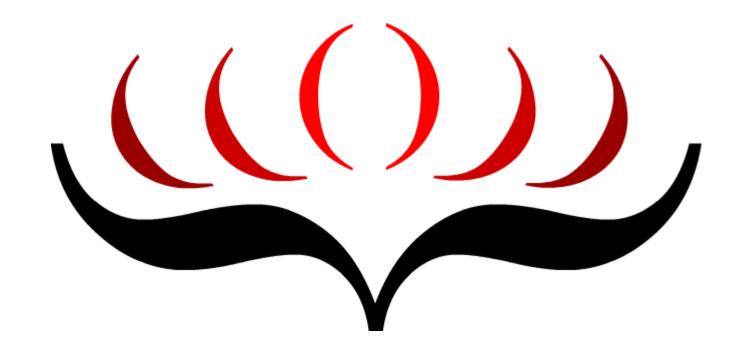
The philosophy and history of Haskell



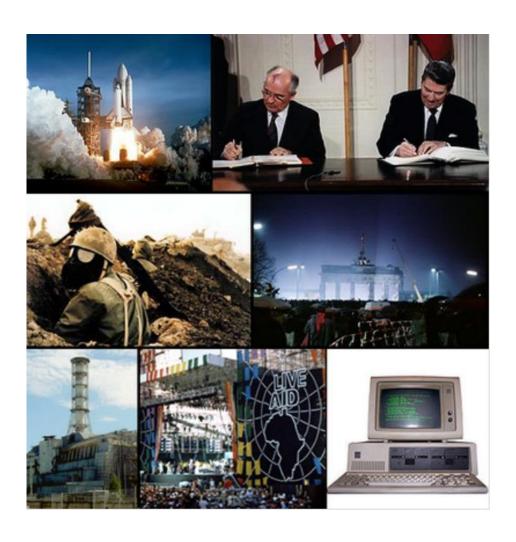
Agenda

- History of Haskell
- Language philosophy
- Goals and principles
- Haskell in industry
- Haskell community

