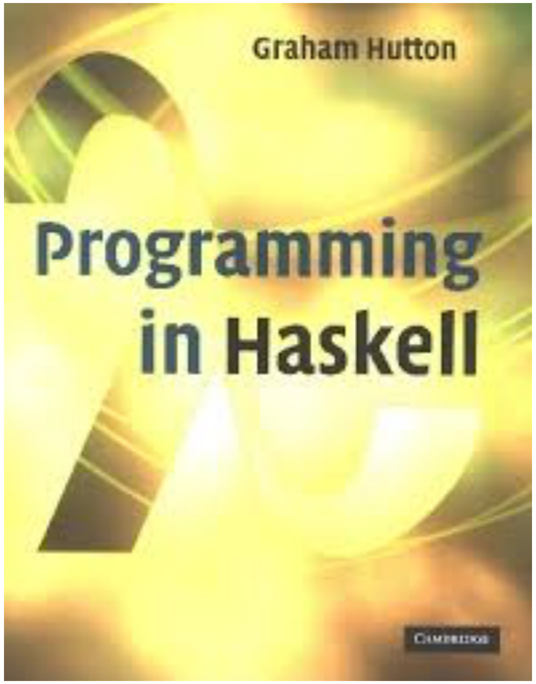
PROGRAMMING IN HASKELL

Chapter 1 and 2 - Introduction and First Steps
(Original Slides by Graham Hutton)

Programming in Haskell

- This course follows the Programming in Haskell book, by Prof. Graham Hutton



Learn you a Haskell for Great Good

- Learn you a Haskell for Great Good is a fun and easy to read alternative

http://learnyouahaskell.com

… there are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.

Tony Hoare, 1980 ACM Turing Award Lecture

The Software Crisis

- How can we cope with the size and complexity of modern computer programs?

- How can we reduce the time and cost of program development?

- How can we increase our confidence that the finished programs work correctly?

Programming Languages

One approach to the software crisis is to design new programming languages that:

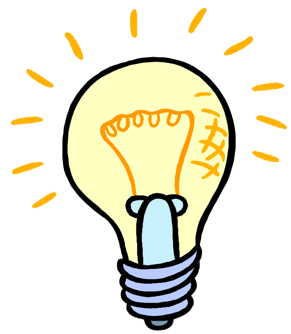
- Allow programs to be written clearly, concisely, and at a high-level of abstraction;

- Support reusable software components;

- Encourage the use of formal verification;

- Permit rapid prototyping;

- Provide powerful problem-solving tools.



Functional languages provide a particularly <u>elegant</u> framework in which to address these goals.

What is a Functional Language?

Opinions differ, and it is difficult to give a precise definition, but generally speaking:

- Functional programming is <u>style</u> of programming in which the basic method of computation is the application of functions to arguments;

- A functional language is one that <u>supports</u> and <u>encourages</u> the functional style.

Computing in Imperative Programming

Summing the integers 1 to 10 in Java:

```
total = 0;
for (i = 1; i ≤ 10; ++i)
   total = total+i;
```

The computation method is <u>variable assignment</u>.

Computing in Functional Programming

Summing the integers 1 to 10 in Haskell:

```
sum [1..10]
```

The computation method is <u>function application</u>.

Example

Double function

$$double\ x = x + x$$

How to compute the result of double 3?

Example

double 3

{by definition of double}

3 + 3

{arithmetic}

6

Example

How about?

double (double 2)

Example

double (double 2)
= {by definition of double}
double (2 + 2)
= {arithmetic}
double 4
= {by definition of double}
4 + 4
= {arithmetic}
8

Example

Summing a list of integers:

```
sum []     = 0
sum (x:xs) = x + sum xs
```

Example

Calculate the result of:

```
sum [1,2,3]
```

Example

```
sum []     = 0
sum (x:xs) = x + sum xs
```

sum [1,2,3]
= {by definition of sum}
1 + sum [2,3]
= {by definition of sum}
1 + 2 + sum [3]
= {by definition of sum}
1 + 2 + 3 + sum []
= {by definition of sum
1 + 2 + 3 + 0
= {arithmetic}
6

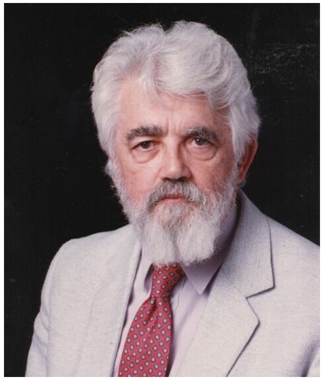
Historical Background

1930s:

Alonzo Church develops the lambda calculus, a simple but powerful theory of functions.

