Not logged in Talk Contributions Create account Log in
Article Talk Read Edit View history Search Wikipedia
Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate
Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file
Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item
Print/export
Download as PDF
Printable version
In other projects
Wikimedia Commons
Wikibooks
Wikiquote
Wikiversity
Languages
Deutsch
Español
Français
한국어
हिन्दी
Italiano
Русский
Tiếng Việt
中文
42 more
Edit links

Haskell

From Wikipedia, the free encyclopedia
(Redirected from Haskell (programming language))

For other uses, see Haskell (disambiguation).

Haskell (/ˈhæskəl/[25]) is a general-purpose, statically-typed, purely functional programming language with type inference and lazy evaluation.[26][27] Designed for teaching, research and industrial applications, Haskell has pioneered a number of programming language features such as type classes, which enable type-safe operator overloading. Haskell's main implementation is the Glasgow Haskell Compiler (GHC). It is named after logician Haskell Curry.[1]

Haskell's semantics are historically based on those of the Miranda programming language, which served to focus the efforts of the initial Haskell working group.[28] The last formal specification of the language was made in July 2010, while the development of GHC continues to expand Haskell via language extensions.

Haskell is used in academia and industry.[29][30][31] As of May 2021, Haskell was the 28th most popular programming language by Google searches for tutorials,[32] and made up less than 1% of active users on the GitHub source code repository.[33]

Haskell

Paradigm	Purely functional
Designed by	Lennart Augustsson, Dave Barton, Brian Boutel, Warren Burton, Joseph Fasel, Kevin Hammond, Ralf Hinze, Paul Hudak, John Hughes, Thomas Johnsson, Mark Jones, Simon Peyton Jones, John Launchbury, Erik Meijer, John Peterson, Alastair Reid, Colin Runciman, Philip Wadler
First appeared	1990; 32 years ago[1]
Stable release	Haskell 2010[2] / July 2010; 12 years ago
Preview release	Haskell 2020 announced[3]
Typing discipline	Inferred, static, strong
OS	Cross-platform
Filename extensions	.hs, .lhs
Website	www.haskell.org

Major implementations
GHC, Hugs, NHC, JHC, Yhc, UHC

Dialects
Gofer

Influenced by
Clean,[4] FP,[4] Gofer,[4] Hope and Hope+,[4] Id,[4] ISWIM,[4] KRC,[4] Lisp,[4] Miranda,[4] ML and Standard ML,[4] Orwell, SASL,[4] Scheme,[4] SISAL[4]

Influenced
Agda,[5] Bluespec,[6] C++11/Concepts,[7] C#/LINQ,[8][9][10][11] CAL,[citation needed] Cayenne,[8] Clean,[8] Clojure,[12] CoffeeScript,[13] Curry,[8] Elm, Epigram,[citation needed] Escher,[14] F#,[15] Hack,[16] Idris,[17] Isabelle,[8] Java/Generics,[8] LiveScript,[18] Mercury,[8] Ωmega, PureScript,[19] Python,[8][20] Raku,[21] Rust,[22] Scala,[8][23] Swift,[24] Visual Basic 9.0[8][9]

Contents [hide]

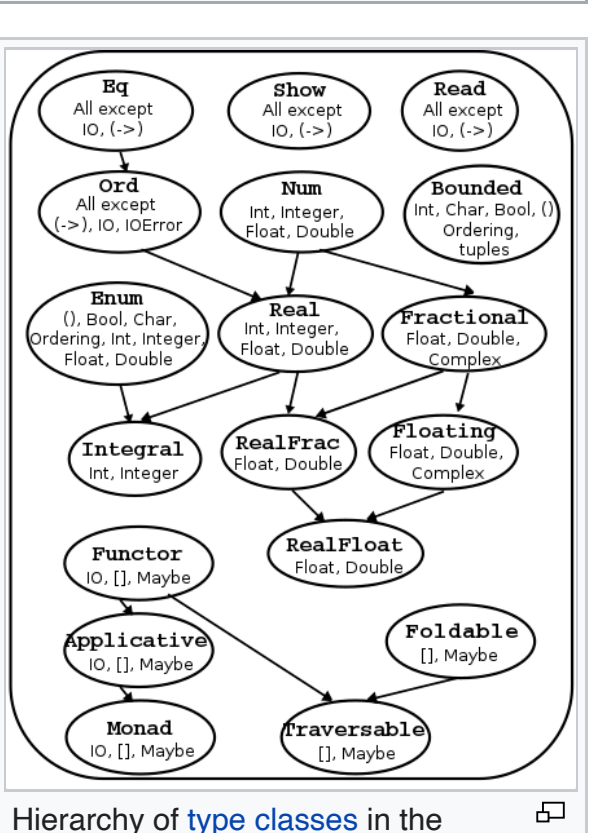
History [edit]

