CS 3520/6520
Programming Languages

Fall 2019

Instructor: **Matthew Flatt**

TAs: **Zhengyang Liu**
Yuwei Wang
Kurt Bruns

CS 3520/6520 Programming Languages

This course is about programming language ***concepts***

It's *not* a survey course

CS 3520/6520 Programming Languages

This course is about programming language ***concepts***

lexical scope	closures	recursion
λ-calculus	objects	classes
continuations	eager and lazy evaluation	
state	type checking	polymorphism
soundness	type inference	subtyping
compilation	garbage collection	

... especially ***functional programming*** concepts

CS 3520/6520 Programming Languages

This course is about programming language ***concepts***

- To help you understand new programming languages
- To make you a better programmer in any language

Course Details

`http://www.eng.utah.edu/~cs3520/`

Formal prerequisite: CS 3500

Informal prerequisite: more programming experience than that

Lectures are Online

After today, all slide presentations are online

- Watch the videos before class

- Class is for more examples and homework solutions
  - a.k.a. "recitation"
  - guideline: no new material introduced in class

Programming Language Concepts

This course teaches concepts in two ways:

- By implementing **interpreters**
  - new concept ⇒ new interpreter
- By using **Plait**, a variant of **Racket**
  - we don't assume that you already know Plait or Racket

Interpreters

An ***interpreter*** takes a program and produces a result

- DrRacket
- x86 processor
- desktop calculator
- **`bash`**
- Algebra student

A ***compiler*** takes a program and produces another program

In the terminology of programming languages, someone who translates Chinese to English is a *compiler*, not an *interpreter*.

Racket and Plait

Lisp ➡ **Scheme** ➡ **Racket**

Racket is

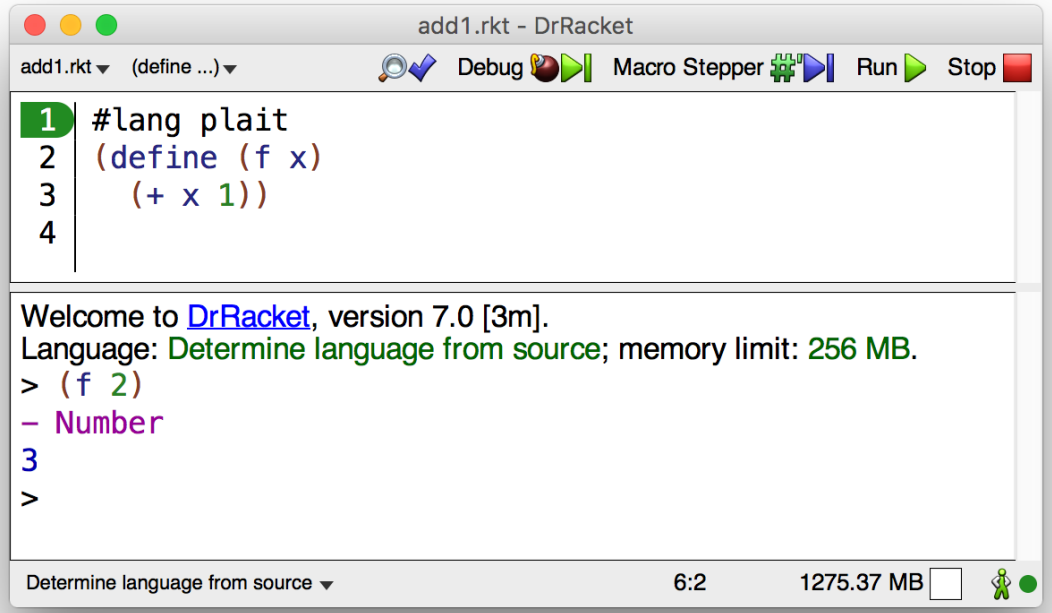
- a programming language
- a family of programming languages
- a language for creating programming languages

... including **Plait**

Racket ➡ **Plait** ⬅ **ML**

PLAI = *Programming Languages: Application and Interpretation*, a textbook

DrRacket


add1.rkt - DrRacket
add1.rkt
(define ...)
Debug
Macro Stepper
Run
Stop
1 #lang plait
2 (define (f x)
3 (+ x 1))
4
Welcome to DrRacket, version 7.0 [3m].
Language: Determine language from source; memory limit: 256 MB.
> (f 2)
- Number
3
>
Determine language from source
6:2
1275.37 MB

Plait Tutorial

http://docs.racket-lang.org/plait/index.html

v.7.0

Plait Language

```
#lang plait                                                    package: plait
```

The Plait language syntactically resembles the `plai` language, which is based on `racket`, but the type system is close to that of ML.