Historical Background

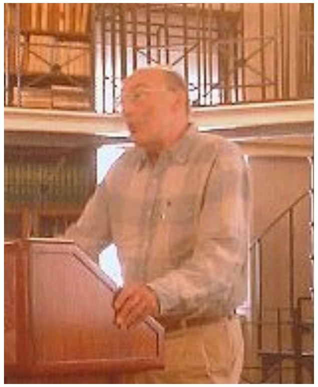
1950s:



John McCarthy develops Lisp, the first functional language, with some influences from the lambda calculus, but retaining variable assignments.

Historical Background

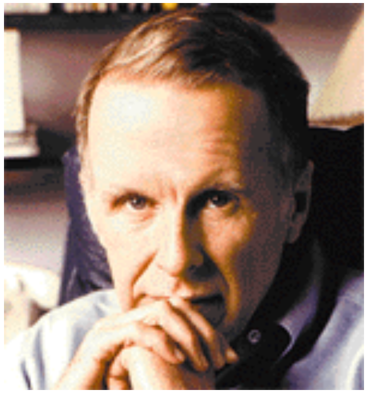
1960s:



Peter Landin develops ISWIM, the first pure functional language, based strongly on the lambda calculus, with no assignments.

Historical Background

1970s:



John Backus develops FP, a functional language that emphasizes higher-order functions and reasoning about programs.

Historical Background

1970s:

Robin Milner and others develop ML, the first modern functional language, which introduced type inference and polymorphic types.

Historical Background

1970s - 1980s:

David Turner develops a number of lazy functional languages, culminating in the Miranda system.

Historical Background

1987:

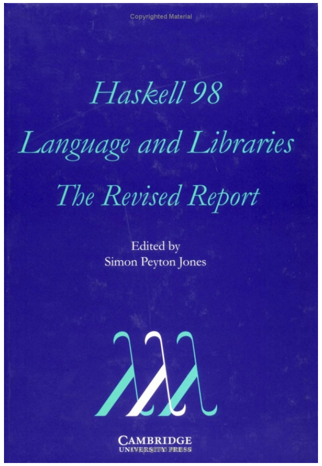
Haskell

A Purely Functional Language

An international committee of researchers initiates the development of Haskell, a standard lazy functional language.

Historical Background

2003:



The committee publishes the Haskell 98 report, defining a stable version of the language.

Historical Background

2003-date:

Standard distribution, library support, new language features, development tools, use in industry, influence on other languages, etc.

A Taste of Haskell

```
f []     = []
f (x:xs) = f ys ++ [x] ++ f zs
           where
             ys = [a | a ← xs, a ≤ x]
             zs = [b | b ← xs, b > x]
```

?

PROGRAMMING IN HASKELL

Chapter 2 - First Steps

Glasgow Haskell Compiler

- GHC is the leading implementation of Haskell, and comprises a compiler and interpreter;

- The interactive nature of the interpreter makes it well suited for teaching and prototyping;

- GHC is freely available from:

  www.haskell.org/

Starting GHC

The GHC interpreter can be started from the Unix command prompt % by simply typing ghci:

```
% ghci
GHCi, version 7.4.1: http://www.haskell.org/
ghc/  :? for help

Prelude>
```

The GHCi prompt > means that the interpreter is ready to evaluate an expression.

For example:

```
> 2+3*4
14

> (2+3)*4
20

> sqrt (3^2 + 4^2)
5.0
```

The Standard Prelude

Haskell comes with a large number of standard library functions. In addition to the familiar numeric functions such as + and *, the library also provides many useful functions on lists.

