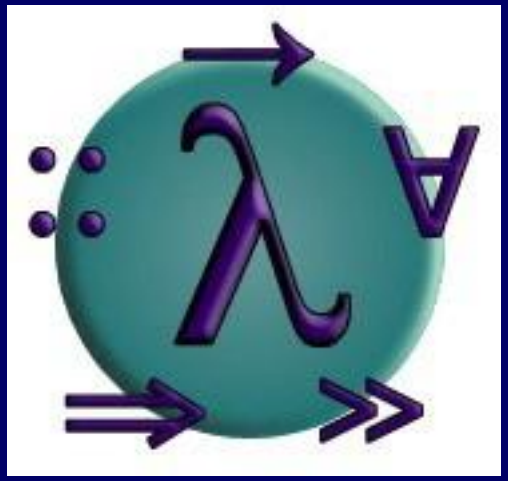
PROGRAMMING IN HASKELL
プログラミングHaskell



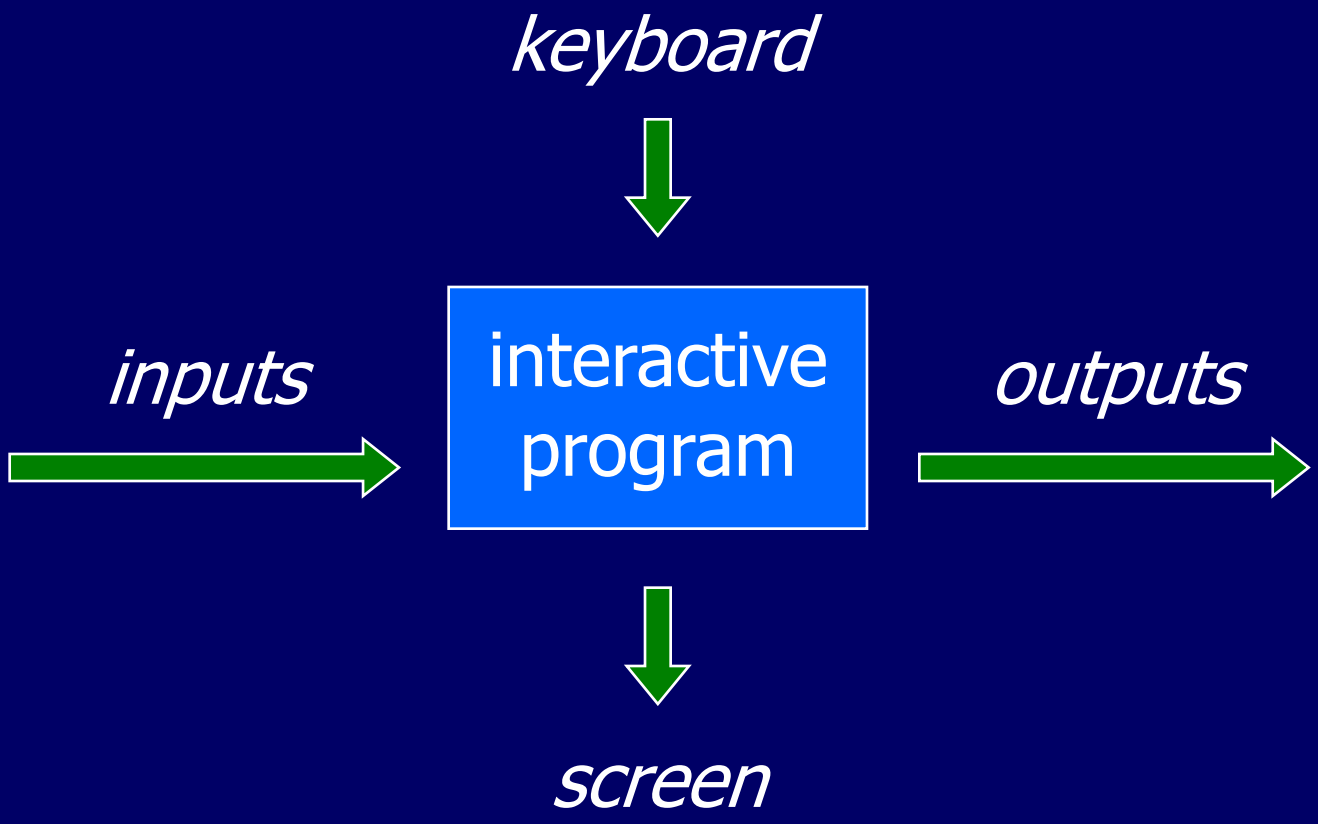
Chapter 9 - Interactive Programs
対話プログラム