Following the release of Miranda by Research Software Ltd. in 1985, interest in lazy functional languages grew. By 1987, more than a dozen non-strict, purely functional programming languages existed. Miranda was the most widely used, but it was proprietary software. At the conference on Functional Programming Languages and Computer Architecture (FPCA '87) in Portland, Oregon, there was a strong consensus that a committee be formed to define an open standard for such languages. The committee's purpose was to consolidate existing functional languages into a common one to serve as a basis for future research in functional-language design.[34]

Haskell 1.0 to 1.4 [edit]

Type classes, which enable type-safe operator overloading, were first proposed by Philip Wadler and Stephen Blott for Standard ML but were first implemented in Haskell between 1987 and version 1.0.[35][36]

The first version of Haskell ("Haskell 1.0") was defined in 1990.[1] The committee's efforts resulted in a series of language definitions (1.0, 1.1, 1.2, 1.3, 1.4).



Hierarchy of type classes in the Haskell prelude as of GHC 7.10. The inclusion of Foldable and Traversable (with corresponding changes to the type signatures of some functions), and of Applicative as intermediate between Functor and Monad, are deviations from the Haskell 2010 standard.

Haskell 98 [edit]

In late 1997, the series culminated in *Haskell 98*, intended to specify a stable, minimal, portable version of the language and an accompanying standard library for teaching, and as a base for future extensions. The committee expressly welcomed creating extensions and variants of Haskell 98 via adding and incorporating experimental features.[34]

In February 1999, the Haskell 98 language standard was originally published as *The Haskell 98 Report*.[34] In January 2003, a revised version was published as *Haskell 98 Language and Libraries: The Revised Report*.[27] The language continues to evolve rapidly, with the Glasgow Haskell Compiler (GHC) implementation representing the current *de facto* standard.[37]

Haskell 2010 [edit]

In early 2006, the process of defining a successor to the Haskell 98 standard, informally named *Haskell Prime*, began.[38] This was intended to be an ongoing incremental process to revise the language definition, producing a new revision up to once per year. The first revision, named **Haskell 2010**, was announced in November 2009[2] and published in July 2010.

Haskell 2010 is an incremental update to the language, mostly incorporating several well-used and uncontroversial features previously enabled via compiler-specific flags.

- Hierarchical module names. Module names are allowed to consist of dot-separated sequences of capitalized identifiers, rather than only one such identifier. This lets modules be named in a hierarchical manner (e.g., `Data.List` instead of `List`), although technically modules are still in a single monolithic namespace. This extension was specified in an addendum to Haskell 98 and was in practice universally used.
- The foreign function interface (FFI) allows bindings to other programming languages. Only bindings to C are specified in the Report, but the design allows for other language bindings. To support this, data type declarations were permitted to contain no constructors, enabling robust nonce types for foreign data that could not be constructed in Haskell. This extension was also previously specified in an Addendum to the Haskell 98 Report and widely used.
- So-called *n*+*k* patterns (definitions of the form `fact (n+1) = (n+1) * fact n`) were no longer allowed. This syntactic sugar had misleading semantics, in which the code looked like it used the `(+)` operator, but in fact desugared to code using `(-)` and `(>=)`.
- The rules of type inference were relaxed to allow more programs to type check.
- Some syntax issues (changes in the formal grammar) were fixed: pattern guards were added, allowing pattern matching within guards; resolution of operator fixity was specified in a simpler way that reflected actual practice; an edge case in the interaction of the language's lexical syntax of operators and comments was addressed, and the interaction of do-notation and if-then-else was tweaked to eliminate unexpected syntax errors.
- The `LANGUAGE` pragma was specified. By 2010, dozens of extensions to the language were in wide use, and GHC (among other compilers) provided the `LANGUAGE` pragma to specify individual extensions with a list of identifiers. Haskell 2010 compilers are required to support the `Haskell2010` extension and are encouraged to support several others, which correspond to extensions added in Haskell 2010.