- Select the first element of a list:

```
> head [1,2,3,4,5]
1
```

- Remove the first element from a list:

```
> tail [1,2,3,4,5]
[2,3,4,5]
```

- Select the nth element of a list:

```
> [1,2,3,4,5] !! 2
3
```

- Select the first n elements of a list:

```
> take 3 [1,2,3,4,5]
[1,2,3]
```

- Remove the first n elements from a list:

```
> drop 3 [1,2,3,4,5]
[4,5]
```

- Calculate the length of a list:

```
> length [1,2,3,4,5]
5
```

- Calculate the sum of a list of numbers:

```
> sum [1,2,3,4,5]
15
```

- Calculate the product of a list of numbers:

```
> product [1,2,3,4,5]
120
```

- Append two lists:

```
> [1,2,3] ++ [4,5]
[1,2,3,4,5]
```

- Reverse a list:

```
> reverse [1,2,3,4,5]
[5,4,3,2,1]
```

Function Application

In mathematics, function application is denoted using parentheses, and multiplication is often denoted using juxtaposition or space.

$$f(a,b) + c\ d$$

Apply the function f to a and b, and add the result to the product of c and d.

In Haskell, function application is denoted using space, and multiplication is denoted using *.

```
f a b + c*d
```

As previously, but in Haskell syntax.

Moreover, function application is assumed to have higher priority than all other operators.

f a + b

Means (f a) + b, rather than f (a + b).

Examples

Mathematics	Haskell
f(x)	f x
f(x,y)	f x y
f(g(x))	f (g x)
f(x,g(y))	f x (g y)
f(x)g(y)	f x * g y

Haskell Scripts

- As well as the functions in the standard library, you can also define your own functions;

- New functions are defined within a <u>script</u>, a text file comprising a sequence of definitions;

- By convention, Haskell scripts usually have a <u>.hs</u> suffix on their filename. This is not mandatory, but is useful for identification purposes.

My First Script

When developing a Haskell script, it is useful to keep two windows open, one running an editor for the script, and the other running GHCi.

Start an editor, type in the following two function definitions, and save the script as Test.hs:

```
double x    = x + x

quadruple x = double (double x)
```

Leaving the editor open, in another window start up GHCi with the new script:

```
% ghci Test.hs
```

Now both the standard library and the file test.hs are loaded, and functions from both can be used:

```
> quadruple 10
40

> take (double 2) [1,2,3,4,5,6]
[1,2,3,4]
```

Leaving GHCi open, return to the editor, add the following two definitions, and resave:

```
factorial n = product [1..n]

average ns  = sum ns `div` length ns
```

Note:

- div is enclosed in back quotes, not forward;
- x `f` y is just syntactic sugar for f x y.

GHCi does not automatically detect that the script has been changed, so a reload command must be executed before the new definitions can be used:

```
> :reload
Reading file "test.hs"

> factorial 10
3628800

> average [1,2,3,4,5]
3
```

Naming Requirements

- Function and argument names must begin with a lower-case letter. For example:

myFun   fun1   arg_2   x'

- By convention, list arguments usually have an s suffix on their name. For example:

xs   ns   nss

The Layout Rule

In a sequence of definitions, each definition must begin in precisely the same column:

```
a = 10

b = 20

c = 30
```

```
a = 10

 b = 20

c = 30
```

```
 a = 10

b = 20

 c = 30
```

The layout rule avoids the need for explicit syntax to indicate the grouping of definitions.

```
a = b + c
    where
      b = 1
      c = 2
d = a * 2
```

means

```
a = b + c
    where
      {b = 1;
       c = 2}
d = a * 2
```

implicit grouping

explicit grouping

Useful GHCi Commands

Command	Meaning
:load name	load script name
:reload	reload current script
:edit name	edit script name
:edit	edit current script
:type expr	show type of expr
:?	show all commands
:quit	quit GHCi

Exercises

(1) Try out slides 2-8 and 14-17 (Chapter 2) using GHCi.

(2) Fix the syntax errors in the program below, and test your solution using GHCi.

```
N = a ’div’ length xs
    where
       a = 10
      xs = [1,2,3,4,5]
```

(3) Show how the library function last that selects the last element of a list can be defined using the functions introduced in this lecture.

(4) Can you think of another possible definition?

(5) Similarly, show how the library function init that removes the last element from a list can be defined in two different ways.