プログラムの数理
Mathematical Structures in Programs

胡　振江

平成15年度冬学期

目的

- プログラミングを数学的（代数的）な活動としての考え方を伝授すること。
- 代数（Algebra）とは
  （The Shorter Oxford English Dictionary):
  – the reunion of broken parts
  – a calculus of symbols combined according to defined laws

内容

- 関数プログラミング言語Haskellの学習
  – プログラム：関数の定義
  – プログラムの実行：式の簡約
- 関数プログラミングの特徴の理解
  – 問題の記述：抽象的
  – プログラム構造：構成的
  – プログラム間の関係：推論，操作しやすい
  – プログラム性質：証明しやすい

他講義との関係

プログラムの数理（3年）

計算モデルの数理（4年）

教科書

- 武市正人訳、「関数プログラミング」，近代科学社，1994年. ISBN4-7649-0181-1
  (R. Bird and P. Wadler, Introduction to Functional Programming, Prentice Hall, 1988)

各自購入すること

参考書など

- Richard Bird. Introduction to Functional Programming in Haskell, Prentice Hall, 1998.
- 講義ページ：
  http://www.ipl.t.u-tokyo.ac.jp/~hu/pm03/

日程

- 10月：6，13（祝日），20，27
- 11月：3（祝日），10，17，24（祝日）
- 12月：1（休講），8，15，22（グループプロジェクト，発表）
- 1月：8，12（祝日），19，26
- 2月：2（予備）

欠席，遅刻しないよう

評価・成績

- 出席 10%
- レポート 20%
- 期末試験 70%

学習方法

- 講義で内容を理解しよう
- 練習問題をやろう
- プログラムを書こう

なぜ関数プログラミングを勉強するのか

注：資料の一部はJohn Hughesの講義資料を参考したものです．

Software

Software = Programs + Data

Data

Data is any kind of storable information. Examples:

- Numbers
- Letters
- Email messages
- Songs on a CD
- Maps
- Video clips
- Mouse clicks
- Programs

Programs

Programs compute new data from old data.

Example: *Baldur's Gate* computes a sequence of screen images and sounds from a sequence of mouse clicks.

Building Software Systems

A large system may contain many *millions* of lines of code.

Software systems are among the most complex artefacts ever made.

Systems are built by combining existing components as far as possible.

Volvo buys engines from Mitsubishi. ⟷ Bonnier buys Quicktime Video from Apple.

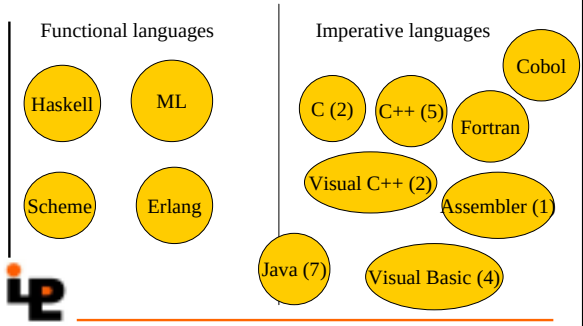
Programming Languages

Programs are written in *programming languages.*

There are hundreds of different programming languages, each with their strengths and weaknesses.

A large system will often contain components in many different languages.

Which Language Should We Teach?



Industrial Uses of Functional Languages

Intel (microprocessor verification)

Hewlett Packard (telecom event correlation)

Ericsson (telecommunications)

Carlstedt Research & Technology (air-crew scheduling)

Legasys (Y2K tool)

Hafnium (Y2K tool)

Shop.com (e-commerce)

Motorola (test generation)

Thompson (radar tracking)

Why Haskell?

•Haskell is a very *high-level language* (many details taken care of automatically).

•Haskell is expressive and concise (can achieve a lot with a little effort).

•Haskell is good at handling complex data and combining components.

•Haskell is not a high-performance language (prioritise programmer-time over computer-time).

Haskellの概観

Functional Programming

A function（**関数**）is a way of computing a result from the function arguments.

A function producing a number from an angle.

f(x) = sin x/cos x

game(mouse clicks) = screen animation

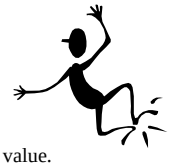
A function producing a sequence of images from a sequence of mouse clicks.

A functional program computes its output as a function of its input.

