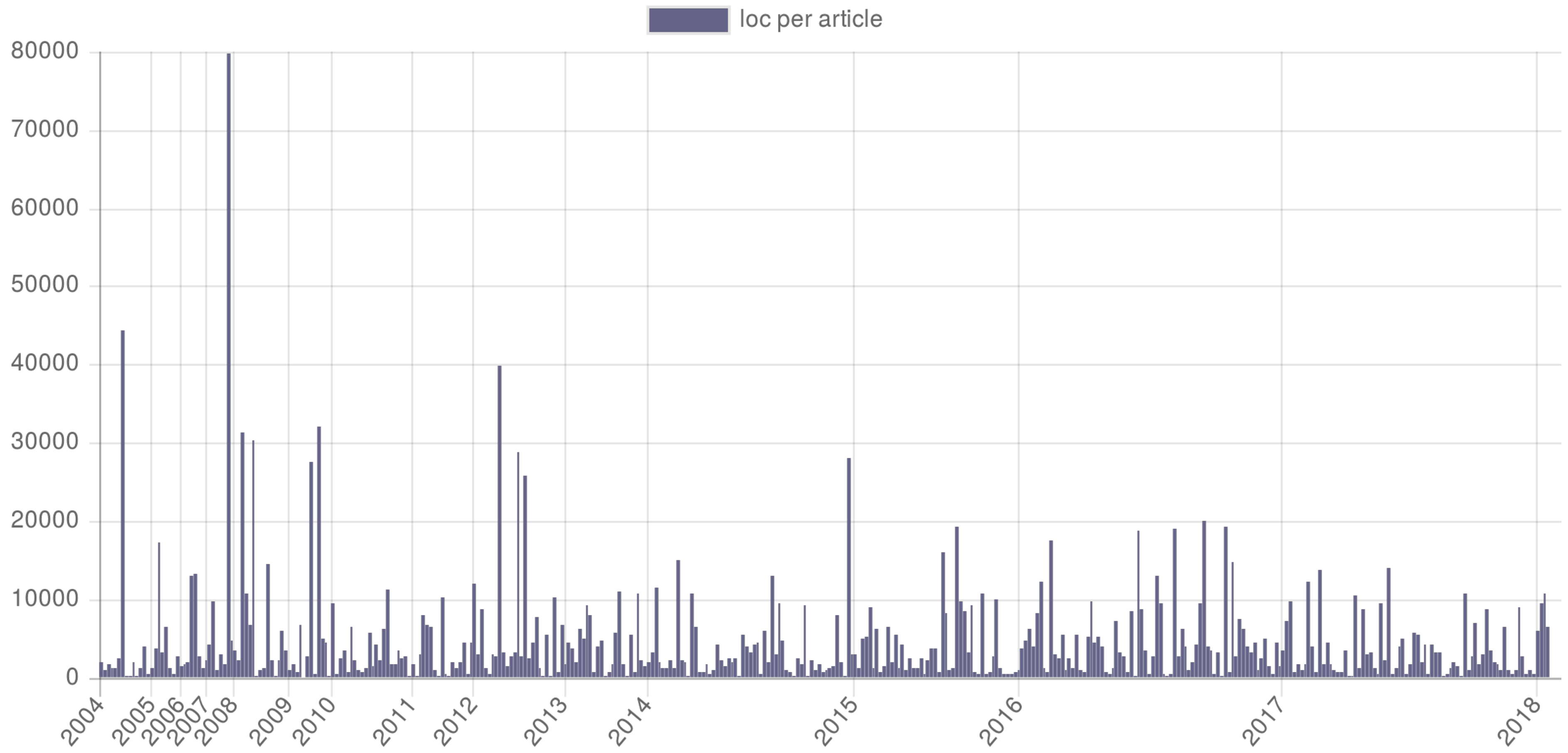
Statistics (29-Jan-2018) https://devel.isa-afp.org

- 275 authors
- 400 articles
- 4070 theories
- 10^5 theorems
- 10^8 bytes of text
- build time (non-slow sessions):
  - 23 h CPU time
  - 75 min elapsed time (factor 19 on 22 cores)

Time scales:

- Paris Commuter Constant: 45 min (online)
- French Lunch Time: 120 min (offline)

