Functional Programming in Education

George Wilson

Data61/CSIRO

george.wilson@data61.csiro.au

15th May 2018

University
First year, first semester

Which language?

```java
class Hello {

  public static void main(String[] args) {
    System.out.println("Hello, world!");
  }

}
```

	Content
Week 1	Basic expressions
Week 2	procedure declarations
Week 3	if-statement
Week 4	while-statement
Week 5	for-statement
...	

Structure and Interpretation of Computer Programs

Second Edition

Harold Abelson and
Gerald Jay Sussman
with Julie Sussman

```scheme
(define (factorial n)
  (if (<= n 1)
      1
      (* n (factorial (- n 1)))))
```

Evaluation by substitution

```
(define (sum-of-squares x y)
  (+ (sqr x) (sqr y)))
```

Evaluation by substitution

```
(define (sum-of-squares x y)
  (+ (sqr x) (sqr y)))
```

```
(sum-of-squares 3 4)
```

Evaluation by substitution

```
(define (sum-of-squares x y)
  (+ (sqr x) (sqr y)))
```

```
     (sum-of-squares 3 4)
=>   (+ (sqr 3) (sqr 4))
```

Evaluation by substitution

```
(define (sum-of-squares x y)
  (+ (sqr x) (sqr y)))
```

```
     (sum-of-squares 3 4)
=>   (+ (sqr 3) (sqr 4))
=>   (+ (* 3 3) (sqr 4))
```

Evaluation by substitution

```
(define (sum-of-squares x y)
  (+ (sqr x) (sqr y)))
```

```
     (sum-of-squares 3 4)
=>   (+ (sqr 3) (sqr 4))
=>   (+ (* 3 3) (sqr 4))
=>   (+ 9 (sqr 4))
```

Evaluation by substitution

```
(define (sum-of-squares x y)
  (+ (sqr x) (sqr y)))
```

```
     (sum-of-squares 3 4)
=>   (+ (sqr 3) (sqr 4))
=>   (+ (* 3 3) (sqr 4))
=>   (+ 9 (sqr 4))
=>   (+ 9 (* 4 4))
```

Evaluation by substitution

```
(define (sum-of-squares x y)
  (+ (sqr x) (sqr y)))
```

```
     (sum-of-squares 3 4)
=>   (+ (sqr 3) (sqr 4))
=>   (+ (* 3 3) (sqr 4))
=>   (+ 9 (sqr 4))
=>   (+ 9 (* 4 4))
=>   (+ 9 16)
```

Evaluation by substitution

```
(define (sum-of-squares x y)
  (+ (sqr x) (sqr y)))
```

```
     (sum-of-squares 3 4)
=>   (+ (sqr 3) (sqr 4))
=>   (+ (* 3 3) (sqr 4))
=>   (+ 9 (sqr 4))
=>   (+ 9 (* 4 4))
=>   (+ 9 16)
=>   25
```

```
(factorial 6)
(* 6 (factorial 5))
(* 6 (* 5 (factorial 4)))
(* 6 (* 5 (* 4 (factorial 3))))
(* 6 (* 5 (* 4 (* 3 (factorial 2)))))
(* 6 (* 5 (* 4 (* 3 (* 2 (factorial 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 1)))))
(* 6 (* 5 (* 4 (* 3 2))))
(* 6 (* 5 (* 4 6)))
(* 6 (* 5 24))
(* 6 120)
720
```

Incredible breadth of content

complexity analysis

symbolic computation with quotation

interpreters

object-oriented programming

logic programming

many other concepts

R

received April 198

A critique of Abelson and Sussman
- or -
Why calculating is better than scheming

Philip Wadler
Programming Research Group
11 Keble Road
Oxford, OX1 3QD

(or similar)

```haskell
data List a
  = Nil
  | Cons a (List a)
```

```scheme
(define (sum items)
  (cond ((null? items) 0)
        (else (+ (car items) (sum (cdr items))))))
```

```scheme
(define (sum items)
  (cond ((null? items) 0)
        (else (+ (car items) (sum (cdr items))))))
```

```haskell
sum items = case items of
  Nil       -> 0
  Cons x xs -> x + sum xs
```

```scheme
(define (new-if predicate then-clause else-clause)
  (cond (predicate then-clause)
        (else else-clause)))
```

```scheme
(define (new-if predicate then-clause else-clause)
  (cond (predicate then-clause)
        (else else-clause)))
```

```haskell
newIf True  t f = t
newIf False t f = f
```