Future standards [edit]

The next formal specification had been planned for 2020.[3] On 29 October 2021, with GHC version 9.2.1, the GHC2021 extension was released. While this is not a formal language spec, it combines a number of stable, widely-used GHC extensions to Haskell 2010. [39]

Features [edit]

Main article: Haskell features
See also: Glasgow Haskell Compiler § Extensions to Haskell

Haskell features lazy evaluation, lambda expressions, pattern matching, list comprehension, type classes and type polymorphism. It is a purely functional language, which means that functions generally have no side effects. A distinct construct exists to represent side effects, orthogonal to the type of functions. A pure function can return a side effect that is subsequently executed, modeling the impure functions of other languages.

Haskell has a strong, static type system based on Hindley–Milner type inference. Its principal innovation in this area is type classes, originally conceived as a principled way to add overloading to the language,[40] but since finding many more uses.[41]

The construct that represents side-effects is an example of a monad: a general framework which can model various computations such as error handling, nondeterminism, parsing and software transactional memory. They are defined as ordinary datatypes, but Haskell provides some syntactic sugar for their use.

Haskell has an open, published specification,[27] and multiple implementations exist. Its main implementation, the Glasgow Haskell Compiler (GHC), is both an interpreter and native-code compiler that runs on most platforms. GHC is noted for its rich type system incorporating recent innovations such as generalized algebraic data types and type families. The Computer Language Benchmarks Game also highlights its high-performance implementation of concurrency and parallelism.[42]

An active, growing community exists around the language, and more than 5,400 third-party open-source libraries and tools are available in the online package repository *Hackage*.[43]

Code examples [edit]

See also: Haskell features § Examples

A "Hello, World!" program in Haskell (only the last line is strictly necessary):

```haskell
module Main (main) where          -- not needed in interpreter, is the default in a module file

main :: IO ()                     -- the compiler can infer this type definition
main = putStrLn "Hello, World!"
```

The factorial function in Haskell, defined in a few different ways:

```haskell
-- [[Type signature|Type annotation]] (optional, same for each implementation)
factorial :: (Integral a) => a -> a

-- Using recursion (with the "ifthenelse" expression)
factorial n = if n < 2
              then 1
              else n * factorial (n - 1)

-- Using recursion (with pattern matching)
factorial 0 = 1
factorial n = n * factorial (n - 1)

-- Using recursion (with guards)
factorial n
   | n < 2     = 1
   | otherwise = n * factorial (n - 1)

-- Using a list and the "product" function
factorial n = product [1..n]

-- Using fold (implements "product")
factorial n = foldl (*) 1 [1..n]

-- Point-free style
factorial = foldr (*) 1 . enumFromTo 1
```

As the *Integer* type has arbitrary-precision, this code will compute values such as `factorial 100000` (a 456,574-digit number), with no loss of precision.

An implementation of an algorithm similar to quick sort over lists, where the first element is taken as the pivot:

```haskell
-- Type annotation (optional, same for each implementation)
quickSort :: Ord a => [a] -> [a]

-- Using list comprehensions
quickSort []     = []                               -- The empty list is already sorted
quickSort (x:xs) = quickSort [a | a <- xs, a < x]   -- Sort the left part of the list
                   ++ [x] ++                        -- Insert pivot between two sorted parts
                   quickSort [a | a <- xs, a >= x]  -- Sort the right part of the list

-- Using filter
quickSort []     = []
quickSort (x:xs) = quickSort (filter (<x) xs)
                   ++ [x] ++
                   quickSort (filter (>=x) xs)
```

Implementations [edit]

All listed implementations are distributed under open source licenses.[44]

Implementations that fully or nearly comply with the Haskell 98 standard, include:

- The Glasgow Haskell Compiler (GHC) compiles to native code on many different processor architectures, and to ANSI C, via one of two intermediate languages: C--, or in more recent versions, LLVM (formerly Low Level Virtual Machine) bitcode.[45][46] GHC has become the *de facto* standard Haskell dialect.[47] There are libraries (e.g., bindings to OpenGL) that work only with GHC. GHC was also distributed with the Haskell platform.
- Jhc, a Haskell compiler written by John Meacham, emphasizes speed and efficiency of generated programs and exploring new program transformations.
  - Ajhc is a fork of Jhc.
- The Utrecht Haskell Compiler (UHC) is a Haskell implementation from Utrecht University.[48] It supports almost all Haskell 98 features plus many experimental extensions. It is implemented using attribute grammars and is currently used mostly for research on generated type systems and language extensions.

Implementations no longer actively maintained include:

- The Haskell User's Gofer System (Hugs) is a bytecode interpreter. It was once one of the implementations used most widely, alongside the GHC compiler,[49] but has now been mostly replaced by GHCi. It also comes with a graphics library.
- HBC is an early implementation supporting Haskell 1.4. It was implemented by Lennart Augustsson in, and based on, Lazy ML. It has not been actively developed for some time.
- nhc98 is a bytecode compiler focusing on minimizing memory use.
  - The York Haskell Compiler (Yhc) was a fork of nhc98, with the goals of being simpler, more portable and efficient, and integrating support for Hat, the Haskell tracer. It also had a JavaScript backend, allowing users to run Haskell programs in web browsers.

Implementations not fully Haskell 98 compliant, and using a variant Haskell language, include:

- Eta and Frege are dialects of Haskell targeting the Java Virtual Machine.
- Gofer was an educational dialect of Haskell, with a feature called *constructor classes*, developed by Mark Jones. It was supplanted by Hugs (Haskell User's Gofer System).
- Helium, a newer dialect of Haskell. The focus is on making learning easier via clearer error messages. It currently lacks full support for type classes, rendering it incompatible with many Haskell programs.

Notable applications [edit]

- The proof assistant Agda is written in Haskell.[50]
- Cabal is a tool for building and packaging Haskell libraries and programs.[51]
- Darcs is a revision control system written in Haskell, with several innovative features, such as more precise control of patches to apply.
- GHC is also often a testbed for advanced functional programming features and optimizations in other programming languages.
- Git-annex is a tool to manage (big) data files under Git version control. It also provides a distributed file synchronization system (git-annex assistant).
- Linspire Linux chose Haskell for system tools development.[52]
- Pandoc is a tool to convert one markup format into another.
- Pugs is a compiler and interpreter for the Raku programming language (formerly Perl 6).
- TidalCycles is a domain special language for live coding musical pattern, embedded in Haskell.[53]
- Xmonad is a window manager for the X Window System, written fully in Haskell.[54]

Industry [edit]

- Bluespec SystemVerilog (BSV) is a language for semiconductor design that is an extension of Haskell. Also, Bluespec, Inc.'s tools are implemented in Haskell.
- Cryptol, a language and toolchain for developing and verifying cryptography algorithms, is implemented in Haskell.
- Facebook implements its anti-spam programs[55] in Haskell, maintaining the underlying data access library as open-source software.[56]
- The Cardano blockchain platform is implemented in Haskell.[57]
- GitHub implemented Semantic, an open-source library for analysis, diffing, and interpretation of untrusted source code, in Haskell.[58]
- Standard Chartered's financial modelling language Mu is syntactic Haskell running on a strict runtime. [59]
- seL4, the first formally verified microkernel,[60] used Haskell as a prototyping language for the OS developer.[60]:p.2 At the same time, the Haskell code defined an executable specification with which to reason, for automatic translation by the theorem-proving tool.[60]:p.3 The Haskell code thus served as an intermediate prototype before final C refinement.[60]:p.3
- Target stores' supply chain optimization software is written in Haskell. [61]

Web [edit]

Notable web frameworks written for Haskell include:[62]

- Snap
- Yesod

Criticism [edit]

Jan-Willem Maessen, in 2002, and Simon Peyton Jones, in 2003, discussed problems associated with lazy evaluation while also acknowledging the theoretical motives for it.[63][64] In addition to purely practical considerations such as improved performance,[65] they note that, in addition to adding some performance overhead, lazy evaluation makes it more difficult for programmers to reason about the performance of their code (particularly its space use).