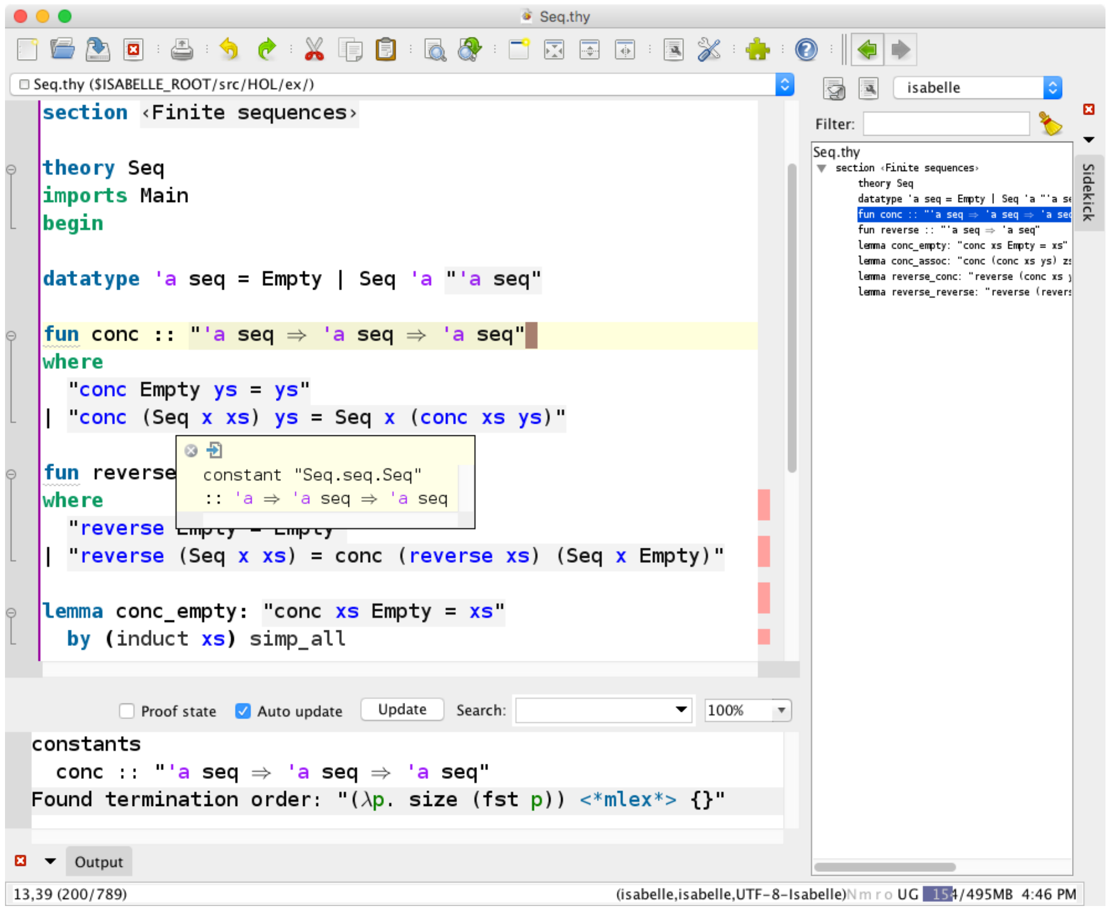
AFP articles and text size



Editing

Isabelle/jEdit Prover IDE (October 2017)

- asynchronous interaction
- continuous checking
- parallel processing



Seq.thy

Seq.thy ($ISABELLE_ROOT/src/HOL/ex/)

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

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

```
constant "Seq.seq.Seq"
:: 'a ⇒ 'a seq ⇒ 'a seq
```

```
Seq.thy
  section ‹Finite sequences›
    theory Seq
    datatype 'a seq = Empty | Seq 'a "'a se
    fun conc :: "'a seq ⇒ 'a seq ⇒ 'a seq
    fun reverse :: "'a seq ⇒ 'a seq"
    lemma conc_empty: "conc xs Empty = xs"
    lemma conc_assoc: "conc (conc xs ys) zs
    lemma reverse_conc: "reverse (conc xs y
    lemma reverse_reverse: "reverse (revers
```

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

```
13,39 (200/789)                    (isabelle,isabelle,UTF-8-Isabelle)Nmro UG 154/495MB 4:46 PM
```

Prover IDE components

jEdit: http://www.jedit.org

- sophisticated text editor implemented in Java

Scala/JVM: http://www.scala-lang.org

- higher-order functional-object-oriented programming

Isabelle/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

Further scaling of the Prover IDE

Aims:

- semantic editing of AFP as one big proof document
- continuous feedback for active sessions
- markup rendering for passive sessions

Approaches:

- GUI panels according to session graph structure (Sidekick, Hypersearch, Status with errors / warnings)
- support skipped proofs and forked proofs: more parallelism
- swapping PIDE markup: JVM heap vs. external database (SQLite, PostgreSQL)
- PIDE markup and edits vs. Mercurial changesets
- repository change management within the Prover IDE

