Interaction with Formal Mathematical Documents in Isabelle/PIDE

Makarius Wenzel, Augsburg
https://sketis.net

July 2019

Summary

Isabelle/PIDE

- long-term effort to support live editing of complex document structures with "active" content
- most ambitious application: interactive theorem proving
- less demanding applications are easy, e.g. Isabelle/Naproche

Greater context:

- LCF/ML approach to interactive theorem proving (by Milner et-al)
- Isar approach to human-readable proof documents (by Wenzel)
- parallel ML and future proofs (by Matthews and Wenzel)
- early prover interfaces (by Aspinall, Bertot et-al)

$\longrightarrow$ forming a limit over decades of implementation-oriented research

Introduction

Isabelle

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

Example: Mathematical Documents

Cantor's Theorem states that there is no surjection from a set to its powerset. The proof works by diagonalization. E.g. see

- MathWorld: http://mathworld.wolfram.com/CantorDiagonalMethod.html
- Wikipedia: https://en.wikipedia.org/wiki/Cantor's_diagonal_argument
- Formal proof in Isabelle/Isar:

theorem Cantor: $\nexists f :: {}'a \Rightarrow {}'a \Rightarrow \mathit{bool}.\ \forall A.\ \exists x.\ A = f\ x$
proof
  assume $\exists f :: {}'a \Rightarrow {}'a \Rightarrow \mathit{bool}.\ \forall A.\ \exists x.\ A = f\ x$
  then obtain $f :: {}'a \Rightarrow {}'a \Rightarrow \mathit{bool}$ **where** $*$: $\forall A.\ \exists x.\ A = f\ x$ **..**
  let $?D = \lambda x.\ \neg\ f\ x\ x$
  from $*$ **have** $\exists x.\ ?D = f\ x$ **..**
  then obtain a **where** $?D = f\ a$ **..**
  then have $?D\ a \longleftrightarrow f\ a\ a$ **by** $(\mathit{rule\ arg\_cong})$
  then have $\neg\ f\ a\ a \longleftrightarrow f\ a\ a$ **.**
  then show False **by** $(\mathit{rule\ iff\_contradiction})$
qed

Interaction in PIDE

History:

- initial sketch at Dagstuhl, October 2009:
  "On prover interaction and integration with Isabelle/Scala"
  https://files.sketis.net/Dagstuhl2009.pdf
- recent overview at Dagstuhl, August 2018:
  "The Isabelle Prover IDE after 10 years of development"
  https://files.sketis.net/Dagstuhl2018.pdf
- cumulative complexity in concepts and implementation