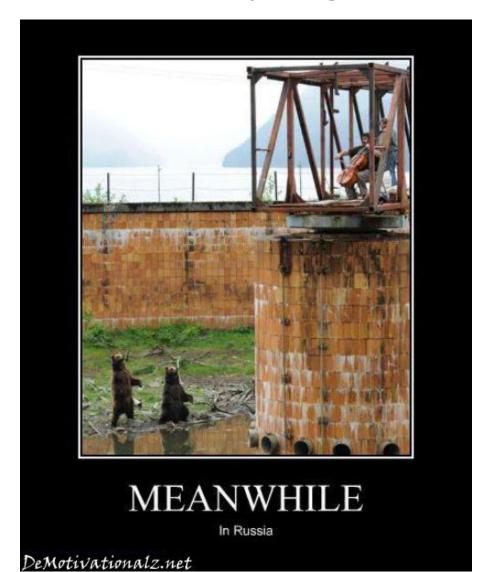




1980s



Meanwhile, in ... functional programming

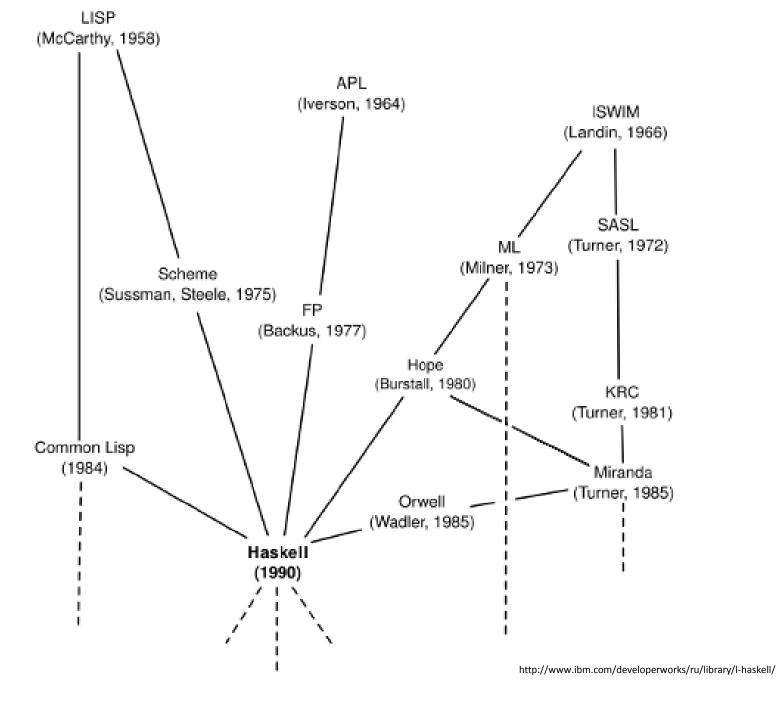


Each group develops its own language



Significant impact on Haskell

- ML (1973) the first language with typed Hindley-Milner.
- SASL, KRC, Miranda (1972–1985) one of the first lazy languages.
- Hope (1980) one of the first languages with algebraic data types.



The birth of Haskell

The FPCA meeting (1987) thus marked the beginning of the Haskell design process.

Although we had no name for the language and very few technical discussions or design decisions occurred.

The Yale meeting

- The first physical meeting (after the impromptu FPCA meeting) was held at Yale, January 9–12, 1988,
- The first order of business was to establish the following goals for the language.

Goals for the language

The Yale meeting

- 1. It should be suitable for teaching, research, and applications, including building large systems.
- 2. It should be completely described via the publication of a formal syntax and semantics.
- 3. It should be freely available.
- 4. It should be usable as a basis for further language research.
- 5. It should be based on ideas that enjoy a wide consensus.
- 6. It should reduce unnecessary diversity in functional programming languages.

Choosing a name

Semla, Haskell, Vivaldi, Mozart, CFL (Common Functional Language), Funl 88, Semlor, Candle (Common Applicative Notation for Denoting Lambda Expressions), Fun, David, Nice, Light, ML Nouveau (or Miranda Nouveau, or LML Nouveau, or ..), Mirabelle, Concord, LL, Slim, Meet, Leval, Curry, Frege, Peano, Ease, Portland, Haskell B Curry.

Haskell B. Curry



The Glasgow meeting

• That meeting was held April 6–9, 1988 at the University of Glasgow, whose Functional programming group was beginning a period of rapid growth.

IFIP WG2.8 meetings



Paul Hudak



Philip Wadler



Simon Peyton Jones



A short history (1)

1 April 1990
 The Haskell version 1.0

August 1991
 The Haskell version 1.1

March 1992
 The Haskell version 1.2

A short history (2)

1994.

Haskell gained Internet presence when John Peterson registered the haskell.org domain name and set up a server and web-site at Yale.

- May 1996.
 The Haskell version 1.3
- April 1997.
 The Haskell version 1.4

A short history (3)