Building

Isabelle/Scala build tools

`build`:

- explicit build of session images and PDF documents:

```
isabelle build -b HOL-Probability
isabelle build -b HOL-Probability -o document=pdf
```

- implicit build of PIDE session images:

```
isabelle jedit -S Deep_Learning -A HOL-Probability
isabelle jedit -S Deep_Learning -A HOL
```

`build_history`:

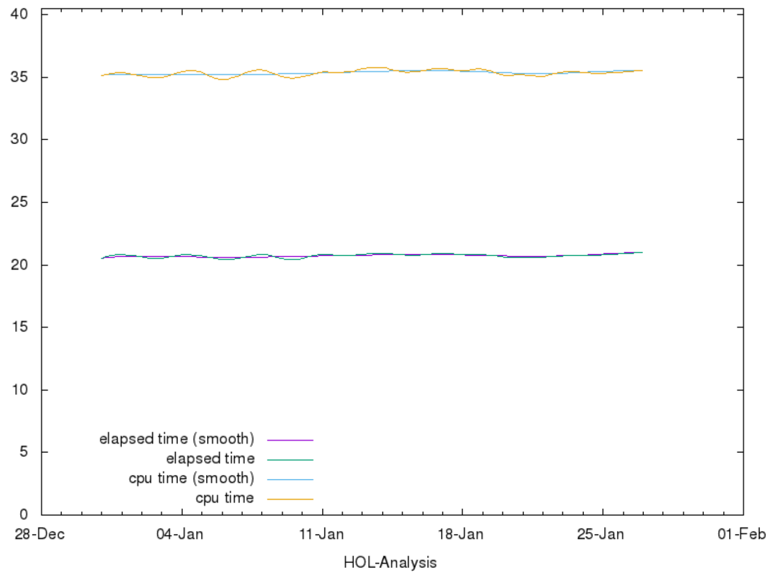
- build historic Isabelle + AFP versions
  (no need for "continuous integration")
- correlation of multicore builds (threads $= 1, 2, \ldots, n$)
- remote builds for multi-platform tests (via SSH)

`build_log`:

- management of build log files (historic formats since 2002)
- data storage: SQLite and PostgreSQL

`build_status`:

- reports on `build_log` database content
- performance charts



Further scaling of the build process

Technical approaches:

- skipping intermediate session images:
  - fork ML processes instead of heap load/store/load cycles
  - build multiple sessions within just one process
- transitions between batch-builds vs. PIDE sessions:
  - failed batch-build $\leadsto$ interactive editor session
  - successful editor session $\leadsto$ stored heap image
- advanced scheduling based on timings from `build_log` database
- distributed builds via SSH: multi-node parallelism
- reduced Poly/ML memory footprint on native 64 bit (emerging)

Browsing

Isabelle document structure

Markup

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

Markdown

- implicit paragraphs and lists: itemize, enumerate, description

Formal comments

- marginal comments: — ⟨*text*⟩
- canceled text: ***cancel*** ⟨*text*⟩ e.g. $\cancel{bad}$
- raw LaTeX: ***latex*** ⟨*text*⟩ e.g. $\lim_{n\to\infty} \sum_{i=0}^{n} q^i$

Document antiquotations

full form: `@{`$name$ $[options]$ $arguments$ $\ldots$`}`

short form:

1. cartouche argument: `\<^`$name$`>`$\langle argument\rangle$
2. no argument: `\<^`$name$`>`
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
- *action*: jEdit action (interaction)

HTML output

Existing:

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

Recent additions:

- support for document markdown structure

Future prospects:

- support for document markup structure
- support for document antiquotations
- high-quality HTML / CSS rendering, e.g. AFP as online Journal
- LaTeX-quality math typesetting (informal), e.g. via Khan/KaTeX
- interactive presentation, e.g. via `reveal.js`, `impress.js`

LaTeX output

Existing:

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

Recent additions:

- LaTeX errors and warnings with source positions
- IDE support for BibTeX databases

Future prospects:

- `pdflatex` runs within PIDE
- `pdflatex` runs after batch-build (using build db)
- use of semantic markup for document output

Further scaling of browsing

- offline document output from build database
  (PDF or HTML/CSS from SQLite db)
- online document output from headless PIDE process
  (HTML/CSS via SSH or Websocket)
- library browsing with semantic markup and high-end HTML:
  (e.g. with PostgreSQL database server for PIDE markup)
- separation of edits/markup protocol vs. display protocol
  (e.g. for web client)