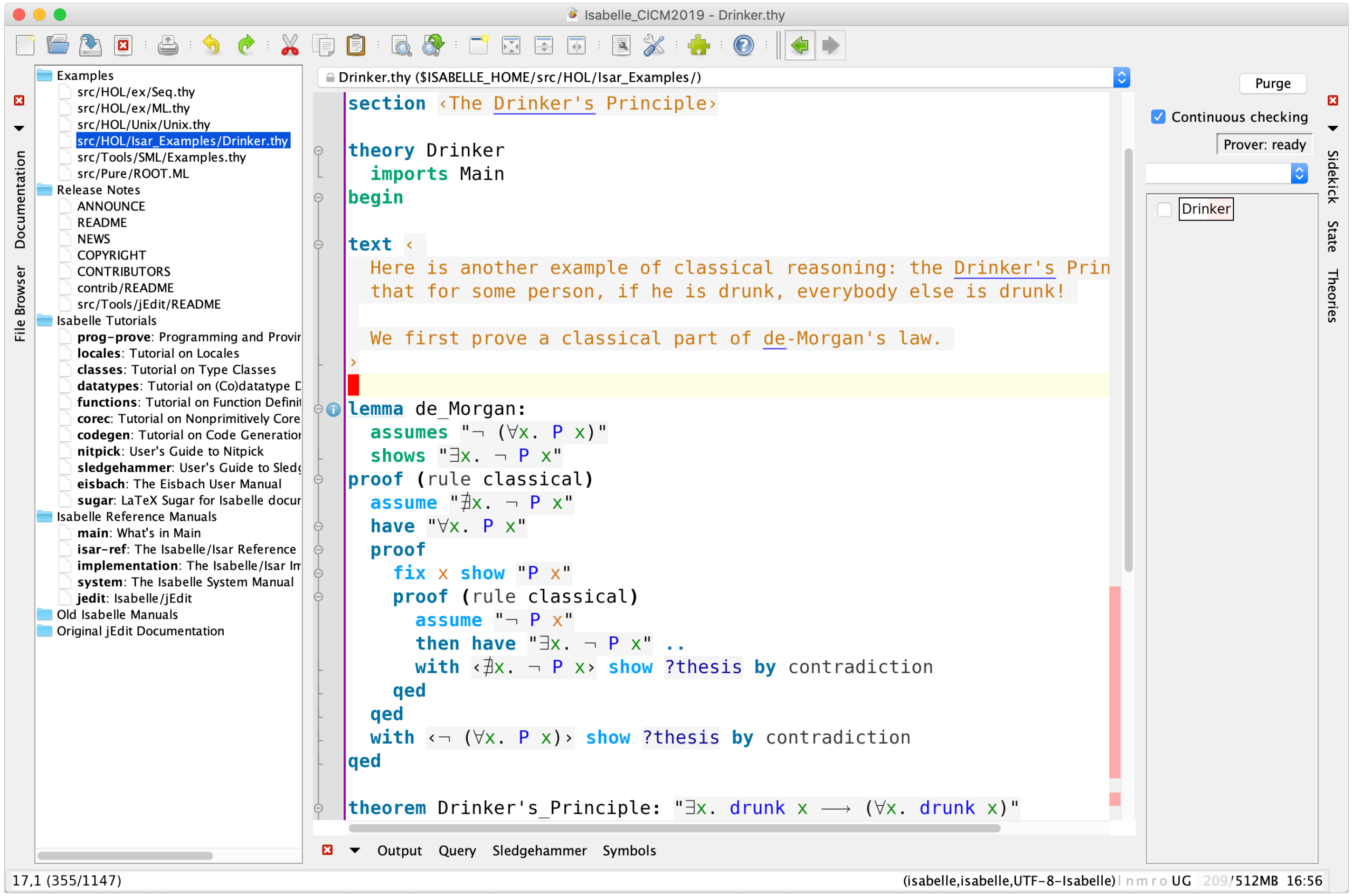
Architecture:

- inside the prover: Isabelle/ML back-end
- outside the prover: Isabelle/Scala front-end
- interaction via document edits vs. markup reports

