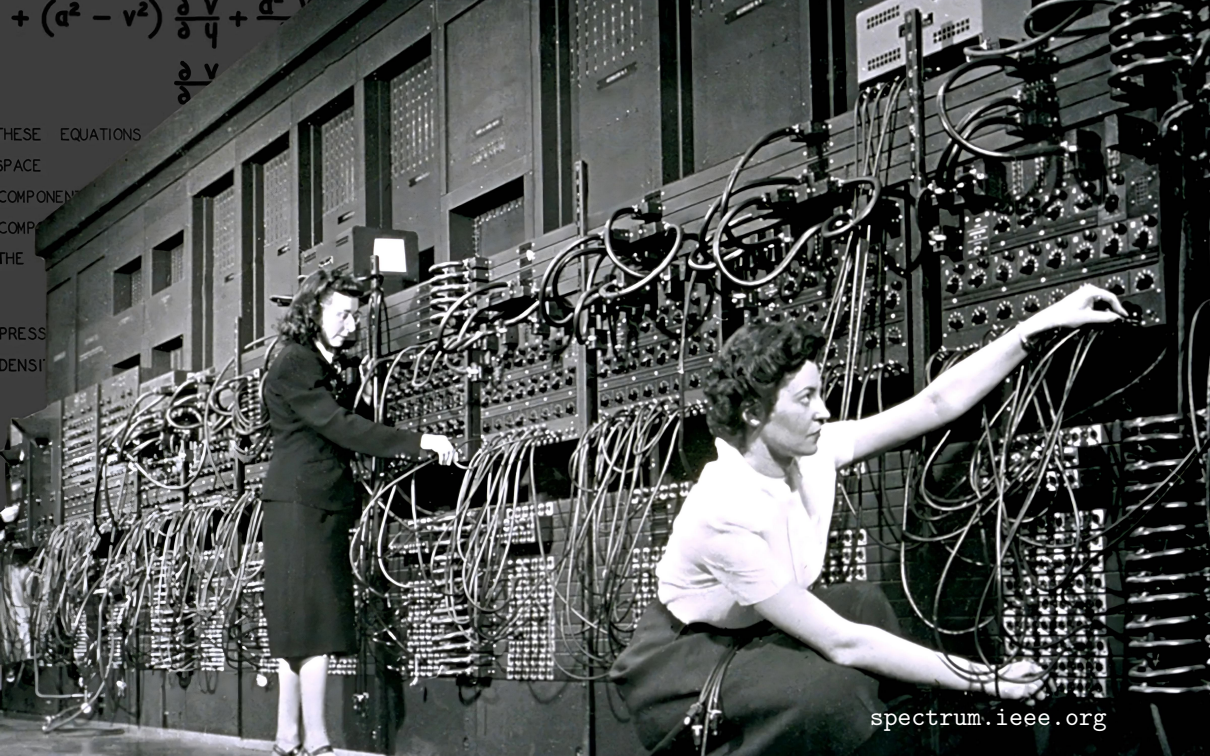
Haskell

Contents


spectrum.ieee.org

Programming

Machine-level programming

- Not portable (depends on specific CPU), error-prone, hard to work with
- Full control

High-level programming

- Code written in style more appealing to humans
- Then transformed into machine code

Interpreter executes code line-by-line

Compiler transforms whole program

Programming

(Thompson: §1)

Imperative Programming

- Oldest and most common control flow paradigm
- Statements give computer orders
- Commands change the program's state (roughly: physical memory)
- Java, C++, JavaScript, Python ...

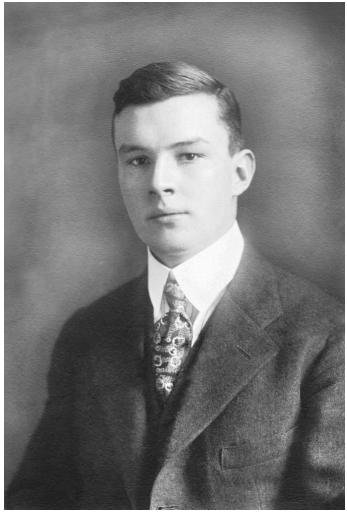
Declarative programming

- Focus on specification of the program you want to write
- Machine level details (e.g. memory usage) handled by programming language
- Not all definitions correspond to good programs!
- Haskell, SQL, Prolog...
  (and supported by most modern programming languages)

About Haskell

(Thompson: §1)

