Haskell
Functional Programming

http://igm.univ-mlv.fr/~vialette/?section=teaching

Stéphane Vialette

LIGM, Université Paris-Est Marne-la-Vallée

November 11, 2016

Everybody's talking about functional programming

Recently, functional programming (FP) has getting a lot of attention.

- F# is being developed at Microsoft Developer Division and is being distributed as a fully supported language in the .NET framework.
- Clojure, which runs on the JVM, was introduced in 2007.
- Haskell, Erlang and Sclala have recently gained much popularity.
- Elixir is the new kid on the block.

Everybody's talking about functional programming

There are 4 primary reasons for FP's newly established popularity:

1. FP offers concurrency/parallelism with tears.
2. FP has succint, concise and understandable syntax.
3. FP offers a different programming perspective.
4. FP is becoming more accessible.

FP is fun!

FP offers concurrency/parallelism with tears

Moore's law has held up for years but it is starting to reach its limits due to physical constraints. Chips aren't getting much faster but multi-core, hyper-threaded, etc machines are becoming far more commonplace.

If you want to take advantages of your machine's full processing power, you can no longer rely on continuous chip advances alone. You need to really start thinking about concurrency, parallelism and multi-threaded if you wish to better performance and use all available CPUs.

Of course, these are not easily implemented concepts so coders need to start considering ways (like FP!) to make these approaches more available and practical.

FP has succint, concise and understandable syntax

The abstract nature of FP leads to considerably simpler programs. It also supports a number of powerful new ways to structure and reason about programs.

`x = x+1;` We understand this syntax because we often resort to telling the computer what to do, but this equation really makes no sense at all!

Ask, don't tell.

FP offers a different programming perspective

For me, the most important thing about FP isn't that functional languages have some particular useful language features, but that it allows to think differently and simply about problems that you encouter when designing and writing applications. This is much more important than understanding any new technology or a programming language.

Tomas Petricek
`http://tomasp.net/blog/`

FP is becoming more accessible

More language options.

Tooling, IDEs.

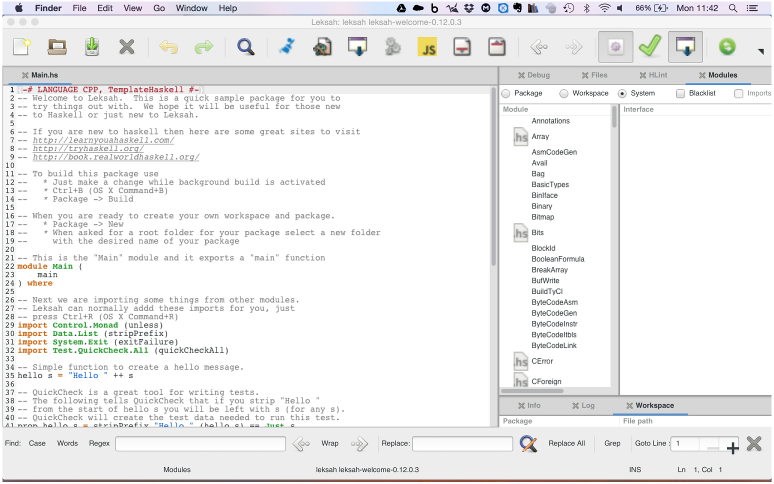
Supports.

Books.

Blogs, podcasts and screencasts.

Conferences and user groups.

FP is becoming more accessible



```haskell
{-# LANGUAGE CPP, TemplateHaskell #-}
-- Welcome to Leksah.  This is a quick sample package for you to
-- try things out with.  We hope it will be useful for those new
-- to Haskell or just new to Leksah.

-- If you are new to haskell then here are some great sites to visit
-- http://learnyouahaskell.com/
-- http://tryhaskell.org/
-- http://book.realworldhaskell.org/

-- To build this package use
--    * Just make a change while background build is activated
--    * Ctrl+B (OS X Command+B)
--    * Package -> Build

-- When you are ready to create your own workspace and package.
--    * Package -> New
--    * When asked for a root folder for your package select a new folder
--      with the desired name of your package

-- This is the "Main" module and it exports a "main" function
module Main (
    main
) where

-- Next we are importing some things from other modules.
-- Leksah can normally addd these imports for you, just
-- press Ctrl+R (OS X Command+R)
import Control.Monad (unless)
import Data.List (stripPrefix)
import System.Exit (exitFailure)
import Test.QuickCheck.All (quickCheckAll)

-- Simple function to create a hello message.
hello s = "Hello " ++ s

-- QuickCheck is a great tool for writing tests.
-- The following tells QuickCheck that if you strip "Hello "
-- from the start of hello s you will be left with s (for any s).
-- QuickCheck will create the test data needed to run this test.
prop_hello s = stripPrefix "Hello " (hello s) == Just s
```

Key FP concepts

High order functions, map, filter reduce (*i.e.*, fold).

Recursion.

Pattern matching.

Currying.

Lazy/eager evaluation.

Strict/non-strict semantics.

Type inference.

Monads.