Values and Expressions

A *value* (**値**) is a piece of data.

2, 4, 3.14159, ”John”,



An *expression* (**式**) computes a value.

2 + 2, 2*pi*r

Expressions combine values using functions and operators.

Operations

Operators（**演算子**）are always explicit:

b^2 - 4*a*c

Power. Multiplication.

•Cannot be written as b2 - 4ac.

•Means (b^2) - (4*a*c), not e.g. b^((2-4)*a*c).

Multiplication (*) *binds more tightly* than subtraction (-).

Functions

The solution of a quadratic equation:

(-b + sqrt (b^2 - 4*a*c)) / (2*a)

A function.

Definitions and Types

A *definition* (**定義**) gives a name to a value.

Types（**型**）specify what kind of value this is.

area :: Int

area = 41*37

Names start with a small letter, and are made up of letters and digits.

An expression says how the value is computed.

Function Definitions

A *function definition* (**関数定義**) specifies how the result is computed from the arguments.

```
area :: Int -> Int -> Int
area l w = l*w
```

Function types specify the types of the arguments and the result.

The body specifies how the result is computed.

The arguments are given names, after the function name.

Cf. area(l,w) = l*w

Function Notation

*Function arguments need **not** be enclosed in brackets!*

Example: average :: Float -> Float -> Float
average x y = (x + y) / 2

Calls: average 2 3 ⟶ 2.5
average (2+2) (3*3) ⟶ 6.5

Brackets are for grouping only!

Functional Programming

A *functional program* consists mostly of function definitions.

Simple functions are used to define more complex ones, which are used to define still more complex ones, and so on.

Finally, we define a function to compute the output of the entire program from its inputs.

If you can write function definitions,
you can write functional programs!

A simple Haskell Program (Script)

プログラム名

Test.hs

定義の並び

```
square x = x * x
side = 12
area = square side
min' :: Int -> Int -> Int
min' x y | x<=y     = x
         | otherwise = y
```

型の宣言は必須ではない

Running Haskell Programs

Install Hugs in your PC
http://www.haskell.org/hugs/
(ECC has Hugs installed)

Primitive Library: Prelude.hs

Extended Library: Char.hs, List.hs, System.hs, …

Your Program: Test.hs, …

A Tour of Some Basic Types

From mathematics, we're used to functions whose arguments and results are numbers. In programs, we usually work with much richer types of values.

Some types are built in to programming languages (e.g. numbers), others are defined by programmers (e.g. MP3).

Let us tour some of Haskell's built-in types (in Prelude.hs).

Types: Integers

1, 2, 3, 4… :: Int

Whole numbers (between -2^31 and 2^31-1).

Some operations:

2+3 → 5

2*3 → 6

2^3 → 8

div 7 2 → 3

mod 7 2 → 1

OBS! integer division!

Types: Real Numbers

1.5, 0.425, 3.14159… :: Float

Real numbers (with about 6 significant figures).

Some operations:

2.5 + 1.5 → 4.0

3 - 1.2 → 1.8

1.4142^2 → 1.99996

1 / 3 → 0.333333

sin (pi/4) → 0.707107

Types: Lists

A list of values enclosed in square brackets.

[1,2,3], [2] :: [Int]

A list of integers.

Some operations:

[1,2,3] ++[4,5] → [1,2,3,4,5]

head [1,2,3] → 1

last [1,2,3] → 3

Quiz

How would you add 4 to the end of the list [1,2,3]?

Quiz

How would you add 4 to the end of the list [1,2,3]?

[1,2,3] ++ [4] → [1,2,3,4]

OBS! [4] not 4!
++ combines two *lists*, and 4 is not a list.

Types: Strings

Any characters enclosed in double quotes.

”Hello!” :: String

The type of a piece of text.

Some operations:

”Hello ” ++ ”World” → ”Hello World”

show (2+2) → ”4”

Quiz

Is ”2+2” equal to ”4”?

Quiz

Is ”2+2” equal to ”4”?

No!

”2+2” is a string three characters long.

”4” is a string one character long.

They are *not* the same text!

Types: Commands

A command to write ”Hello!” to myfile.

The type of a command which produces no value.

```
writeFile ”myfile” ”Hello!”   :: IO ()
readFile ”myfile”             :: IO String
```

The type of a command which produces a String.