Bastiaan Heeren, Daan Leijen, and Arjan van IJzendoorn in 2003 also observed some stumbling blocks for Haskell learners: "The subtle syntax and sophisticated type system of Haskell are a double edged sword – highly appreciated by experienced programmers but also a source of frustration among beginners, since the generality of Haskell often leads to cryptic error messages."[66] To address these, researchers from Utrecht University developed an advanced interpreter called Helium, which improved the user-friendliness of error messages by limiting the generality of some Haskell features, and in particular removing support for type classes.

Ben Lippmeier designed Disciple[67] as a strict-by-default (lazy by explicit annotation) dialect of Haskell with a type-and-effect system, to address Haskell's difficulties in reasoning about lazy evaluation and in using traditional data structures such as mutable arrays.[68] He argues (p. 20) that "destructive update furnishes the programmer with two important and powerful tools ... a set of efficient array-like data structures for managing collections of objects, and ... the ability to broadcast a new value to all parts of a program with minimal burden on the programmer."

Robert Harper, one of the authors of Standard ML, has given his reasons for not using Haskell to teach introductory programming. Among these are the difficulty of reasoning about resource use with non-strict evaluation, that lazy evaluation complicates the definition of datatypes and inductive reasoning,[69] and the "inferiority" of Haskell's (old) class system compared to ML's module system.[70]

Haskell's build tool, Cabal, has historically been criticized for poorly handling multiple versions of the same library, a problem known as "Cabal hell". The Stackage server and Stack build tool were made in response to these criticisms.[71] Cabal itself now has a much more sophisticated build system, heavily inspired by Nix,[72] which became the default with version 3.0.

Related languages [edit]

Clean is a close, slightly older relative of Haskell. Its biggest deviation from Haskell is in the use of uniqueness types instead of monads for I/O and side-effects.

A series of languages inspired by Haskell, but with different type systems, have been developed, including:

- Agda, a functional language with dependent types.
- Cayenne, with dependent types.
- Elm, a functional language to create web front-end apps, no support for user-defined or higher-kinded type classes or instances.
- Epigram, a functional language with dependent types suitable for proving properties of programs.
- Idris, a general purpose functional language with dependent types, developed at the University of St Andrews.
- PureScript compiles to JavaScript.
- Ωmega, strict and more.[clarification needed]

Other related languages include:

- Curry, a functional/logic programming language based on Haskell.

Notable Haskell variants include:

- Generic Haskell, a version of Haskell with type system support for generic programming.
- Hume, a strict functional language for embedded systems based on processes as stateless automata over a sort of tuples of one element mailbox channels where the state is kept by feedback into the mailboxes, and a mapping description from outputs to channels as box wiring, with a Haskell-like expression language and syntax.

Conferences and workshops [edit]

The Haskell community meets regularly for research and development activities. The main events are:

- International Conference on Functional Programming (ICFP)
- Haskell Symposium (formerly the Haskell Workshop)
- Haskell Implementors Workshop
- Commercial Users of Functional Programming (CUFP)

Starting in 2006, a series of organized *hackathons* has occurred, the Hac series, aimed at improving the programming language tools and libraries.[73]

References [edit]