Continuations.

Closures.

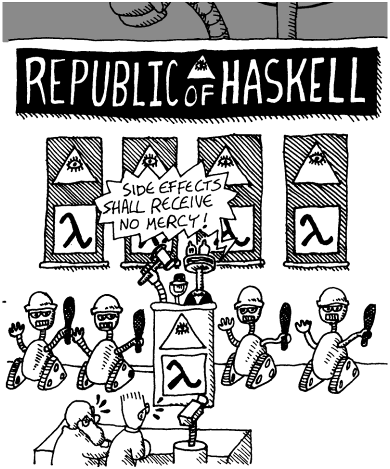
Haskell

Haskell

Haskell is a standardized, general-purpose purely functional programming language, with non-strict semantics and strong static typing.

It is named after logician Haskell Curry.

Haskell



What can Haskell offer the programmer?

Purity: Unlike some other functional programming languages Haskell is pure. It doesn't allow any side-effects. This is probably the most important feature of Haskell.

Laziness: Haskell is lazy (technically speaking, it's "non-strict"). This means that nothing is evaluated until it has to be evaluated.

Strong typing: Haskell is strongly typed, this means just what it sounds like. It's impossible to inadvertently convert a `Double` to an `Int`, or follow a null pointer. Unlike other strongly typed languages types in Haskell are automatically inferred.

Elegance: Another property of Haskell that is very important to the programmer, even though it doesn't mean as much in terms of stability or performance, is the elegance of Haskell. To put it simply: stuff just works like you'd expect it to.

Haskell and bugs

Pure. There are no side effects.

Strongly typed. There can be no dubious use of types. And No Core Dumps!

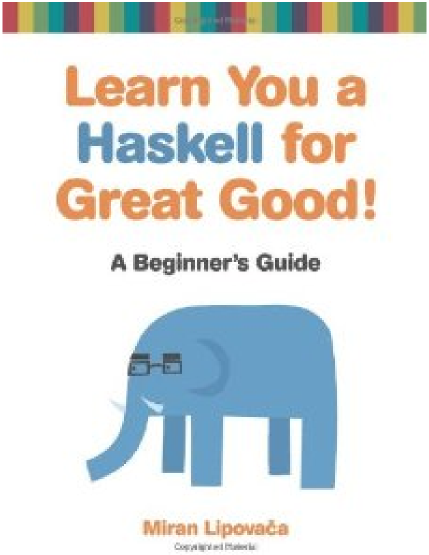
Concise. Programs are shorter which make it easier to look at a function and "take it all in" at once, convincing yourself that it's correct.

High level. Haskell programs most often reads out almost exactly like the algorithm description. Which makes it easier to verify that the function does what the algorithm states.

Memory managed. There's no worrying about dangling pointers, the Garbage Collector takes care of all that.

Modular. Haskell offers stronger and more "glue" to compose your program from already developed modules.

Reference book



Hello, World!

```haskell
module Main where

main :: IO ()
main = putStrLn "Hello, World!"
```

Hello, World!: Compile to native code

```
barbalala: ghc -o Hello Hello.hs
[1 of 1] Compiling Main             ( Hello.hs, Hello.o )
Linking Hello ...
barbalala: ./Hello
Hello, World!
barbalala:
```

Hello, World!: Interpreter

```
barbalala: ghci
GHCi, version 7.8.3: http://www.haskell.org/ghc/
:? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude> :load "Hello"
[1 of 1] Compiling Main ( Hello.hs, interpreted )
Ok, modules loaded: Main.
*Main> main
Hello, World!
*Main>
```

Quicksort in Haskell

```haskell
quicksort :: Ord a => [a] -> [a]
quicksort []     = []
quicksort (p:xs) = quicksort lesser ++ -- Sort the left part of the list
                   [p]              ++ -- Insert pivot
                   quicksort greater   -- Sort the right part of the list
    where
        lesser  = filter (< p)  xs
        greater = filter (>= p) xs
```

Implementations

The Glasgow Haskell Compiler (GHC) compiles to native code on a number of different architectures. GHC has become the de facto standard Haskell dialect. There are libraries (e.g. bindings to OpenGL) that will work only with GHC. GHC is also distributed along with the Haskell platform.

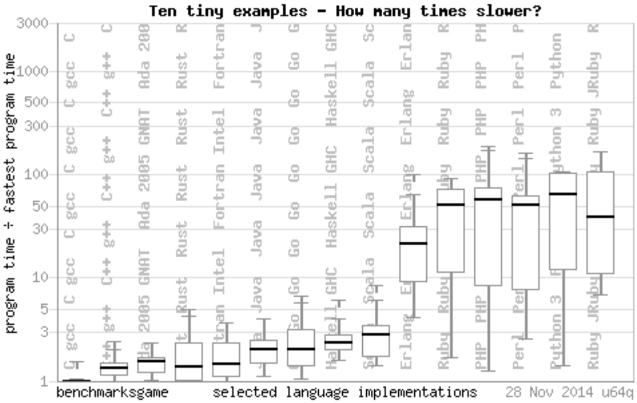
The Utrecht Haskell Compiler (UHC) is a Haskell implementation from Utrecht University. UHC supports almost all Haskell 98 features plus many experimental extensions.