Quiz

If myfile contains ”Hello!”,

is readFile ”myfile” equal to ”Hello!”?

Quiz

If myfile contains ”Hello!”,

is readFile ”myfile” equal to ”Hello!”?

NO!

This is a *command* to read a file.

This is a constant piece of text.

The result of a function depends only on its arguments; ”Hello!” cannot be computed from ”myfile”.

Effects of Commands

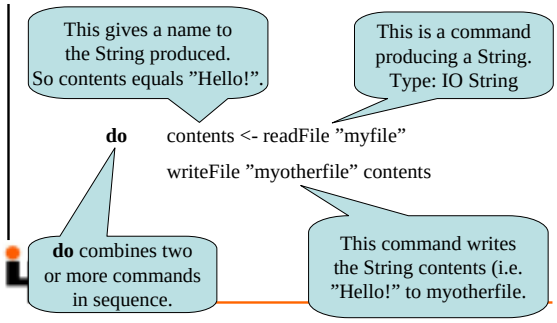
The result of a function depends only on its arguments.

The *effect* of a command may be different, depending on when it is executed.

”Take one step backwards”

is always the same command...

Combining Commands



More Examples

hello.hs

```
hello :: IO ()
hello = putStrLn “Hello World!”
```

Display “Hello World!”

greeting.hs

```
greeting :: IO ()
greeting = do putStr “Tell me your name: “
             name <- getLine
             putStrLn (“Hello “ ++ name ++ “ !”)
```

Get a line from the input.

Types: Functions

```
double :: Int -> Int
double x = x+x
```

double 2 ⟶ 4

is a function call.

double

(no arguments) is a *function value*.

Function Composition

```
quadruple :: Int -> Int
quadruple = double . double
```

Function composition: an operator on functions!

quadruple 2 ⟶ double (double 2)

⟶ double 4

⟶ 8

The map Function

```
doubles :: [Int] -> [Int]
doubles = map double
```

A function with a function as its argument and its result!

doubles [1,2,3] ⟶ [double 1, double 2, double 3]

⟶ [2, 4, 6]

”Higher-Order” Functions

The ability to compute functions (= programs) is one of Haskell’s greatest strengths.

Large parts of a program may be computed (”written by the computer”) rather than programmed by hand.

But this is an advanced topic to which we will return many times.

Putting it Together: A Friendly Email Sender

mail hu@mist.i.u-tokyo.ac.jp

email hu

Define a *command* to send mail, which looks up the right email address automatically.

Storing Email Addresses

Should we store email addresses:

in the program?

in a separate file?

Easy to modify. Many users can share the program.

File: addresses

Zhenjiang Hu	hu@mist.i.u-tokyo.ac.jp
Masato Takeichi	takeichi@mist.i.u-toklyo.ac.jp
Kazuhiko Kakehi	kaz@ipl.t.u-tokyo.ac.jp

What Components Can We Reuse?

- •grep to search for the email address.
- •emacs to edit the message.
- •mail to send the message.

Our Plan: To Send Email to John

Produces "Zhenjiang Hu hu@mist.i.u-tokyo.ac.jp" in file recipient.

- •grep Hu addresses > recipient.
- •readFile recipient: the address is the last "word".
- •emacs message.
- •mail address < message

Create the message file.

Send the contents of the message file to the address.

How Can We Run Another Program?

system "emacs message" :: IO ExitCode

A command which executes a String as a shell command.

The result produced is an *exit code*; ignore it.

How Can We Extract the Email Address?

Reuse a standard function:

words :: String -> [String]

words "Zhenjiang Hu hu@mist.i.u-tokyo.ac.jp"

→ ["Zhenjiang", "Hu", "hu@mist.i.u-tokyo.ac.jp"]

Putting it all Together

```
email :: String -> IO Int
email name =
    do  system (”grep ”++name++” addresses>recipient”)
        recipient <- readFile ”recipient”
        system (”emacs message”)
        system (”mail ”++last (words recipient)++
                    ” <message”)
```

Create the String
”grep Hu addresses>recipient”

Create the String
”mail hu@mist.i.u-tokyo.ac.jp <message”

宿題

- 教科書を購入し，第一章を読む．
- Hugs をインストールする．
- Hugsを使ってみる．
  - http://cvs.haskell.org/Hugs/pages/hugsman/basics.html を読む
  - （スライド中の）例を確認する