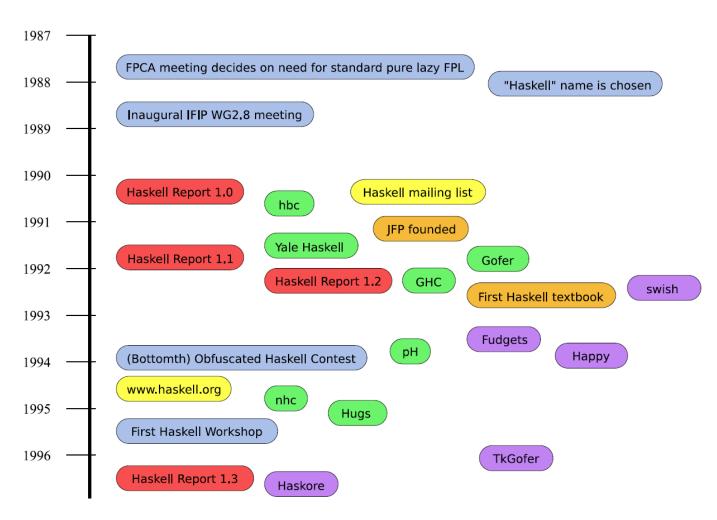
February 1999

The Haskell 98

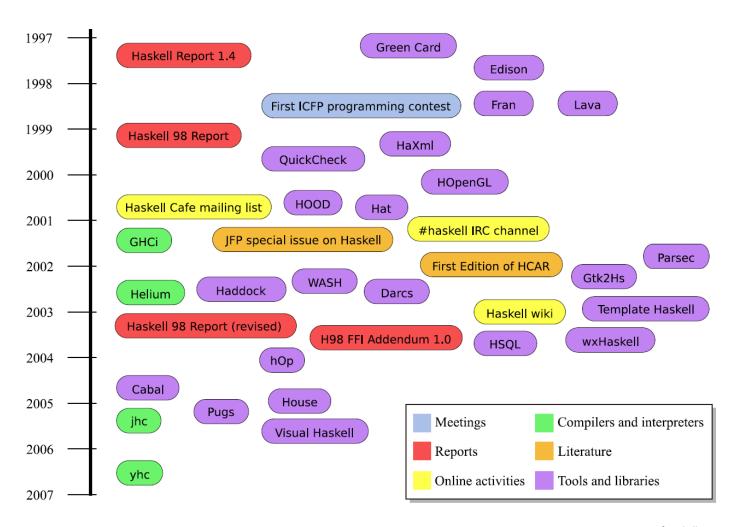
• July 2010

The Haskell 2010

Haskell timeline (1)



Haskell timeline (2)



Haskell is a committee language

Haskell is a language designed by committee, and conventional wisdom would say that a committee language will be full of warts and awkward compromises.

Properties of Haskell

- Lazy evaluation
- Non-side effect
- GC and automatic allocation of data included
- Strong typing

Lazy evaluation Properties of Haskell

Functions won't perform sequentially, instead they will evaluate expressions when needed.

Non-side effect Properties of Haskell

There is a controlled environment for side effect functions but by default functions are pure.

GC Properties of Haskell

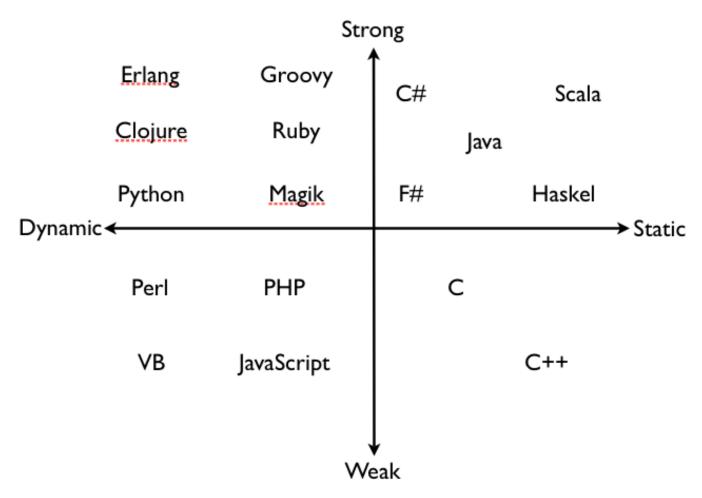
Management of resources is abstracted completely from the developer.

Haskell's type system

There are three interesting aspects to types in Haskell:

- they are strong,
- they are static,
- and they can be automatically inferred.

Strong vs weak dynamic vs static



Strong types

- The type system guarantees that a program cannot contain certain kinds of errors.
- These errors come from trying to write expressions that don't make sense, such as using an integer as a function.

Strong types

Another aspect of Haskell's view of strong typing is that it will not automatically coerce values from one type to another.

Coercion is also known as casting or conversion.

Implicitly convert in C#

```
int intValue = 1;
uint uintValue = 1;
long longValue = 1;
ulong ulongValue = 1;
```

C# is strongly a typed language, but it has a different meaning than in Haskell.

```
float floatValueFromInt = intValue;
float floatValueFromUint = uintValue;
float floatValueFromLong = ulongValue;
float floatValueFromUlong = longValue;
double doubleValueFromLong = longValue;
double doubleValueFromUlong = ulongValue;
```

Static types

The compiler knows the type of every value and expression at compile time, before any code is executed.

Static types

```
Prelude> True && 0
<interactive>:15:9:
    No instance for (Num Bool) arising from the literal `0'
    Possible fix: add an instance declaration for (Num Bool)
    In the second argument of (\&\&)', namely 0'
    In the expression: True && 0
    In an equation for `it': it = True && 0
Prelude> True && "False"
<interactive>:16:9:
    Couldn't match expected type `Bool' with actual type `[Char]'
    In the second argument of `(&&)', namely `"False"'
    In the expression: True && "False"
    In an equation for `it': it = True && "False"
```

Static types

 Static typing can occasionally make it difficult to write some useful kinds of code.

 Fortunately, Haskell's system of typeclasses, provides almost all of the benefits of dynamic typing, in a safe and convenient form.

What are typeclasses?

 Typeclasses define a set of functions that can have different implementations depending on the type of data they are given.