Jhc is a Haskell compiler written by John Meacham emphasising speed and efficiency of generated programs as well as exploration of new program transformations.

Ajhc is a fork of Jhc.

The speed of Haskell

For most applications the difference in speed between C++ and Haskell is so small that it's utterly irrelevant



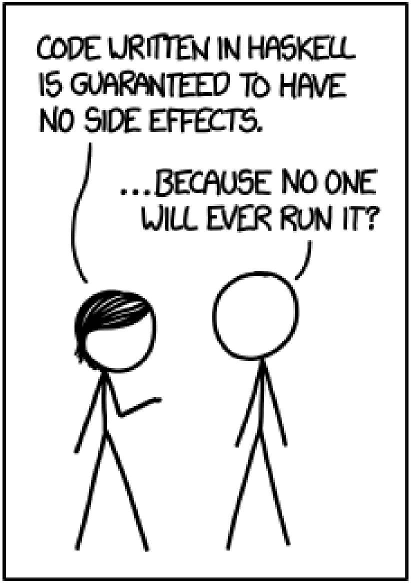
The speed of Haskell

There's an old rule in computer programming called the "*80/20 rule*". It states that 80% of the time is spent in 20% of the code. The consequence of this is that any given function in your system will likely be of minimal importance when it comes to optimizations for speed. There may be only a handful of functions important enough to optimize.

Remember that algorithmic optimization can give much better results than code optimization.

Last but not least, Haskell offers substantially increased programmer productivity (Ericsson measured an improvement factor of between 9 and 25 using Erlang, a functional programming language similar to Haskell, in one set of experiments on telephony software.)

Haskell in Industry


CODE WRITTEN IN HASKELL IS GUARANTEED TO HAVE NO SIDE EFFECTS.
...BECAUSE NO ONE WILL EVER RUN IT?

Haskell in Industry

ABN AMRO Amsterdam, The Netherlands

ABN AMRO is an international bank headquartered in Amsterdam. For its investment banking activities it needs to measure the counterparty risk on portfolios of financial derivatives.

Aetion Technologies LLC, Columbus, Ohio, USA

Aetion was a defense contractor in operation from 1999 to 2011, whose applications use artificial intelligence.

Alcatel-Lucent

A consortium of groups, including Alcatel-Lucent, have used Haskell to prototype narrowband software radio systems, running in (soft) real-time.

Haskell in Industry

Soostone, New York, NY, USA

Soostone is an advanced analytics technology provider specializing in algorithmic optimization opportunities in marketing, pricing, advertising, sales and product management.

NRAO

NRAO has used Haskell to implement the core science algorithms for the Robert C. Byrd Green Bank Telescope (GBT) Dynamic Scheduling System (DSS).

IMVU, Inc

IMVU, Inc. is a social entertainment company connecting users through 3D avatar-based experiences.

Functor AB, Stockholm, Sweden

Functor AB offers new tools for ground-breaking static analysis with pre-test case generation of programs to eliminate defects and

Functional programming languages

Functional programming languages

Clojure

General-purpose programming language with an emphasis on functional programming.

It runs on the Java Virtual Machine, Common Language Runtime, and JavaScript engines. Like other Lisps, Clojure treats code as data and has a macro system.

Designed by Rich Hickey (2007).

Compojure, Ring, Incanter, Datmomic, . . .

Functional programming languages
Scala

Object-functional programming language for general software applications. Scala has full support for functional programming and a very strong static type system.

Scala source code is intended to be compiled to Java bytecode, so that the resulting executable code runs on a Java virtual machine. Java libraries may be used directly in Scala code, and vice versa.

Designed by Martin Odersky (EPFL).

Spark, Akka, . . .

Functional programming languages

Erlang

General-purpose concurrent, garbage-collected programming language and runtime system

The Ericsson Erlang implementation loads virtual machine bytecode which is converted to threaded code at load time.

Designed by Joe Armstrong, Robert Virding and Mike Williams at Ericsson (1986) to support distributed, fault-tolerant, soft-real-time, non-stop applications.Released as open source in 1998.

CouchDB, Couchbase Server, Riak, Chef, Wings 3D, RabbitMQ, WhatsApp, Call of Duty server core, Goldman Sachs (high-frequency trading programs), . . .

Functional programming languages
Elixir

Elixir is a dynamic, functional language designed for building scalable and maintainable applications.

Elixir leverages the Erlang VM, known for running low-latency, distributed and fault-tolerant systems, while also being successfully used in web development and the embedded software domain.

Phoenix, Ecto, . . .

Functional programming languages
Ocaml

OCaml unifies functional, imperative, and object-oriented programming under an ML-like type system.

OCaml's toolset includes an interactive top level interpreter, a bytecode compiler, and an optimizing native code compiler.

INRIA (1996).

Hack (Facebook), Unison, Frama-C, Coq, Mirage, . . .