




# The Agda standard library: version 2.0

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

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

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

## Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Daniel S. Katz](#) 

## Reviewers:

- [@dorchard](#)
- [@vidsinghal](#)
- [@benmandrew](#)

Submitted: 23 September 2025

Published: unpublished

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

## 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 correspondence ([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](#))

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

## 44 Design

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

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

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

## 63 Testing

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

## 67 Version 2.0

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

- 70     ▪ Minimised Dependency Graphs: core modules rely on fewer parts of the library, resulting  
71       in faster load times.
- 72     ▪ Standardisation: mathematical objects and their morphisms (e.g. groups, rings) are now  
73       constructed more uniformly, enhancing consistency and usability.
- 74     ▪ Tactics Library: expanded the set of available tactics (although performance can still be  
75       improved).
- 76     ▪ Testing Framework: introduced a golden testing framework to let users write their own  
77       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:

- Jason Z. S. Hu: funded Master's/PhD.
- Shu-Hung You: U.S. National Science Foundation Awards 2237984 and 2421308.

## 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](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:98b8bf5-33f0-4470-88b0-39a9d526b115>
- 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/>.

- 120 The Rocq Development Team. (2025). *The Rocq reference manual – release 9.0.0*. [https://](https://rocq-prover.org/doc/V9.0.0/refman/index.html)  
121 [rocq-prover.org/doc/V9.0.0/refman/index.html](https://rocq-prover.org/doc/V9.0.0/refman/index.html).
- 122 van Doorn, F., Ebner, G., & Lewis, R. Y. (2020). Maintaining a library of formal mathematics.  
123 In C. Benzmüller & B. Miller (Eds.), *Intelligent computer mathematics* (pp. 251–267).  
124 Springer International Publishing. [https://doi.org/10.1007/978-3-030-53518-6\\_16](https://doi.org/10.1007/978-3-030-53518-6_16)
- 125 Wadler, P. (2015). Propositions as types. *Commun. ACM*, 58(12), 75–84. [https://doi.org/](https://doi.org/10.1145/2699407)  
126 [10.1145/2699407](https://doi.org/10.1145/2699407)
- 127 Wadler, P., Kokke, W., & Siek, J. G. (2022). *Programming language foundations in Agda*.  
128 <https://doi.org/10.1016/j.scico.2020.102440>

DRAFT