1 Tutorial

Plait's Parenthesized Prefix Notation

```
f(x)
```

```
(f x)
```

```
1+2
```

```
(+ 1 2)
```

```
1+2*3
```

```
(+ 1 (* 2 3))
```

```
s=6
```

```
(define s 6)
```

```
f(x)=x+1
```

```
(define (f x)
  (+ x 1))
```

$$\begin{cases} x<0 & -1 \\ x=0 & 0 \\ x>0 & 1 \end{cases}$$

```
(cond
  [(< x 0) -1]
  [(= x 0) 0]
  [(> x 0) 1])
```

Plait Data

- Numbers and strings — obvious

```
1   3.4   "Hello, World!"
```

- Booleans — straightforward

```
#t   #f
```

- Symbols and quoted lists — unusual

```
'apple   'define   '+

'(1 2 3)   '(f x)
```

Plait S-Expressions

- Backquote ` instead of regular quote '

convenient

```
`x
```

```
`{+ x 1}
```

```
`{define {f x}
   {+ x 1}}
```

Plait Datatypes

```
(define-type Shape
  (circle [radius : Number])
  (rectangle [width : Number]
             [height : Number]))

(define (area s)
  (type-case Shape s
    [(circle r) (* 3.14 (* r r))]
    [(rectangle w h) (* w h)]))

(test (area (circle 2))
      12.56)
(test (area (rectangle 3 4))
      12)
```

Interpreters

See **`lambda.rkt`**

Example Plait program:

```
(define-type Value
  (numV [n : Number])
  (closV [arg : Symbol]
         [body : Exp]
         [env : Env]))
```

Example **Curly** program:

```
{+ {* 3 4} 8}
```

Example Curly program as a Plait value:

```
`{+ {* 3 4} 8}
```

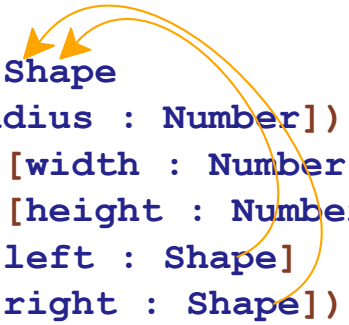
Datatype and Function Shapes Match

```
(define-type Shape
  (circle [radius : Number])
  (rectangle [width : Number]
             [height : Number])
  (adjacent [left : Shape]
            [right : Shape]))

(define (area s)
  (type-case Shape s
    [(circle r) (* 3.14 (* r r))]
    [(rectangle w h) (* w h)]
    [(adjacent l r) (+ (area l)
                       (area r))]))

(test (area (circle 2))
      12.56)
(test (area (rectangle 3 4))
      12)
(test (area (adjacent (circle 2) (rectangle 3 4)))
      24.56)
```