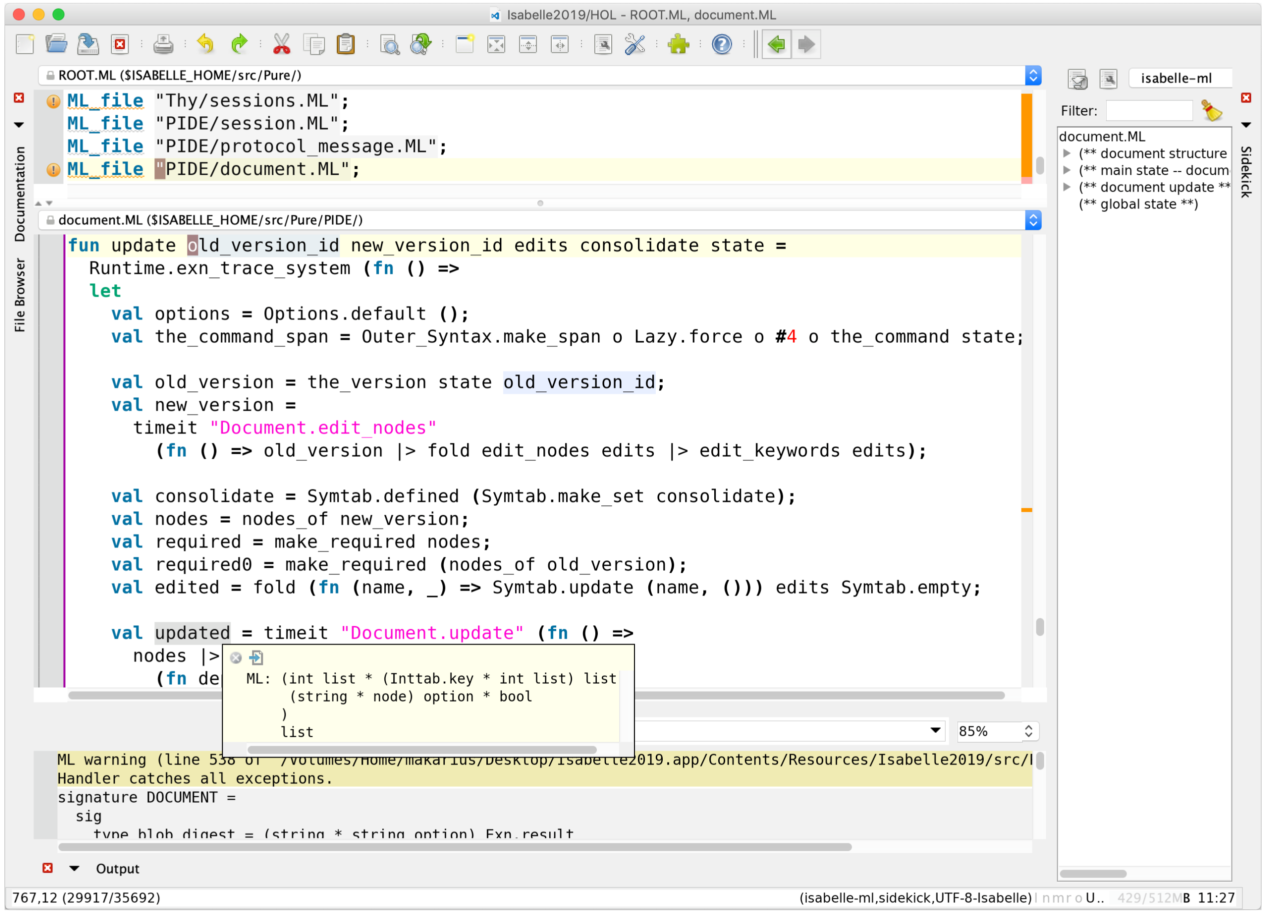
Notable applications

- Isabelle/jEdit Prover IDE (back-end: Isabelle theory processing)
  e.g. `$ISABELLE_HOME/src/HOL/Isar_Examples/Drinker.thy`
  e.g. `$ISABELLE_HOME/src/Doc/JEdit/JEdit.thy`
- Isabelle/jEdit ML IDE (back-end: Isabelle/ML run-time compiler)
  e.g. `$ISABELLE_HOME/src/Pure/ROOT.ML`
- Isabelle/jEdit BibTEX IDE (back-end: `bibtex`)
  e.g. `$ISABELLE_HOME/src/Doc/manual.bib`
- Isabelle/Naproche (back-end: `Naproche-SAD` server in Haskell)
  "Automatic Proof-Checking of Ordinary Mathematical Texts"
  (by Frerix and Koepke)