1. ^ a b c Hudak et al. 2007.
2. ^ a b Marlow, Simon (24 November 2009). "Announcing Haskell 2010". *Haskell* (Mailing list). Retrieved 12 March 2011.
3. ^ a b Riedel, Herbert (28 April 2016). "ANN: Haskell Prime 2020 committee has formed". *Haskell-prime* (Mailing list). Retrieved 6 May 2017.
4. ^ a b c d e f g h i j k l m Peyton Jones 2003, p. xi
5. ^ Norell, Ulf (2008). "Dependently Typed Programming in Agda" (PDF). Gothenburg: Chalmers University. Retrieved 9 February 2012.
6. ^ Hudak et al. 2007, pp. 12–38, 43.
7. ^ Stroustrup, Bjarne; Sutton, Andrew (2011). "Design of Concept Libraries for C++" (PDF). Archived from the original (PDF) on 10 February 2012.
8. ^ a b c d e f g h i j Hudak et al. 2007, pp. 12-45–46.
9. ^ a b Meijer, Erik (2006). "Confessions of a Used Programming Language Salesman: Getting the Masses Hooked on Haskell". *Oopsla 2007*. CiteSeerX 10.1.1.72.868.
10. ^ Meijer, Erik (1 October 2009). "C9 Lectures: Dr. Erik Meijer – Functional Programming Fundamentals, Chapter 1 of 13". *Channel 9*. Microsoft. Retrieved 9 February 2012.
11. ^ Drobi, Sadek (4 March 2009). "Erik Meijer on LINQ". *InfoQ*. QCon SF 2008: C4Media Inc. Retrieved 9 February 2012.
12. ^ Hickey, Rich. "Clojure Bookshelf". *Listmania!*. Archived from the original on 3 October 2017. Retrieved 3 October 2017.
13. ^ Heller, Martin (18 October 2011). "Turn up your nose at Dart and smell the CoffeeScript". *InfoWorld*. Retrieved 2020-07-15.
14. ^ "Declarative programming in Escher" (PDF). Retrieved 7 October 2015.
15. ^ Syme, Don; Granicz, Adam; Cisternino, Antonio (2007). *Expert F#*. Apress. p. 2. "F# also draws from Haskell particularly with regard to two advanced language features called *sequence expressions* and *workflows*."
16. ^ "Facebook Introduces 'Hack,' the Programming Language of the Future". *WIRED*. 20 March 2014.
17. ^ "Idris, a dependently typed language". Retrieved 26 October 2014.
18. ^ "LiveScript Inspiration". Retrieved 4 February 2014.
19. ^ Freeman, Phil (2016). "PureScript by Example". Leanpub. Retrieved 23 April 2017.
20. ^ Kuchling, A. M. "Functional Programming HOWTO". *Python v2.7.2 documentation*. Python Software Foundation. Retrieved 9 February 2012.
21. ^ "Glossary of Terms and Jargon". *Perl Foundation Perl 6 Wiki*. The Perl Foundation. Archived from the original on 21 January 2012. Retrieved 9 February 2012.
22. ^ "The Rust Reference: Appendix: Influences". Retrieved 3 February 2016.
23. ^ Fogus, Michael (6 August 2010). "MartinOdersky take(5) toList". *Send More Paramedics*. Retrieved 9 February 2012.
24. ^ Lattner, Chris (3 June 2014). "Chris Lattner's Homepage". Chris Lattner. Retrieved 3 June 2014. "The Swift language is the product of tireless effort from a team of language experts, documentation gurus, compiler optimization ninjas, and an incredibly important internal dogfooding group who provided feedback to help refine and battle-test ideas. Of course, it also greatly benefited from the experiences hard-won by many other languages in the field, drawing ideas from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list."
25. ^ Chevalier, Tim (28 January 2008). "anybody can tell me the pronunciation of "haskell"?". *Haskell-cafe* (Mailing list). Retrieved 12 March 2011.
26. ^ Type inference originally using Hindley-Milner type inference
27. ^ a b c Peyton Jones 2003.
28. ^ Edward Kmett, Edward Kmett – Type Classes vs. the World
29. ^ Mossberg, Erik (8 June 2020), *erkmos/haskell-companies*, retrieved 22 June 2020
30. ^ O'Sullivan, Bryan; Goerzen, John; Stewart, Donald Bruce (15 November 2008). *Real World Haskell: Code You Can Believe In*. "O'Reilly Media, Inc.". pp. xxviii–xxxi. ISBN 978-0-596-55430-9.
31. ^ "Haskell in Production: Riskbook". *Serokell Software Development Company*. Retrieved 7 September 2021.
32. ^ "PYPL PopularitY of Programming Language index". *pypl.github.io*. May 2021. Archived from the original on 7 May 2021. Retrieved 16 May 2021.
33. ^ Frederickson, Ben. "Ranking Programming Languages by GitHub Users". *www.benfrederickson.com*. Retrieved 6 September 2019.
34. ^ a b c Peyton Jones 2003, Preface.