The Structure and Interpretation of the Computer Science Curriculum

Matthias Felleisen, Northeastern University, Boston, MA, USA

Robert Bruce Findler, University of Chicago, Chicago, IL, USA

Matthew Flatt, University of Utah, Salt Lake City, UT, USA

Shriram Krishnamurthi, Brown University, Providence, RI, USA

Email: {matthias,robby,mflatt,shriram}@plt-scheme.org

Criticisms

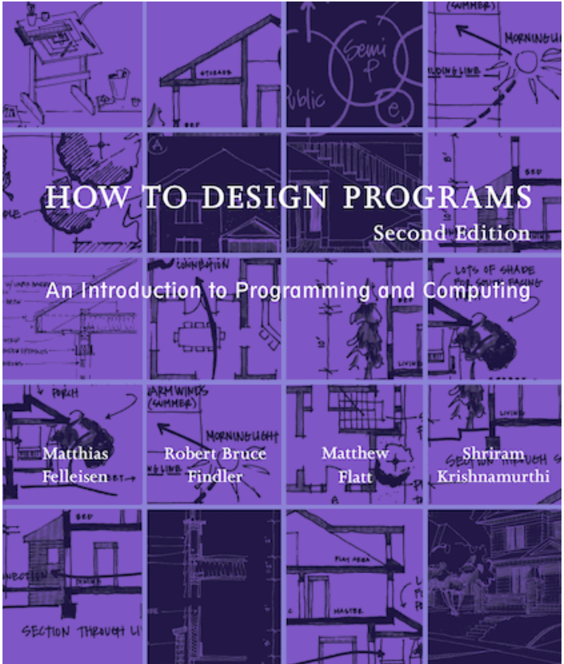
Examples are drawn from overly-technical domains

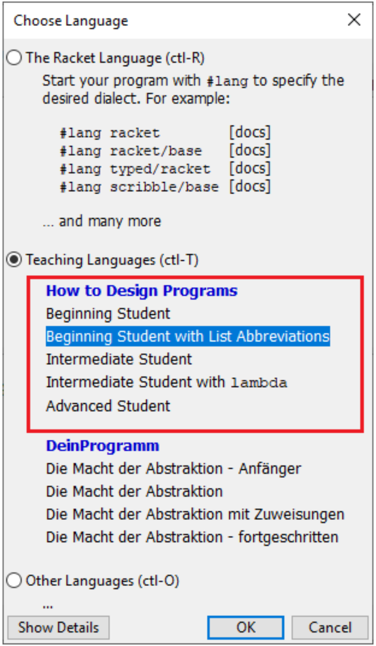
```scheme
(define (deriv exp var)
  (cond ((number? exp) 0)
        ((variable? exp)
         (if (same-variable? exp var) 1 0))
        ((sum? exp)
         (make-sum (deriv (addend exp) var)
                   (deriv (augend exp) var)))
        ((product? exp)
         (make-sum
           (make-product (multiplier exp)
                         (deriv (multiplicand exp) var))
           (make-product (deriv (multiplier exp) var)
                         (multiplicand exp))))
        (else
         (error "unknown expression type -- DERIV" exp))))
```

Criticisms

Lacking coverage of foundational problem-solving techniques

> From an educational point of view, our experience suggests that undergraduate computer science courses should emphasize basic notions of modularity, specification, and data abstraction, and should not let these be displaced by more advanced topics, such as design patterns, object-oriented methods, concurrency, functional languages, and so on.
>
> — Jackson and Chapin, 2000 (emphasis mine)