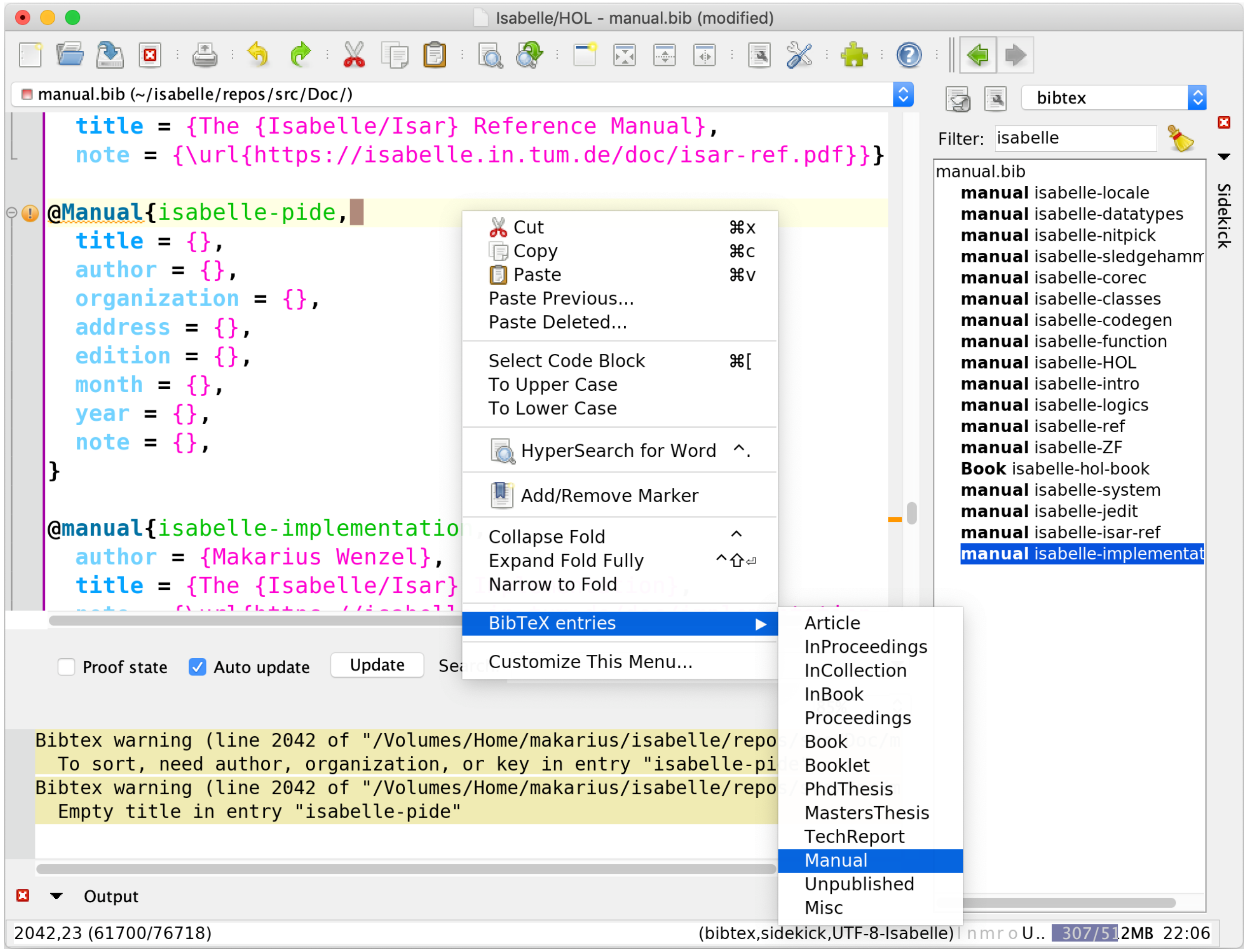
Screenshot: Isabelle/jEdit Prover IDE



Screenshot: Isabelle/jEdit ML IDE



Screenshot: Isabelle/jEdit BibTEX IDE



The PIDE Document Model

The PIDE Document Model

Main ideas:

- large expression of embedded sub-languages
- interactive exploration in the editor
- parallel processing by the prover
- prover/editor communication via asynchronous document edits
- document perspective determines focus of execution

Document content:

- theory sources: plain text
- auxiliary files: arbitrary blobs (usually plain text)
- output with semantic markup (untyped XML)
- output formatted as Oppen-style pretty trees

Document structure and organization

Theories:

- definition: e.g. **definition**, **inductive**, **primrec**
- statement: e.g. **lemma**, **function**, **termination**
- proof: Isar proof text (not "proof script")
- document outline: e.g. **chapter**, **section**, **text**

Note:

- proper foundational order of all entities
  (mutual recursion limited to single definition)
- implicit monotonic reasoning for derived elements

Sessions:

- acyclic sub-graph of imported theories (and other sessions)
- optional LaTeX document (generated by Isabelle)

Session exports

Main ideas:

- output of arbitrary blobs (analogous to auxiliary files)
- hierarchical name space (for each theory)
- virtual file-system `isabelle-export:` in Isabelle/jEdit
- stored within session database
- retrieved via `isabelle export` or `isabelle build -e`

Examples: generated sources

- **export_code** e.g. `~~/src/HOL/Quotient_Examples/Lift_Set.thy`
- **export_generated_files**, e.g. `~~/src/Tools/Haskell/Haskell.thy`
  command-line: `isabelle export -l Haskell`

Common syntax for embedded languages

Outer theory syntax:

- keywords: user-defined commands (e.g. **definition**, **inductive**)
- identifiers, numerals etc.
- quoted strings "$source$": nesting requires backslash-escapes
- cartouches $\langle source \rangle$: arbitrary nesting without no escapes

Example:

ML $\langle val\ t = \boldsymbol{term}\langle \lambda x.\ x \le y + z$ — comment in term$\rangle$ — comment in ML$\rangle$

Aims and Approaches of Isabelle/PIDE

Isabelle/ML vs. Isabelle/Scala (1)