 Typeclasses may look like the objects of object-oriented programming, but they are truly quite different.

A Haskell program that compiles will not suffer from type errors when it runs.

Haskell's combination of strong and static typing makes it impossible for type errors to occur at runtime.

Jigsaw puzzle



Type inference

A Haskell compiler can automatically deduce the types of almost all expressions in a program.

Implementations

Haskell is a language with many implementations, two of which are widely used:

- GHC
- Hugs

The Glasgow haskell compiler

 Probably the most fully featured Haskell compiler today is the Glasgow Haskell Compiler (GHC), an open-source project with a liberal BSD-style licence.

GHC has three main components

- ghc an optimizing compiler that generates fast native code
- ghci an interactive interpreter and debugger
- runghc a program for running Haskell programs as scripts, without needing to compile them first

Haskell User's Gofer System
Hugs

It's a bytecode interpreter for the functional programming language Haskell.

GHC vs Hugs

Compared to Hugs, GHC is more suited to "real work": it compiles to native code, supports parallel execution, and provides useful performance analysis and debugging tools.

Good For Equational Reasoning Gofer

It's an implementation of the programming language Haskell intended for educational purposes and supporting a language based on version 1.2 of the Haskell report.

Yale Haskell

An early implementation of Haskell in Lisp

The Yale Haskell compiler is slower than the Glasgow and Chalmers Haskell compilers.

Other Haskell compilers

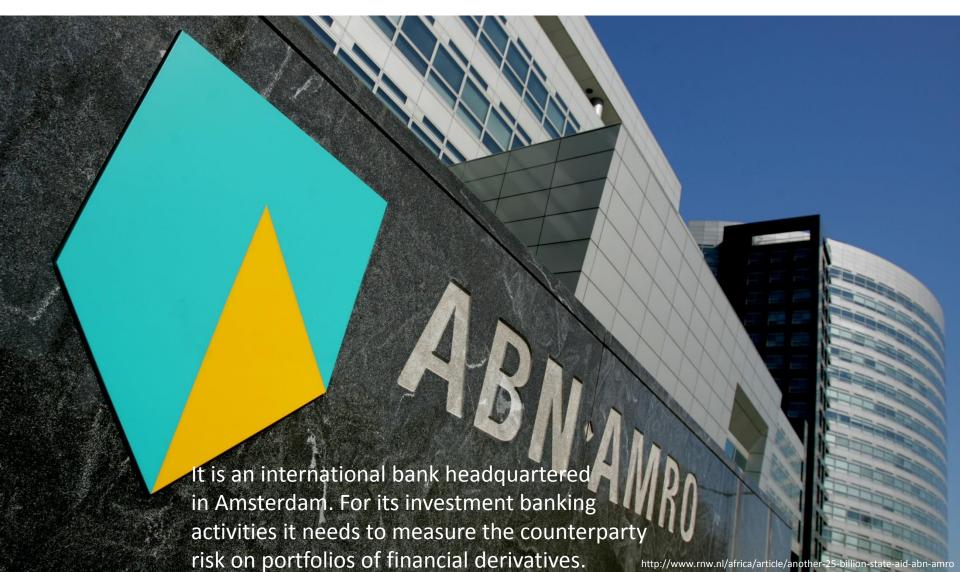
- Helium
- UHC and EHC
- Jhc
- The York Haskell Compiler