愛知県立大学 情報科学部 計算機言語論(山本晋一郎・大久保弘崇、2011年)講義資料
オリジナルは http://www.cs.nott.ac.uk/~gmh/book.html を参照のこと

Introduction

8 章まで、Haskell でバッチ処理プログラムを作る方法を見てきた。このとき、全ての入力は開始時に与えられ、全ての出力は終了時に得られる。

しかし、Haskell で対話的なプログラムも作りたい。このとき、プログラムが動作している間に、キーボードから入力を読み取り、スクリーンに出力される。



問題点

Haskell のプログラムは数学的に純粋な関数である:

- Haskell のプログラムは副作用(side effect)を持たない

しかし、キーボードからの入力やスクリーンへの出力は副作用である:

- 対話的プログラムは副作用を持つ

解決策

Haskell で対話的プログラムを書くとき、純粋な式と副作用をもたらす純粋でないアクションを型によって区別する

```
IO a
```

型 a の値を返すアクションの型

For example:

```
IO Char
```

文字を返すアクションの型

```
IO ()
```

値を返さない、純粋な
副作用のアクションの型

Note:

- () は要素を持たないタプル型(ユニット型、教科書 p.23)

基本アクション

標準ライブラリは以下の 3 つのプリミティブを含む多くのアクションを提供する:

- アクション getChar は、キーボードから 1 文字を読み、スクリーンにエコーバックし、その文字を結果の値として返す:

```
getChar :: IO Char
```

- アクション purChar c は、文字 c をスクリーンに出力し、何も返さない:

```
putChar :: Char → IO ()
```

- アクション return v は、対話処理を行わず、単に値 v を返す:

```
return :: a → IO a
```

アクションの列(連結、Sequencing)

8 章の do とのアナロジー

ひとつながりのアクション群は、予約語 do を用いて
一つの合成アクションに結合することができる

例:

```
a :: IO (Char,Char)
a  = do x ← getChar
        getChar
        y ← getChar
        return (x,y)
```

導出されたアクションの部品

- キーボードから 1 行読み込む:

```
getLine :: IO String
getLine  = do x ← getChar
              if x == '¥n' then
                 return []
               else
                 do xs ← getLine
                    return (x:xs)
```

- 文字列をスクリーンに出力する:

```
putStr        :: String → IO ()
putStr []      = return ()
putStr (x:xs) = do putChar x
                   putStr xs
```

- 文字列をスクリーンに出力して改行する:

```
putStrLn    :: String → IO ()
putStrLn xs = do putStr xs
                 putChar '¥n'
```

Example

文字列の入力を促すプロンプトを表示し、入力された文字列の長さを表示するアクション:

```
strlen :: IO ()
strlen  = do putStr "Enter a string: "
             xs ← getLine
             putStr "The string has "
             putStr (show (length xs))
             putStrLn " characters"
```

For example:

```
> strlen

Enter a string: abcde
The string has 5 characters
```

Note:

アクションを評価すると、その副作用が実行され、
その最終結果の値は捨てられる

ここから教科書に載っていない
ハングマンというゲームの説明

Hangman

次のようなハングマンというゲームを考える:

- プレイヤーが英単語をこっそり入力する
- 別なプレイヤーはその単語を推測し、入力する
- 計算機は、毎回の推測毎に、秘密の単語中のどの文字が、推測された単語に出現するかを示す
- 推測が的中するとゲームは終了

Haskellでハングマンを実装するのに、トップダウンのアプローチを採用し、次のコードから始める:

```
hangman :: IO ()
hangman  =
   do putStrLn "Think of a word: "
      word ← sgetLine
      putStrLn "Try to guess it:"
      guess word
```

アクション sgetLine は、ダッシュ記号をエコーバックしつつ、キーボードから 1 行のテキストを読み込む:

```
sgetLine :: IO String
sgetLine  = do x ← getCh
               if x == '¥n' then
                  do putChar x
                     return []
                else
                  do putChar '-'
                     xs ← sgetLine
                     return (x:xs)
```

Note:

- アクション getCh はキーボードから 1 文字を読み込むが、スクリーンにエコーバックしない
- この便利なアクションは標準ライブラリではなく、Hugs の特別なプリミティブであるため、次のようにインポートする:

```
primitive getCh :: IO Char
```

