The Journal of Open Source Software

The Agda standard library: version 2.0

Matthew L. Daggitt [1¶], **Guillaume Allais** [2], **James McKinna** [3], **Andreas Abel** [4], **Nathan van Doorn** [5], **James Wood** [6], **Ulf Norell** [4], **Donnacha Oisín Kidney** [7], **Sergei Meshveliani** [8], **Sandro Stucki** [4], **Jacques Carette** [9], **Alex Rice** [10], **Jason Z. S. Hu** [11], **Li-yao Xia** [12], **Shu-Hung You** [13], **Reed Mullanix** [9], **and Wen Kokke** [10]

1 University of Western Australia, Australia **2** University of Strathclyde, United Kingdom **3** Heriot-Watt University, United Kingdom **4** University of Gothenburg and Chalmers University of Technology, Sweden **5** Independent Software Developer **6** Huawei Technologies Research & Development, United Kingdom **7** Imperial College London, United Kingdom **8** Russian Academy of Sciences, Russia **9** McMaster University, Canada **10** University of Edinburgh, United Kingdom **11** Amazon, USA **12** INRIA, France **13** Northwestern University, USA ¶ Corresponding author

DOI: 10.21105/joss.09241

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @dorchard
- @vidsinghal
- @benmandrew

Submitted: 23 September 2025
Published: 20 December 2025

Summary

Agda (The Agda Development Team, 2024) is a dependently-typed functional language that serves as both a programming language and an interactive theorem prover (ITP). In Agda, one can formulate requirements on programs as types and build programs satisfying these requirements interactively. The Curry-Howard correspondance (Wadler, 2015) allows types and programs to be seen as theorems and proofs. We present the Agda standard library (The Agda community, 2023) (`agda-stdlib`), which provides functions and mathematical concepts helpful in the development of both programs and proofs.

Statement of need

Besides providing common utilities and data structures, `agda-stdlib` is especially necessary compared to standard libraries for traditional languages for two reasons.

First, Agda is a small, powerful language that omits concepts usually built-in to a language (e.g., numbers, strings). This reduces compiler complexity, but leaves `agda-stdlib` to define them.

Second, functions in `agda-stdlib` come with correctness proofs - these require substantial work that should not fall to users.

Impact

A diverse set of verification projects use `agda-stdlib`, including:

- Programming Language Foundations in Agda (Wadler et al., 2022)
- Category theory (Hu & Carette, 2021)
- Scala's type system (Stucki & Giarrusso, 2021)
- Calculus for the Esterel language (Florence et al., 2019)
- Hardware circuit design (Pizani Flor et al., 2018)
- Routing protocols (Daggitt & Griffin, 2023)

The library has had a synergistic relationship with Agda itself, both testing and motivating new language features. For example, since Agda supports many incompatible language extensions, `agda-stdlib` is structured modularly to remain compatible with different combinations of extensions. Each module requests only the minimal expressive power it needs and to facilitate this Agda now categorises extensions as "infective" (affecting all import*ing* modules), "coinfective" (affecting all import*ed* modules), or "neither". The library has also served as a test bed for alternative approaches to defining co-inductive data types in Agda.

Design

Organising libraries of discrete mathematics and algebra coherently is notoriously difficult (Carette et al., 2020; Cohen et al., 2020). There is a tension between maximising generality and providing direct, intuitive definitions. Mathematical objects often admit multiple representations with different benefits, but this leads to redundancy. Some ITPs ((Paulson, 1994; The Rocq Development Team, 2025)) have a rich ecosystem of external libraries, avoiding canonical definitions at the cost of incompatibilities. We have chosen, like Lean's `mathlib` (van Doorn et al., 2020), to provide a repository of canonical definitions.

`agda-stdlib` adopts the "intrinsic style" of dependent types, where data structures themselves contain correctness invariants. For examples, rational numbers carry a proof that the numerator and denominator are coprime and decision procedures return proofs rather than booleans. To our knowledge, `agda-stdlib` is among the first ITP standard libraries to whole-heartedly embrace this style of programming.

In contrast to the type-class mechanisms often used by other functional languages, `agda-stdlib` primarily supports polymorphism (de Bruin, 2023) via extensive use of parametrised modules. This allows users to specify instantiations of abstract parameters for whole modules in a single location, reducing the need for instance search. A drawback is that imports must be qualified when code is instantiated multiple times in the same scope. Parameterised modules are also used to safely and scalably embed non-constructive mathematics into a constructive setting.

The README directory within the library contains both documentation on the general design decisions and examples of how to use the most common modules in the library to prove basic concepts. There are many excellent tutorials online that introduce both Agda and `agda-stdlib` together, with "Programming Language Foundations in Agda'' (Wadler, 2018) being an example of one such tutorial.