35. ^ "Type classes, first proposed during the design of the Haskell programming language, ..." —John Garrett Morris (2013), "Type Classes and Instance Chains: A Relational Approach"
36. ^ Wadler, Philip (October 1988). "How to make ad-hoc polymorphism less ad hoc".
37. ^ "Haskell Wiki: Implementations". Retrieved 18 December 2012.
38. ^ "Welcome to Haskell'". *The Haskell' Wiki*. Archived from the original on 20 February 2016. Retrieved 11 February 2016.
39. ^ GHC 2020 Team (29 October 2021) GHC 9.2.1 released
40. ^ Wadler, P.; Blott, S. (1989). "How to make ad-hoc polymorphism less ad hoc". *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM: 60–76. doi:10.1145/75277.75283. ISBN 978-0-89791-294-5. S2CID 15327197.
41. ^ Hallgren, T. (January 2001). "Fun with Functional Dependencies, or Types as Values in Static Computations in Haskell". *Proceedings of the Joint CS/CE Winter Meeting*. Varberg, Sweden.
42. ^ Computer Language Benchmarks Game
43. ^ "HackageDB statistics". Hackage.haskell.org. Archived from the original on 3 May 2013. Retrieved 26 June 2013.
44. ^ "Implementations" at the Haskell Wiki
45. ^ "The LLVM Backend". *GHC Trac*.
46. ^ Terei, David A.; Chakravarty, Manuel M. T. (2010). "An LLVM Backend for GHC". *Proceedings of ACM SIGPLAN Haskell Symposium 2010*. ACM Press.
47. ^ C. Ryder and S. Thompson (2005). "Porting HaRe to the GHC API"
48. ^ Utrecht Haskell Compiler
49. ^ Hudak et al. 2007, pp. 12–22.
50. ^ *Agda 2*, Agda Github Community, 15 October 2021, retrieved 16 October 2021
51. ^ "The Haskell Cabal". Retrieved 8 April 2015.
52. ^ "Linspire/Freespire Core OS Team and Haskell". *Debian Haskell mailing list*. May 2006.
53. ^ "Live code with Tidal Cycles I Tidal Cycles". *doc.tidalcycles.org*. Retrieved 19 January 2022.
54. ^ xmonad.org
55. ^ "Fighting spam with Haskell". *Facebook Code*. 26 June 2015. Retrieved 11 August 2019.
56. ^ "Open-sourcing Haxl, a library for Haskell". *Facebook Code*. 10 June 2014. Retrieved 11 August 2019.
57. ^ "input-output-hk/cardano-node: The core component that is used to participate in a Cardano decentralised blockchain". GitHub. Retrieved 18 March 2022.
58. ^ *Parsing, analyzing, and comparing source code across many languages: github/semantic*, GitHub, 7 June 2019, retrieved 7 June 2019
59. ^ "Commercial Users of Functional Programming Workshop Report" (PDF). Retrieved 10 June 2022.
60. ^ a b c d A formal proof of functional correctness was completed in 2009. Klein, Gerwin; Elphinstone, Kevin; Heiser, Gernot; Andronick, June; Cock, David; Derrin, Philip; Elkaduwe, Dhammika; Engelhardt, Kai; Kolanski, Rafal; Norrish, Michael; Sewell, Thomas; Tuch, Harvey; Winwood, Simon (October 2009). "seL4: Formal verification of an OS kernel" (PDF). *22nd ACM Symposium on Operating System Principles*. Big Sky, MT, USA.
61. ^ Yula Gavrilova and Gints Dreimanis. Top 6 Software Projects Written in Haskell. Serokell Blog. October 17, 2019. <https://serokell.io/blog/top-software-written-in-haskell>. Accessed November 4, 2021
62. ^ "Web/Frameworks – HaskellWiki". *wiki.haskell.org*. Retrieved 11 August 2019.
63. ^ Jan-Willem Maessen. *Eager Haskell: Resource-bounded execution yields efficient iteration*. Proceedings of the 2002 Association for Computing Machinery (ACM) SIGPLAN workshop on Haskell.
64. ^ [dead link]Simon Peyton Jones. *Wearing the hair shirt: a retrospective on Haskell*. Invited talk at POPL 2003.
65. ^ "Lazy evaluation can lead to excellent performance, such as in The Computer Language Benchmarks Game".
66. ^ Heeren, Bastiaan; Leijen, Daan; van IJzendoorn, Arjan (2003). "Helium, for learning Haskell" (PDF). *Proceedings of the 2003 ACM SIGPLAN Workshop on Haskell*: 62–71. doi:10.1145/871895.871902. ISBN 1581137583. S2CID 11986908.
67. ^ "DDC – HaskellWiki". Haskell.org. 3 December 2010. Retrieved 26 June 2013.
68. ^ Ben Lippmeier, Type Inference and Optimisation for an Impure World, Australian National University (2010) PhD thesis, chapter 1
69. ^ Robert Harper. "The point of laziness".
70. ^ Robert Harper. "Modules matter most".
71. ^ "Solving Cabal Hell". *www.yesodweb.com*. Retrieved 11 August 2019.
72. ^ "Announcing cabal new-build: Nix-style local builds". Retrieved 1 October 2019.
73. ^ "Hackathon – HaskellWiki".