HOW TO DESIGN PROGRAMS
Second Edition
An Introduction to Programming and Computing
Matthias Felleisen
Robert Bruce Findler
Matthew Flatt
Shriram Krishnamurthi


Choose Language
The Racket Language (ctl-R)
Start your program with #lang to specify the desired dialect. For example:
#lang racket [docs]
#lang racket/base [docs]
#lang typed/racket [docs]
#lang scribble/base [docs]
... and many more
Teaching Languages (ctl-T)
How to Design Programs
Beginning Student
Beginning Student with List Abbreviations
Intermediate Student
Intermediate Student with lambda
Advanced Student
DeinProgramm
Die Macht der Abstraktion - Anfänger
Die Macht der Abstraktion
Die Macht der Abstraktion mit Zuweisungen
Die Macht der Abstraktion - fortgeschritten
Other Languages (ctl-O)
...
Show Details
OK
Cancel

prologue.rkt - DrRacket

File Edit View Language Racket Insert Scripts Tabs Help

prologue.rkt ▾ (define ...) ▾

```
(require 2htdp/image)
(require 2htdp/universe)
```

(define rocket )

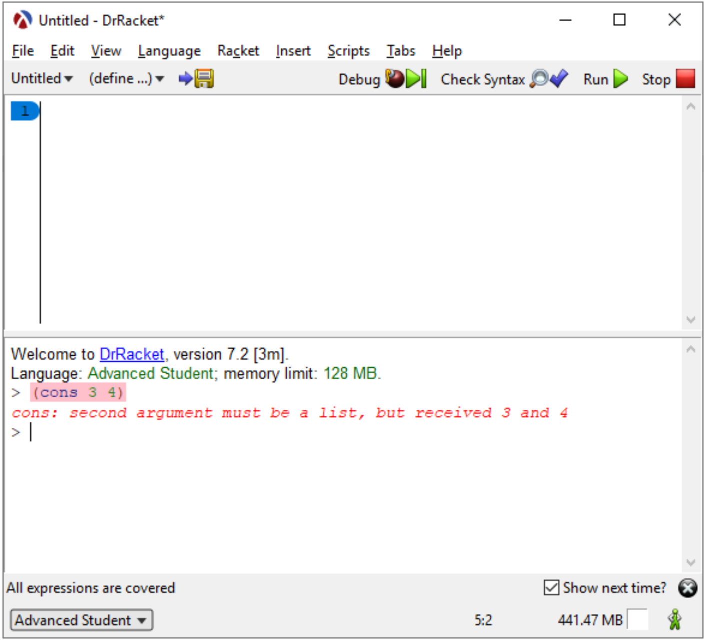
```
(define (picture-of-rocket height)
  (place-image rocket 50 height (empty-scene 100 60)))

(define (sign x)
  (cond ((> x 0) 1)
        ((= x 0) 0)
        ((< x 0) -1)))

(define (picture-of-rocket.v2 height)
  (cond
    [(<= height 60)
     (place-image  50 height
                  (empty-scene 100 60))]
    [(> height 60)
     (place-image  50 60
                  (empty-scene 100 60))]))
```

Welcome to DrRacket, version 7.2 [3m].
Language: Beginning Student; memory limit: 128 MB.

```
> (picture-of-rocket.v2 5555)
place-image: expects 4 arguments, but found only 3
> 
```


Untitled - DrRacket*
File Edit View Language Racket Insert Scripts Tabs Help
Untitled (define ...)
Debug Check Syntax Run Stop
1
Welcome to DrRacket, version 7.2 [3m].
Language: Advanced Student; memory limit: 128 MB.
> (cons 3 4)
cons: second argument must be a list, but received 3 and 4
>
All expressions are covered
Show next time?
Advanced Student
5:2
441.47 MB

Strings aren't compared with `=` and its relatives. Instead, you must use `string=?` or `string<=?` or `string>=?` if you ever need to compare strings. While it is obvious that `string=?` checks whether the two given strings are equal, the other two primitives are open to interpretation. Look up their documentation. Or, experiment, guess a general law, and then check in the documentation whether you guessed right.