Datatype and Function Shapes Match

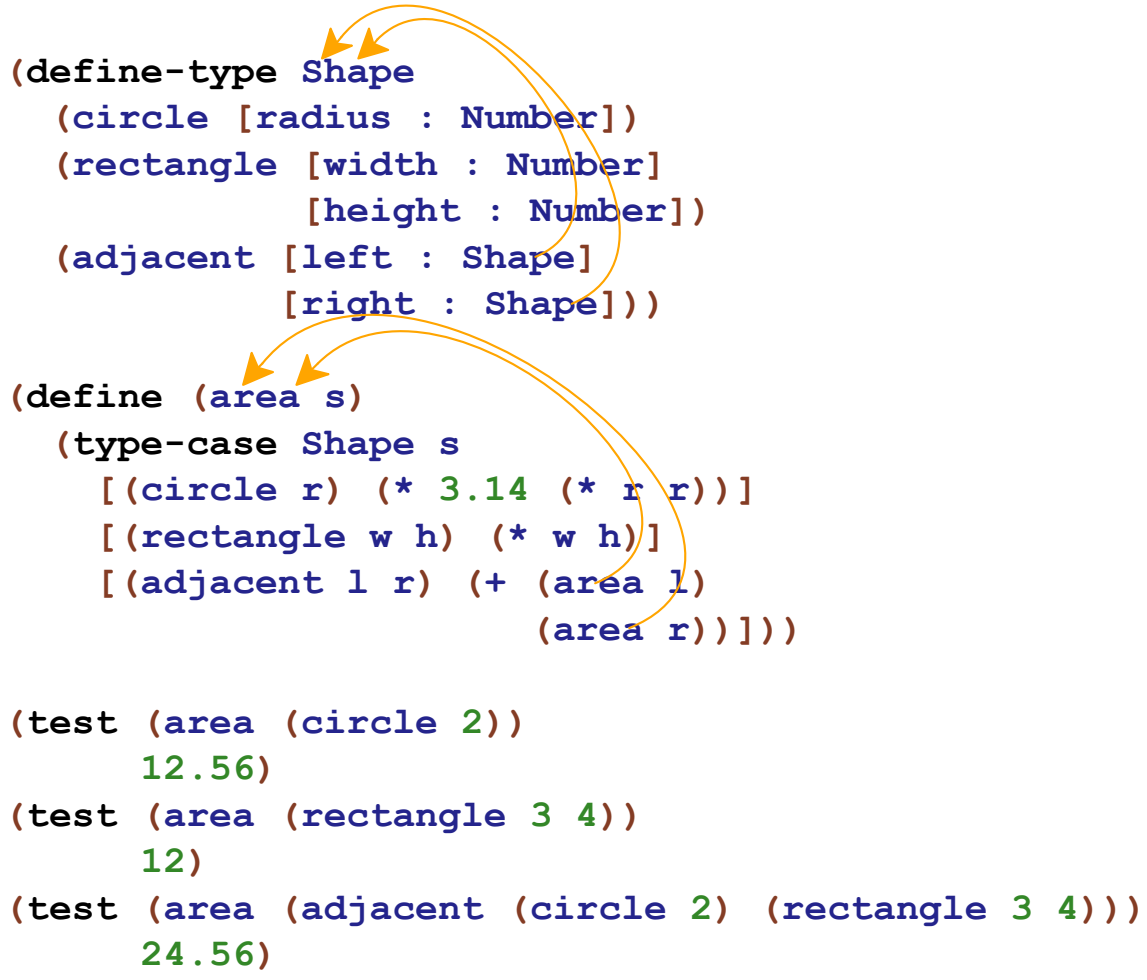


```
(define-type Shape
  (circle [radius : Number])
  (rectangle [width : Number]
             [height : Number])
  (adjacent [left : Shape]
            [right : Shape]))

(define (area s)
  (type-case Shape s
    [(circle r) (* 3.14 (* r r))]
    [(rectangle w h) (* w h)]
    [(adjacent l r) (+ (area l)
                       (area r))]))

(test (area (circle 2))
      12.56)
(test (area (rectangle 3 4))
      12)
(test (area (adjacent (circle 2) (rectangle 3 4)))
      24.56)
```

Datatype and Function Shapes Match



```
(define-type Shape
  (circle [radius : Number])
  (rectangle [width : Number]
             [height : Number])
  (adjacent [left : Shape]
            [right : Shape]))

(define (area s)
  (type-case Shape s
    [(circle r) (* 3.14 (* r r))]
    [(rectangle w h) (* w h)]
    [(adjacent l r) (+ (area l)
                       (area r))]))

(test (area (circle 2))
      12.56)
(test (area (rectangle 3 4))
      12)
(test (area (adjacent (circle 2) (rectangle 3 4)))
      24.56)
```

Homework 0

- Create handin account
- Plait warm-up exercises

Due Friday, August 23