Further reading [edit]

Reports

- Peyton Jones, Simon, ed. (2003). *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press. ISBN 978-0521826143.
- Marlow, Simon, ed. (2010). *Haskell 2010 Language Report* (PDF). Haskell.org.

Textbooks

- Davie, Antony (1992). *An Introduction to Functional Programming Systems Using Haskell*. Cambridge University Press. ISBN 978-0-521-25830-2.
- Bird, Richard (1998). *Introduction to Functional Programming using Haskell* (2nd ed.). Prentice Hall Press. ISBN 978-0-13-484346-9.
- Hudak, Paul (2000). *The Haskell School of Expression: Learning Functional Programming through Multimedia*. New York: Cambridge University Press. ISBN 978-0521643382.
- Hutton, Graham (2007). *Programming in Haskell*. Cambridge University Press. ISBN 978-0521692694.
- O'Sullivan, Bryan; Stewart, Don; Goerzen, John (2008). *Real World Haskell*. Sebastopol: O'Reilly. ISBN 978-0-596-51498-3 (full text) {{cite book}}: External link in |postscript= (help)
- Thompson, Simon (2011). *Haskell: The Craft of Functional Programming* (3rd ed.). Addison-Wesley. ISBN 978-0201882957.
- Lipovača, Miran (April 2011). *Learn You a Haskell for Great Good!*. San Francisco: No Starch Press. ISBN 978-1-59327-283-8. (full text)
- Bird, Richard (2014). *Thinking Functionally with Haskell*. Cambridge University Press. ISBN 978-1-107-45264-0.

Tutorials

- Hudak, Paul; Peterson, John; Fasel, Joseph (June 2000). "A Gentle Introduction To Haskell, Version 98". *Haskell.org*.
- Learn You a Haskell for Great Good! - A community version (*learnyouahaskell.github.io*). An up-to-date community maintained version of the renowned "Learn You a Haskell" (LYAH) guide.
- Yet Another Haskell Tutorial, by Hal Daumé III; assumes far less prior knowledge than official tutorial
- Yorgey, Brent (12 March 2009). "The Typeclassopedia" (PDF). *The Monad.Reader* (13): 17–68.
- Maguire, Sandy (2018). *Thinking with Types: Type-Level Programming in Haskell*.

History

- Hudak, Paul; Hughes, John; Peyton Jones, Simon; Wadler, Philip (2007). "A History of Haskell: Being Lazy with Class" (PDF). *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages (HOPL III)*: 12–1–55. doi:10.1145/1238844.1238856. ISBN 978-1-59593-766-7. S2CID 52847907.
- Hamilton, Naomi (19 September 2008). "The A-Z of Programming Languages: Haskell". *Computerworld*.

External links [edit]

- Official website
- Official wiki

Wikibooks has a book on the topic of: *Haskell*

Wikibooks has a book on the topic of: *Write Yourself a Scheme in 48 Hours*