- Isabelle/ML (based on Poly/ML): "pure mathematics"
- Isabelle/Scala (based on Java 11 platform): "real physics"

Success:

- clean and efficient (parallel) functional programming on both sides
- minimality / purity of the library, overlap of modules on both sides
- manual migration / translation of modules on demand

Failure:

- Isabelle/ML perceived as difficult for many users
- Isabelle/Scala perceived as inaccessible for most users

Isabelle/ML vs. Isabelle/Scala (2)

Changes:

- Isabelle/Scala has grown in importance over the year: integral part of Isabelle, not just add-on
- Isabelle/Scala code base has similar size as Isabelle/ML/Pure

Future:

- proper IDE support for Isabelle/Scala (e.g. IntelliJ instead of Isabelle/Scala/PIDE itself)

Private protocol vs. public API (1)

- PIDE protocol: untyped messages between prover and editor (blobs, XML/YXML)
- PIDE APIs: typed interfaces in ML and Scala (e.g. messages with logical markup and Oppen-style pretty trees)

Success:

- efficient and robust implementation of bi-lingual PIDE
- easy maintenance of corresponding modules in same directory

Failure:

- alternative PIDE prover implementation difficult to maintain (e.g. PIDE/Coq remains unfinished)

Private protocol vs. public API (2)

Changes:

- PIDE protocol started plain and simple, but has become complex (e.g. for scaling, add-on features)

Future:

- re-open old idea to retarget PIDE, e.g. for Coq (??)
- addition display protocol for PIDE front-end, e.g. for web interface

Pervasive parallelism on multicore hardware (1)

- routine support for shared-memory multiprocessing in Isabelle/ML (and Isabelle/Scala)
- low-level POSIX threads/locks or high-level future values

Success:

- parallel Isabelle/ML works well since 2008, with increasing stability and scalability; 8–16 cores for parallel theory and proof checking

Failure:

- stagnation of the multicore market: light-weight mobile devices (2–8 cores) vs. high-end servers (32–128 cores)
- high-end machines are often clusters of low-end CPUs, e.g. 64 hardware threads $=$ 8 cores $\times$ 8 nodes (NUMA)

Pervasive parallelism on multicore hardware (2)

Future:

- maybe follow the trend towards "cloud computing", e.g. local Isabelle/jEdit or Isabelle/VSCode editor (not web browser interface)
- further refinement of Headless PIDE server

Multi-platform desktop application bundles (1)

- support for mainstream platforms: Linux, Windows, macOS
- no self-assembly by users
- no re-packaging by OS developers (e.g. Debian)
- no support for exotic platforms (e.g. BSD, Solaris, NixOS)

Success:

- all-inclusive Isabelle (1 GB unpacked) just works for most users
- "download–unpack–run" comparable to e.g. Firefox, LibreOffice

Failure:

- OS non-uniformity: varying GUI quality and external tool stability
- OS malware protection hinders external tools
- OS vendors tend to reject non-registered applications

Multi-platform desktop application bundles (2)

Changes:

- early deployment was too optimistic about fragile dependencies (e.g. Java, Scala)
- almost everything is now bundled (similar to SageMath)
- few implicit dependencies: e.g. `libc`, `libc++`, `curl`, `perl`

Future:

- better integration of the Archive of Formal Proofs (AFP)
- better support for derived application bundles, e.g. Isabelle/MMT, Isabelle/Naproche

Application: Isabelle/Naproche

Automatic Proof-Checking of Ordinary Mathematical Texts

Naproche-SAD: 2017/2018

- Steffen Frerix and Peter Koepke (Bonn): reworked and extended version of SAD by Andrei Paskevich (LRI, Paris Sud)
- ForTheL (Formal Theory Language): restricted subset of mathematical jargon
- based on First-Order Logic and Classic Set-Theory
- automated reasoning via E Prover (Stephan Schulz)
- Haskell implementation: command-line tool, sequential function from input files to informal output messages