If the documentation in HelpDesk appears confusing, experiment with the functions in the interactions area. Give them appropriate arguments, and find out what they compute. Also use **inappropriate** arguments for some operations just to find out how BSL reacts:

```
> (string-length 42)
string-length:expects a string, given 42
```

on software design.

1. **From Problem Analysis to Data Definitions**

   Identify the information that must be represented and how it is represented in the chosen programming language. Formulate data definitions and illustrate them with examples.

2. **Signature, Purpose Statement, Header**

   State what kind of data the desired function consumes and produces. Formulate a concise answer to the question *what* the function computes. Define a stub that lives up to the signature.

3. **Functional Examples**

   Work through examples that illustrate the function's purpose.

4. **Function Template**

   Translate the data definitions into an outline of the function.

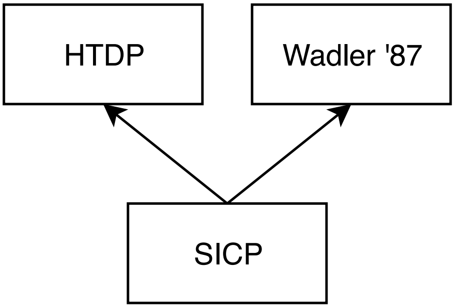
5. **Function Definition**

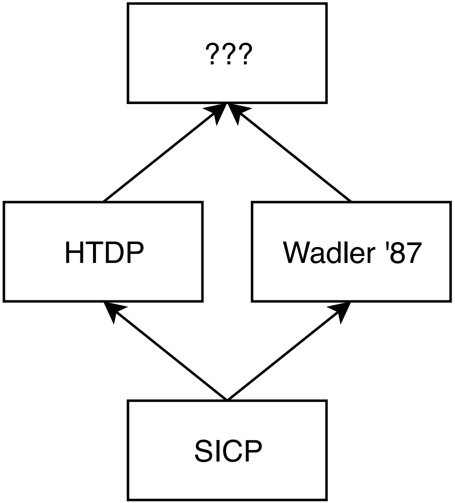
   Fill in the gaps in the function template. Exploit the purpose statement and the examples.

6. **Testing**

   Articulate the examples as tests and ensure that the function passes all. Doing so discovers mistakes. Tests also supplement examples in that they help others read and understand the definition when the need arises—and it will arise for any serious program.

Figure 1: The basic steps of a function design recipe


HTDP
Wadler '87
SICP


???
HTDP
Wadler '87
SICP

3 + False

```
<interactive>:1:1: error:
    • No instance for (Num Bool) arising from a use of ‘+’
    • In the expression: 3 + False
      In an equation for ‘it’: it = 3 + False
```

GHC custom type errors

```haskell
{-# language DataKinds, TypeFamilies, TypeOperators #-}
{-# language UndecidableInstances #-}

import GHC.TypeLits

instance TypeError (Text "Booleans are not numbers" :$$:
                    Text "so we cannot add or multiply them")
  => Num Bool where
```

3 + False

```
<interactive>:1:1: error:
    • Booleans are not numbers
      so we cannot add or multiply them
    • In the expression: 3 + False
      In an equation for ‘it’: it = 3 + False
```

Custom preludes for a staged introduction

Prelude.hs

```haskell
module Prelude
  ( Integer, (+)
  )
where

import GHC.Num (Integer)
import qualified GHC.Num as N

(+) :: Integer -> Integer -> Integer
(+) = (N.+)
```

A brief personal anecdote. . .

Thanks for listening!

References

- Structure and Interpretation of Computer Programs
  Harold Abelson and Gerald Jay Sussman with Julie Sussman
- A Critique of Abelson and Sussman
  Philip Wadler
- The Structure and Interpretation of the Computer Science Curriculum
  Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi
- How to Design Programs
  Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi
- The Risks and Benefits of Teaching Purely Functional Programming in First Year
  Manuel Chakravarty and Gabriele Keller