Haskell in industry



ABN AMRO

Amsterdam, The Netherlands





A consortium of groups, including Alcatel-Lucent, have used Haskell

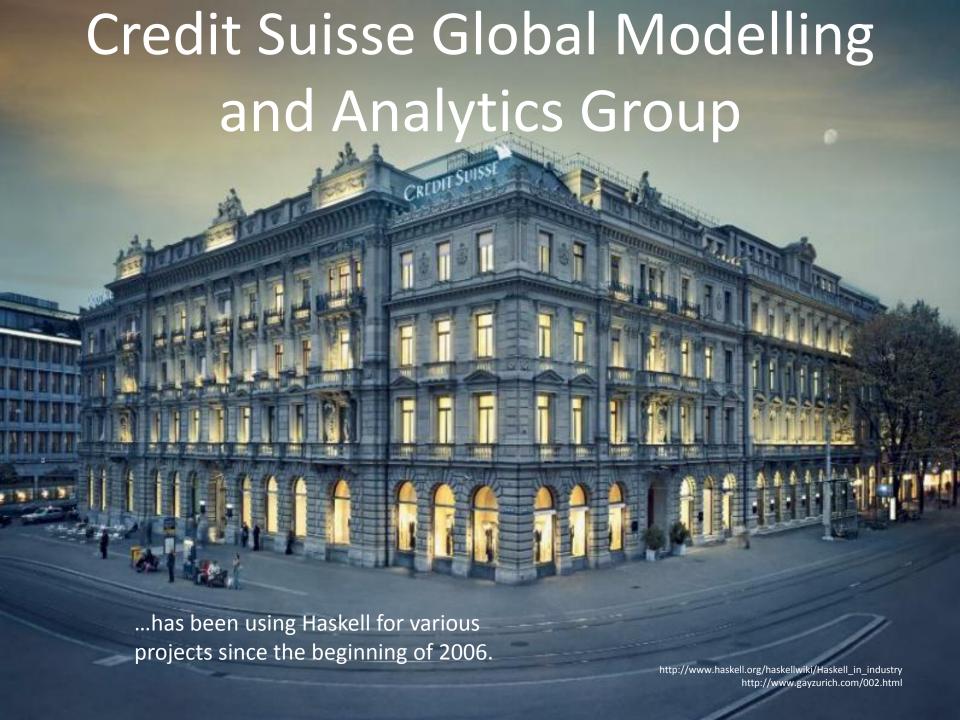
to prototype narrowband software radio systems, running in (soft) real-time.

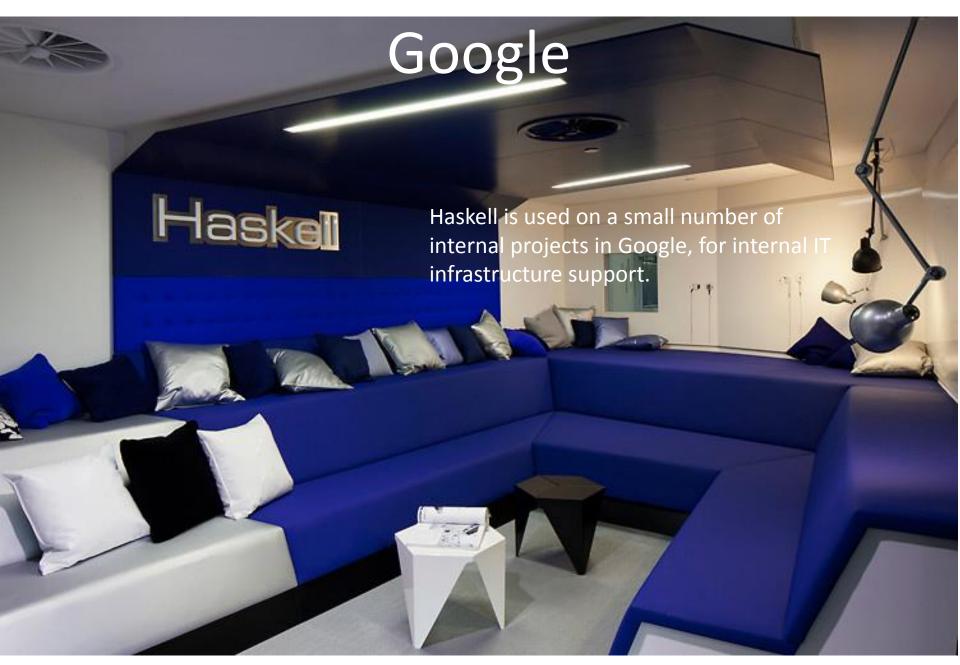
ALLSTON

TRADING

It is a premier high frequency market maker in over 40 financial exchanges, in 20 countries, and in nearly every conceivable product class.



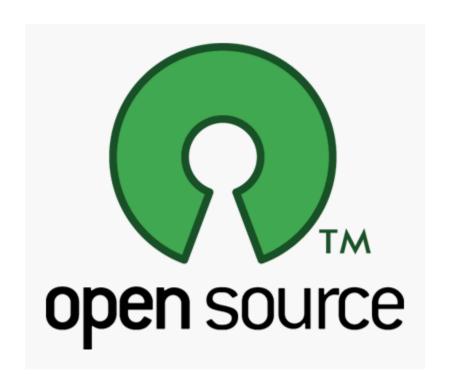




Facebook uses some Haskell internally for tools

http://www.haskell.org/haskellwiki/Haskell_in_industry

Haskell in open source



A databases

- Haskelldb
- HDBC
- Takusen
- Library to work with BerkeleyDB, CouchDB, MongoDB, Redis, TokyoCabinet, TokyoTyrant, SimpleDB, SQLite

A graphics

- Blobs
- Diagrams
- FieldTrip
- Glome
- GLUT and OpenGL
- GPipe
- Grapefruit
- Haven
- HaskellCharts
- SDL
- Yampa

• ...