関数 guess がメインループであり、ゲームが終了するまで、プレイヤーの推測を促して処理を行う

```
guess      :: String → IO ()
guess word =
    do putStr "> "
       xs ← getLine
       if xs == word then
          putStrLn "You got it!"
        else
          do putStrLn (diff word xs)
             guess word
```

関数 diff は、1 つめの文字列中のどの文字が、
2 つめの文字列に出現しているかを示す:

```
diff      :: String → String → String
diff xs ys =
   [if elem x ys then x else '-' | x ← xs]
```

例:

```
> diff "haskell" "pascal"

"-as--ll"
```

まとめ(9章)

- Haskell の式
  - 純粋な式　　大部分の式
  - 純粋でない式　アクション、副作用を伴う
    IO a は型 a を返すアクションの型
    - IO Char　　文字を返すアクションの型
    - IO ()　　値を返さない、純粋な副作用の型
- アクションの部品
  - getChar :: IO Char　　1 文字入力、入力された文字を返す
  - putChar :: Char -> IO ()　　1 文字出力、戻り値なし
  - return v :: a -> IO a　　対話処理を行わず、値 v を返す
  - getLine :: IO String　　行入力、入力された文字列を返す
  - putStr :: String -> IO ()　　行出力、戻り値なし (putStrLn は改行付き)
  - 部品から新しいアクションを構成する
    - do { x ← p1; p2; …; y ← pn; } (連結)

練習問題(9章)

- 教科書 p.109 の strlen を実行して、動作を確認せよ
- スライドに説明されているハングマンを実行して、動作を確認せよ
  - Hangman.hs を講義 Web からダウンロードせよ
    http://www.ist.aichi-pu.ac.jp/lab/yamamoto/program_languages/2011/
- 教科書の電卓またはライフゲームのどちらかを実行して動作を確認すると共に、プログラムを簡単に説明せよ
  - 分からない部分があれば、明示せよ

動作確認方法(Linux)

- http://www.cs.nott.ac.uk/~gmh/book.html の Code から以下のファイルをダウンロードする
  - calculator.lhs (Calculator, 9.6 節まで)
  - life.lhs (Game of life, 9.7 節)
  - Parsing.lhs (Functional parsing library, 8 章と同じ)
- calculator.lhs と Parsing.lhs を同じディレクトリに置く
  - calculator.lhs が Parsing.lhs を import しているため
- Linux 上の ghci で以下を実行する
  - calculator.lhs をロードし、run と入力すれば、教科書の電卓の動作を確認できる
  - life.lhs をロードし、life glider と入力すれば、教科書のライフゲームの動作を確認できる

動作確認方法(Windows, その1)

- http://www.cs.nott.ac.uk/~gmh/book.html の Code から以下のファイルをダウンロードする
  - Parsing.lhs (Functional parsing library, 8 章と同じ)
- http://www.ist.aichi-pu.ac.jp/lab/yamamoto/program_languages/ の 2011 年度から変更済みのファイルをダウンロードする
  - calculatoWin.lhs (Calculator, 9.6 節まで)
  - lifeWin.lhs (Game of life, 9.7 節)
- calculatorWin.lhs と Parsing.lhs を同じディレクトリに置く
  - calculatorWin.lhs が Parsing.lhs を import しているため

動作確認方法(Windows, その2)

- ansi-terminal をインストールする
  - cabal は Haskell Platform に付属するコマンド
  1. コマンドプロンプトを起動する
  2. リポジトリをアップデートする
     $ cabal update
  3. ansi-terminal をインストールする
     $ cabal install ansi-terminal
- Windows の GHCi で以下を実行する
  - 現時点では、WinGHCi で動作しない
  - calculatorWin.lhs をロードし、run と入力すれば、教科書の電卓の動作を確認できる
  - lifeWin.lhs をロードし、life glider と入力すれば、教科書のライフゲームの動作を確認できる

演習問題

Nim というゲームを Haskell で実装せよ。ゲームのルールは以下の通り:

- ゲーム盤は星が入る 5 つの列からなる:

- 2 人のプレイヤーは、交互に、どれか 1 つの行を選び、その末尾から 1 つ以上の星を取り去る

- ゲーム盤上の最後の星を取った方が勝ち

ヒント:

ゲーム盤を、それぞれの列にある星の数を意味する 5 つの整数からなるリストで表す。例えば、初期状態は [5,4,3,2,1] となる。