Formal Docs

Documentation Practices Across Theorem Provers and Model Checkers

Scope of the Survey

- Review documentation of software for model checking/theorem proving
- Identify what kinds of documentation there is
    - User guide, API reference, Tutorials/Resources/Cookbook, Contributing/Developer guide
- Identify whether documentation is official, unofficial or mixed
- Identify tools used to generate, edit and/or host documentation

Alloy - https://alloy.readthedocs.io/en/latest/

- Runs on Sphinx
  - Some customization
- Unofficial
- Meant to be a reference, not a tutorial

CONTENTS:

Signatures

A **signature** expresses a new type in your spec. It can be anything you want. Here are some example signatures:

- Time
- State
- File
- Person
- Msg
- Pair

Alloy can generate models that have elements of each signature, called **atoms**. Take the following spec:

```
sig A {}
```

The following would be an example generated model:

SPIN - https://spinroot.com/spin/whatispin.html

- Basic HTML
- Not clear how to add content, dated
- Course materials - https://spinroot.com/course/
- Large number of books, course materials
- Github not very active

https://spinroot.com/spin/whatispin.html

Spin is a widely used open-source software verification tool. The tool can be used for the formal verification of multi-threaded software applications. The tool was developed at Bell Labs in the Unix group of the Computing Sciences Research Center, starting in 1980, and has been available freely since 1991. Spin continues to evolve to keep pace with new developments in the field. In April 2002 the tool was awarded the ACM System Software Award. `[read more]`

discover

- what is spin?
- success stories
- examples
- roots

learn

- tutorials
- books
- papers
- model extraction
- exercises

use

- installation
- man pages
- options
- releases

community

- forum
- symposia
- support
- projects

Open Source: Starting with Version 6.4.5 from January 2016, the Spin sources are available under the standard BSD 3-Clause open source license. Spin is now also part of the latest stable release of Debian Linux, and has made it into the 16.10+ distributions of Ubuntu. The current Spin version is 6.5.1 (July 2020).

Symposia: The 29th International Spin Symposium will be held in April 26-27 2023 in Paris, co-located with ETAPS-2023. The Symposium is organized by Georgiana Caltais and Christian Schilling.

Courses: A short online course in software verification and logic model checking is available (password required). There are a total 15 short lectures covering the automata-theoretic verification method, the basic use of Spin, model extraction from C source code, abstraction methods, and swarm verification techniques. You can see an overview via this link. An excellent introduction to the basics of model checking.

In-Depth: A full one semester college-level course is also available, complete with transcripts of every lecture, quizzes, assignments, and exercises to test your understanding and practice new skills. Details can be found in this syllabus.

Coq - https://coq.inria.fr/refman/index.html

- Sphinx for Reference Manual
- Just uses markdown for Contributing
    - *“Our official resources, such as the reference manual are not suited for learning Coq, but serve as reference documentation to which you can link from your tutorials.”*
- Wiki - Installation, Development, Additional Resources
    - *“Coq's wiki is an informal source of additional documentation which anyone with a GitHub account can edit directly.”*

Introduction and Contents

This is the reference manual of Coq. Coq is an interactive theorem prover. It lets you formalize mathematical concepts and then helps you interactively generate machine-checked proofs of theorems. Machine checking gives users much more confidence that the proofs are correct compared to human-generated and -checked proofs. Coq has been used in a number of flagship verification projects, including the CompCert verified C compiler, and has served to verify the proof of the four color theorem (among many other mathematical formalizations).

Users generate proofs by entering a series of tactics that constitute steps in the proof. There are many built-in tactics, some of which are elementary, while others implement complex decision procedures (such as `lia`, a decision procedure for linear integer arithmetic). Ltac and its planned replacement, Ltac2, provide languages to define new tactics by combining existing tactics with looping and conditional constructs. These permit automation of large parts of proofs and sometimes entire proofs. Furthermore, users can add novel tactics or functionality by creating Coq plugins using OCaml.

The Coq kernel, a small part of Coq, does the final verification that the tactic-generated proof is valid. Usually the tactic-generated proof is indeed correct, but delegating proof verification to the kernel means that even if a tactic is buggy, it won't be able to introduce an incorrect proof into the system.