Graphical user interface

- Gtk2Hs
- Grapefruit and wxFruit
- qtHaskell
- wxHaskell

Games

- bloxorz
- Frag
- monadius
- Raincat

Internet

- gitit
- happstack
- Twidge

Word processing

- HaXml
- HXT
- Leksah
- Pandoc
- Parsec
- The Grammatical Framework
- Yi

Parallel, multi-tasking and multithreaded programming

- CHP Communicating Haskell Processes
- Data Parallel Haskell
- parallel
- STM

Development

- alex
- cabal-install
- happy
- haddock
- HUnit
- QuickCheck

System software

- Darcs
- House
- Xmonad
- Himerge
- FreeArc

Languages and compilers

- Agda
- Curry
- cpphs
- Epigram
- Flapjax
- Language.Python
- Language.C
- Lava
- IIvm
- Pugs
- WebBits

Compilation, debugging, and performance analysis

- For practical work, almost as important as a language itself is the ecosystem of libraries and tools around it.
- Haskell has a strong showing in this area.

Compilation, debugging, and performance analysis

- Compiles to efficient native code on all major modern operating systems and CPU architectures.
- Easy deployment of compiled binaries, unencumbered by licensing restrictions.
- Code coverage analysis.
- Detailed profiling of performance and memory usage.

The Haskell community



The Haskell community

- 1. http://www.haskell.org/
- 2. http://haskell.org/haskellwiki/Mailing lists
- http://haskell.org/haskellwiki/IRC_channel named #haskell
- 4. <u>http://haskell.org/haskellwiki/User_groups</u>
- 5. http://www.meetup.com/find/
- 6. http://sequence.complete.org/
- 7. http://www.haskell.org/haskellwiki/Haskell Communities and Activities Report

Previous Haskell.org logo



The great 2009 Haskell logo contest





Haskell
BIND THE REAL WORLD

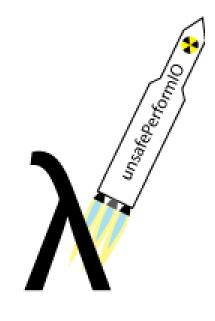




http://www.haskell.org/haskellwiki/Haskell_logos/New_logo_ideas

The great 2009 Haskell logo contest















Current Haskell logo



The current Thompson-Wheeler logo was chosen in the logo competition in early 2009. It replaces the previous Haskell.org logo.

When learning Haskell is a good idea?



When learning Haskell is a good idea?

- If you are a experienced developer that wants to grow professionally by learning completely new and different approaches to solve problems
- If you need to build systems where correctness is critical (Banks, Safety-Critical Systems)

When learning Haskell is a good idea?

- Is well suited to implement Domain Specific Languages, so it could work very well in systems programming
- as Erlang, it is easy to implement parallel and concurrent programs safely due to the pureness of the language

Benefits of using Haskell



Benefits of using Haskell

- Algorithms are normally shorter and more concise than their iterative versions
- Due to the use of function composability, curryfication and high-order functions, code can be reused in really smart ways
- The kick-ass type system with type inference makes you develop code that would normally work at the first compilation

Why Haskell is not being used as much?



Why Haskell is not being used as much?

- It's different, people is always afraid of what is different
- It's difficult, mostly related to the first point
- It's unpopular, mostly related to the first and second point

Why Haskell is not being used as much?

- It's risky (for employers), there are not many Haskell developers to count on, of course this is mostly related to the first, second and third point
- Libraries are broad, but there is no depth (Many incompatible experimental libraries that do the same thing)

Resources

Books and papers

- A History of Haskell: Being LazyWith Class by Paul Hudak, John Hughes, Philip Wadler, Simon Peyton Jones
- Real World Haskell by Bryan O'Sullivan, Don Stewart, and John Goerzen
- Programming in Haskell by Graham Hutton
- Haskell The Craft of Functional Proaramming by Simon Thompson