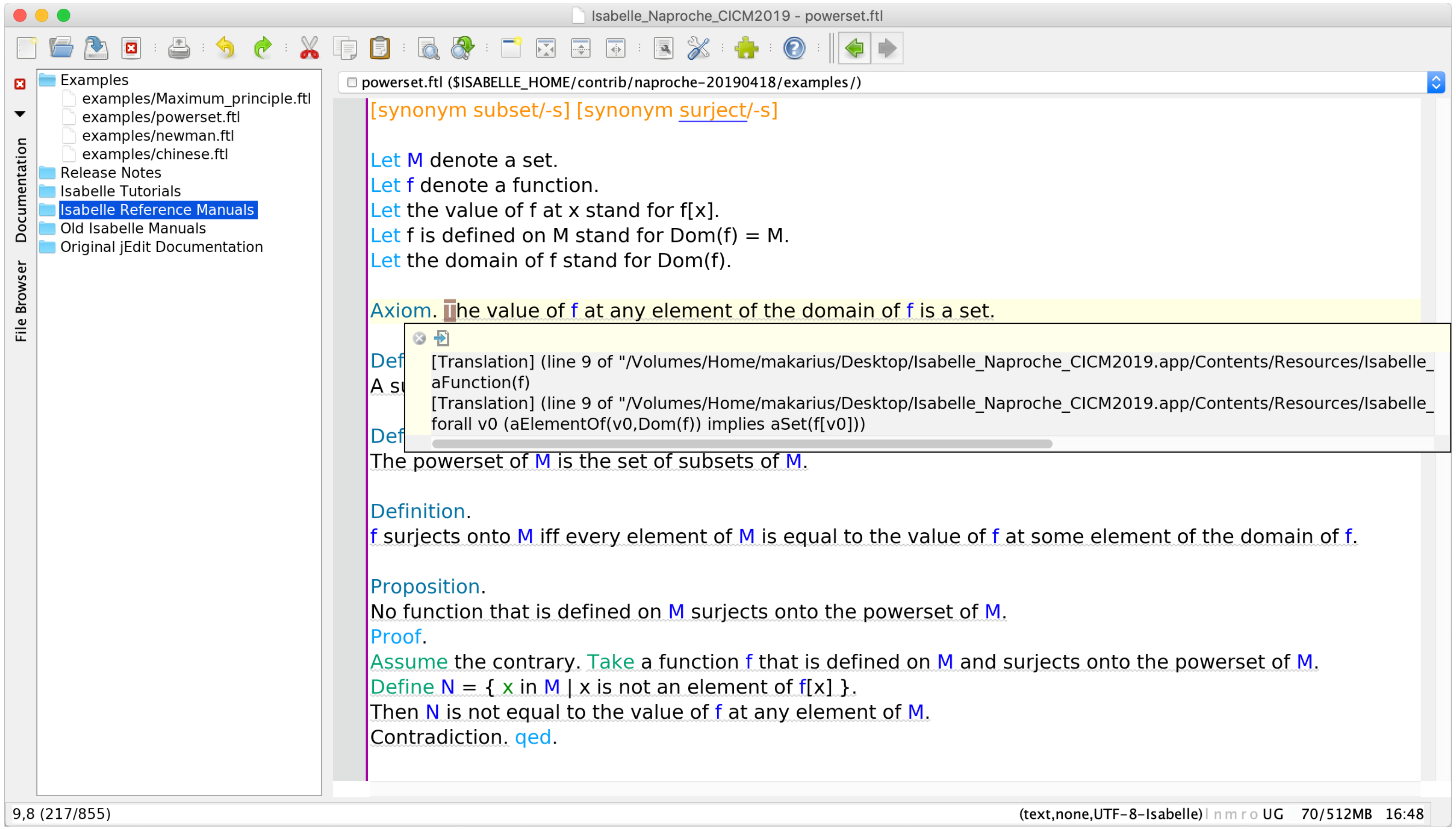
Isabelle/Naproche: 2018

- Haskell implementation: TCP server with cached blocks of text, reactive function from input text to formal output messages
- based on general Isabelle/Haskell library for Isabelle/PIDE (new in Isabelle2019)
- Isabelle/Scala add-on to register `.ftl` as auxiliary file format with implicit theory context (new in Isabelle2019)
- derived application bundling and branding as Isabelle/Naproche

Corollary:

- Isabelle applications are not necessarily tied to Isabelle/HOL (nor Isabelle/Pure)
- further PIDE applications in Haskell will be easy to implement

Screenshot: Isabelle/Naproche



Conclusion

Future work (after 11 years of PIDE)

PIDE technology:

- dynamic session management
- dynamic PDF-LaTeX document preparation
- real-time HTML/CSS preview, approaching LaTeX quality

PIDE sociology:

- better visibility outside of Isabelle community
- more re-use of the Isabelle/PIDE platform