Testing

Correctness proofs do not remove the need for testing performance and features that cannot be reasoned about internally (such as the FFI and macros). However, the test suite's coverage is incomplete as this is not a community priority.

Version 2.0

Version 2.0 of `agda-stdlib` (The Agda community, 2023) has attempted to address some of the design flaws and missing functionality of previous versions, including:

- Minimised Dependency Graphs: core modules rely on fewer parts of the library, resulting in faster load times.
- Standardisation: mathematical objects and their morphisms (e.g., groups, rings) are now constructed more uniformly, enhancing consistency and usability.
- Tactics Library: expanded the set of available tactics (although performance can still be improved).

- Testing Framework: introduced a golden testing framework to let users write their own test suites.

Acknowledgements

Nils Anders Danielsson provided substantial feedback.

Authors are listed approximately in order of contribution. Manuscript by Daggitt, Allais, McKinna, Carette, and van Doorn. A list of all contributors is available on GitHub.

Funding and conflicts of interest

The authors have no conflicts of interest. Some contributions were enabled by funding for related projects:

- Shu-Hung You: U.S. National Science Foundation Awards 2237984 and 2421308.
- Jason Z. S. Hu: Postgraduate Scholarship (Doctoral) from the Natural Sciences and Engineering Research Council of Canada and Doctoral (B2X) Research Scholarship from Fonds de recherche du Québec (Nature et technologies).

References

Carette, J., Farmer, W. M., & Sharoda, Y. (2020). Leveraging the information contained in theory presentations. *Intelligent Computer Mathematics: 13th International Conference, CICM 2020, Bertinoro, Italy, July 26–31, 2020, Proceedings*, 55–70. https://doi.org/10.1007/978-3-030-53518-6_4

Cohen, C., Sakaguchi, K., & Tassi, E. (2020). Hierarchy Builder: Algebraic Hierarchies made easy in Coq with Elpi. *FSCD 2020-5th International Conference on Formal Structures for Computation and Deduction*, 34–31. https://doi.org/10.4230/LIPIcs.FSCD.2020.34

Daggitt, M. L., & Griffin, T. G. (2023). Formally verified convergence of policy-rich DBF routing protocols. *IEEE/ACM Transactions on Networking*, *32*(2), 1645–1660. https://doi.org/10.1109/TNET.2023.3326336

de Bruin, I. C. (2023). *Improving Agda's module system* [Master's thesis, Delft University of Technology]. https://resolver.tudelft.nl/uuid:98b8fbf5-33f0-4470-88b0-39a9d526b115

Florence, S. P., You, S.-H., Tov, J. A., & Findler, R. B. (2019). A calculus for Esterel: If can, can. If no can, no can. *Proc. ACM Program. Lang.*, *3*(POPL). https://doi.org/10.1145/3290374

Hu, J. Z. S., & Carette, J. (2021). Formalizing category theory in Agda. *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*, 327–342. https://doi.org/10.1145/3437992.3439922

Paulson, L. C. (1994). *Isabelle: A generic theorem prover*. Springer. https://doi.org/10.1007/BFb0030541

Pizani Flor, J. P., Swierstra, W., & Sijsling, Y. (2018). Pi-Ware: Hardware description and verification in Agda. In T. Uustalu (Ed.), *21st international conference on types for proofs and programs (TYPES 2015)* (Vol. 69, pp. 9:1–9:27). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.TYPES.2015.9

Stucki, S., & Giarrusso, P. G. (2021). A theory of higher-order subtyping with type intervals. *Proceedings of the ACM on Programming Languages*, *5*(ICFP), 69:1–69:30. https://doi.org/10.1145/3473574

The Agda community (Ed.). (2023). *The Agda standard library, version 2.0*. HTML-indexed sources also at: https://agda.github.io/agda-stdlib/v2.0/. https://github.com/agda/agda-stdlib/tree/v2.0-release

The Agda Development Team. (2024). *The Agda manual – release 2.7.0.1*. https://agda.readthedocs.io/en/v2.7.0.1/.

The Rocq Development Team. (2025). *The Rocq reference manual – release 9.0.0*. https://rocq-prover.org/doc/V9.0.0/refman/index.html.

van Doorn, F., Ebner, G., & Lewis, R. Y. (2020). Maintaining a library of formal mathematics. In C. Benzmüller & B. Miller (Eds.), *Intelligent computer mathematics* (pp. 251–267). Springer International Publishing. https://doi.org/10.1007/978-3-030-53518-6_16

Wadler, P. (2018). Programming language foundations in Agda. *Brazilian Symposium on Formal Methods*, 56–73.

Wadler, P. (2015). Propositions as types. *Commun. ACM*, *58*(12), 75–84. https://doi.org/10.1145/2699407

Wadler, P., Kokke, W., & Siek, J. G. (2022). *Programming language foundations in Agda*. https://doi.org/10.1016/j.scico.2020.102440