Finally, Coq also supports extraction of verified programs to programming languages such as OCaml and Haskell. This provides a way of executing Coq code efficiently and can be used to create verified software libraries.

Lean - https://leanprover-community.github.io/

- Custom doc-gen
  - https://github.com/leanprover-community/doc-gen
- Very clear “Getting Started” section
- Thorough guidelines on contributing
- Clean looking

Learning Lean

There are many ways to start learning Lean, depending on your background and taste. They are all fun and rewarding, but also difficult and occasionally frustrating. Proof assistants are still difficult to use, and you cannot expect to become proficient after one afternoon of learning.

Hands-on approaches

- Whatever your background, if you want to dive right away, you can play the Natural Number Game by Kevin Buzzard and Mohammad Pedramfar. This is a online interactive tutorial to Lean focused on proving properties of the elementary operations on natural numbers.
- For a faster paced and broader dive, you can get the tutorials project. (You already have it if you installed an autonomous bundle or followed the instructions on this page.) This tutorial is specifically geared towards mathematics rather than computer science. The last files of this project are easier if you have already encountered the definition of limits of sequences of real numbers.
- The lfctm2020 exercises, developed for the July 2020 virtual meeting Lean for the Curious Mathematician, are another good resource. There are corresponding tutorial videos from the meeting.
- A brand new resource that is still under construction is *Mathematics in Lean*. It can be read online, or downloaded as a pdf, but it is really meant to be used in VSCode, doing exercises on the fly (see the instructions). It currently covers roughly the same ground as the tutorials project.
- Once you know the basics, you can also learn by solving Lean puzzles on Codewars.

Whatever resource you choose to use from the above list, it could be useful to have a copy of our tactic cheat sheet at hand, for reference.

Isabelle - https://isabelle.in.tum.de/documentation.html

- Home
  - HTML links to PDFs
  - Official
  - Unclear how to contribute
- https://isabelle.systems/
  - Github Pages
  - Links to Home, Cookbook
- Community Cookbook
  - Github Pages

Documentation

Tutorials and manuals for Isabelle2022

Isabelle Tutorials

- **prog-prove**: Programming and Proving in Isabelle/HOL
- **locales**: Tutorial on Locales
- **classes**: Tutorial on Type Classes
- **datatypes**: Tutorial on (Co)datatype Definitions
- **functions**: Tutorial on Function Definitions
- **corec**: Tutorial on Nonprimitively Corecursive Definitions
- **codegen**: Tutorial on Code Generation
- **nitpick**: User's Guide to Nitpick
- **sledgehammer**: User's Guide to Sledgehammer
- **eisbach**: The Eisbach User Manual
- **sugar**: LaTeX Sugar for Isabelle documents

Isabelle Reference Manuals

- **main**: What's in Main
- **isar-ref**: The Isabelle/Isar Reference Manual
- **implementation**: The Isabelle/Isar Implementation Manual
- **system**: The Isabelle System Manual
- **jedit**: Isabelle/jEdit

Old Isabelle Manuals

- **tutorial**: Tutorial on Isabelle/HOL
- **intro**: Old Introduction to Isabelle
- **logics**: Isabelle's Logics: HOL and misc logics
- **logics-ZF**: Isabelle's Logics: FOL and ZF

Z3 - https://z3prover.github.io/api/html/

- API reference docs
  - Madoko
  - Auto-generated
- Guide
  - Docusaurus
  - Free-form editor (but it has a bug!)

Z3

An Efficient Theorem Prover

Z3 is a high-performance theorem prover being developed at Microsoft Research.

The Z3 website is at http://github.com/z3prover.

This website hosts the automatically generated documentation for the Z3 APIs.

- C API
- C++ API
- .NET API
- Java API
- Python API (also available in pydoc format)
- ML/OCaml API

AGREE - https://loonwerks.com/tools/agree.html

- PanDocs
    - Uses ANT for its build process
- https://github.com/loonwerks/AGREE
    - *“The documentation source code is maintained in Markdown from which HTML, PDF, and DOCX output is generated.”*
- Docs not available online

Takeaways

- No general consensus among documentation tools
  - Sphinx has a plurality, but almost all are different
- Combination of approaches is common
  - “Official” homepage supported by “unofficial” ecosystem
- It’s not a beauty contest
  - General focus on content, not styling

Thank you!