- Functional programming language
- Launched in 1990
- Named after logician Haskell Curry (1900–1982)
- Short history in the article: A history of Haskell: being lazy with class



https://iep.utm.edu/haskell-brooks-curry/

The essence of functional programming

(Thompson: §1)

Types play the role of **sets**

- Sets of functions, sums, products are similar in both types and sets

Programs inhabit types, so they are like the **elements** of sets

- Programs usually have function type (from input to output)
- We think of a program not of function type as a function without inputs
- All functions are **partial**: they can crash or fail to return an output

Functional programming in practice

(Thompson: §1)

- Types model objects in the problem domain
- Programming means defining types and writing functions between them
- Computing with functions means evaluation (reduction), rather than executing a sequence of instructions
- Variables name values, and **do not change** their contents

Haskell is functional

(Thompson: §1)

Functions in Haskell are ...

- first-class
- values which can be used in computations

Haskell is pure

(Thompson: §1)

Functions expressions are **pure**

- expessions have **no side-effects**
  - So everything that might effect the running of your program needs to be explicitly considered as an 'input'
  - With **impure** languages, your program might depend implicitly on something like the state of memory
- programs are **deterministic**: using a function twice with the same input yields the same output

Haskell is lazy

(Thompson: §1)

Expressions are not evaluated **until their results are needed**

- Makes is possible to define and work with infinite data structures
- Enables more compositional programming style
- Makes reasoning about time and space usage more difficult

Haskell in the industry

Haskell has a reputation as an academic language, but it is also used in industry:

- The Sigma anti-spam system at Meta (Facebook)
- The virtual machine management tool Ganeti (Google)
- The production serialisation system Bond (Microsoft)

Haskell is also used by ABN AMRO, AT&T, Barclays, Detexify, intel, NVIDIA, and many more companies

Why we teach Haskell

Reasoning about correctness

- Close to maths
- Pure, so we can reason compositionally (piece by piece)

Well supported

- Good compiler errors
- Good documentation

Important ideas

- Types
- unctional programming

Scientifically important New language developments often use Haskell

Clean slate New language for (almost) everyone in the course

Evaluating expressions

(Thompson: §1.10)

Evaluation any Haskell expression works much like a calculator

```
input:      42.0
output:     42.0

input:      3 + 4 / (1234 - 1)
reduction:  3 + 4 / 1233
reduction:  3 + 3.2441200324412004e-3
reduction:  3.0032441200324413
output:     3.0032441200324413
```

Evaluating expressions

(Thompson: §1.10)

Evaluation any Haskell expression works much like a calculator

```
1  double :: Integer -> Integer
2  double n = 2 * n
```

```
input:       double (3 + 1)
reduction:   2 * (3 + 1)
reduction:   2 * 4
reduction:   8
output:      8
```

Applying a function

(Thompson: §1.3, §1.4)

```
invert :: Picture -> Picture
knight :: Picture

invert knight :: Picture
```

Applying a function

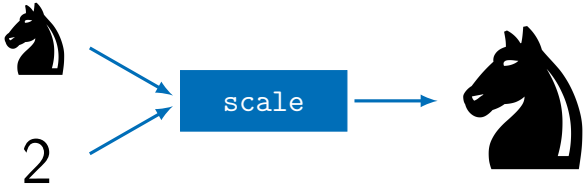
(Thompson: §1.3, §1.4)

```
invert :: Picture -> Picture
knight :: Picture

invert knight :: Picture
```

Applying a function

(Thompson: §1.2, §1.3)

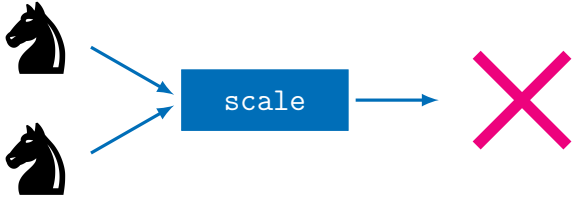
```
scale  :: Picture -> Integer -> Picture
knight :: Picture

scale knight :: Integer -> Picture
scale knight 2 :: Picture
```



Applying a function

(Thompson: §1.2, §1.3)

```
scale  :: Picture -> Integer -> Picture
knight :: Picture

scale knight knight :: ???
```



Summary

Haskell is ...

- a **functional** programming language (everything is a function)
- **pure** (no side-effects)
- **lazy** (expressions only evaluated when needed)
- **typed** (everything has a type)
- used in academia as well as industry

Next lectures

- Wednesday: Types
- Thursday: Algebraic datatypes