PP1
Lecture 1

Introduction to Functional Programming in Scheme

The goals of PP1, lecture 1

- Knowledge
  - The basic parts of Scheme
  - List-based data types
  - The concepts of functions and closures including lambda expressions.
  - Name binding constructs

- Skills:
  - To be able to use list-based datatypes to express and manipulate data in Scheme
  - To be able to write simple functions
  - To be able to use (simple) recursive programming techniques to express algorithms in Scheme

Functional Programming in Scheme

- Includes
  - Pure functions
  - Powerful functions
    - Higher-order, closures, lambda expressions
  - S-expressions and lists
  - Simple types, numeric types, chars, strings, …
  - Dynamic typing
  - Uniformity and minimalism

Functional Programming in Scheme

- Excludes
  - Commands, in particular assignment
  - Control structures
  - Procedures
  - Mutation of existing data objects

The mindset of the functional Scheme programmer

- Frame the solution to your problem as the returned result of a function
- Work with lists as much as possible
- Decompose functions into smaller functions
  - Use a standard repertoire of (higher-order) library functions as much as possible.
- If you need to modify an object (probably a list) arrange for creation of a modified copy

The mindset of the functional Scheme programmer

- Iterate by means of recursion
  - Certain recursive patterns can be abstracted away, typically via use of higher-order functions
- A copy of an object is as good as the original object
  - You cannot tell the difference between structural equality and reference equality

Programs as lists in Lisp languages

- PROGRAM = DATA = LIST
  - Another syntactic idea
- A bridge between source programs and data in the running program
- Do we know of other languages wich represent programs by use of its primary data structure?

Expressions

- Fully parentherized
- Prefix notation

} Uniformity

No operators, no precedence rules, no associativity rules

Keeping track of parentheses

- The use of parentheses makes the program structure *explicit.*
  - No ambiguities
  - Easy parsing (almost trivial parsing)
  - Easy structure-editing

Parentheses are your friends

Keeping track of parentheses

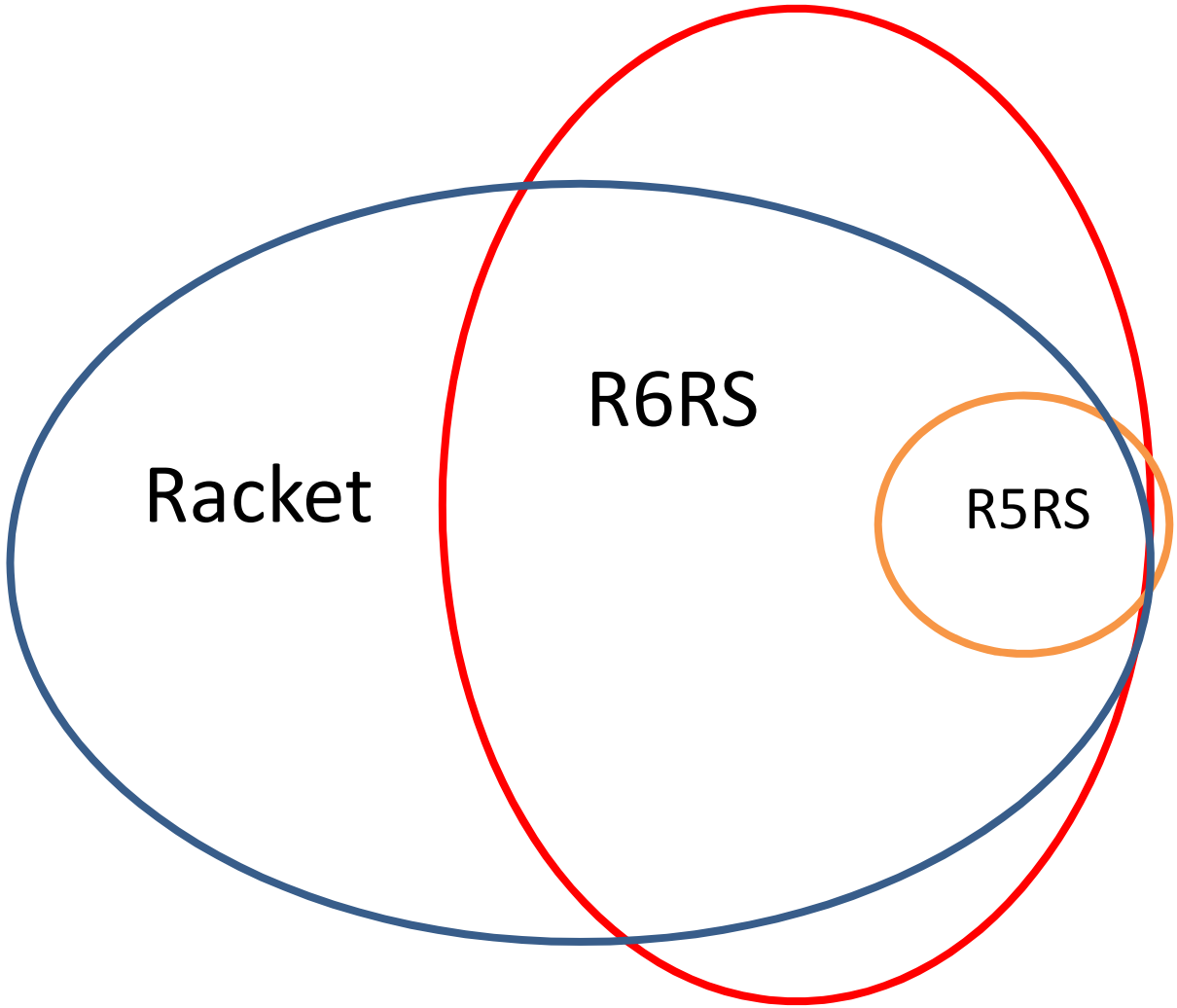
- You must come to a point where the parentheses are *no longer part of the problem.*
  - But rather seen as a *pleasant property of the solution*.

Define forms

```
(define incr (lambda (x) (+ x 1)))
```

```
(define (incr x) (+ x 1))
```

They are equivalent!


Racket
R6RS
R5RS

Group exercises today

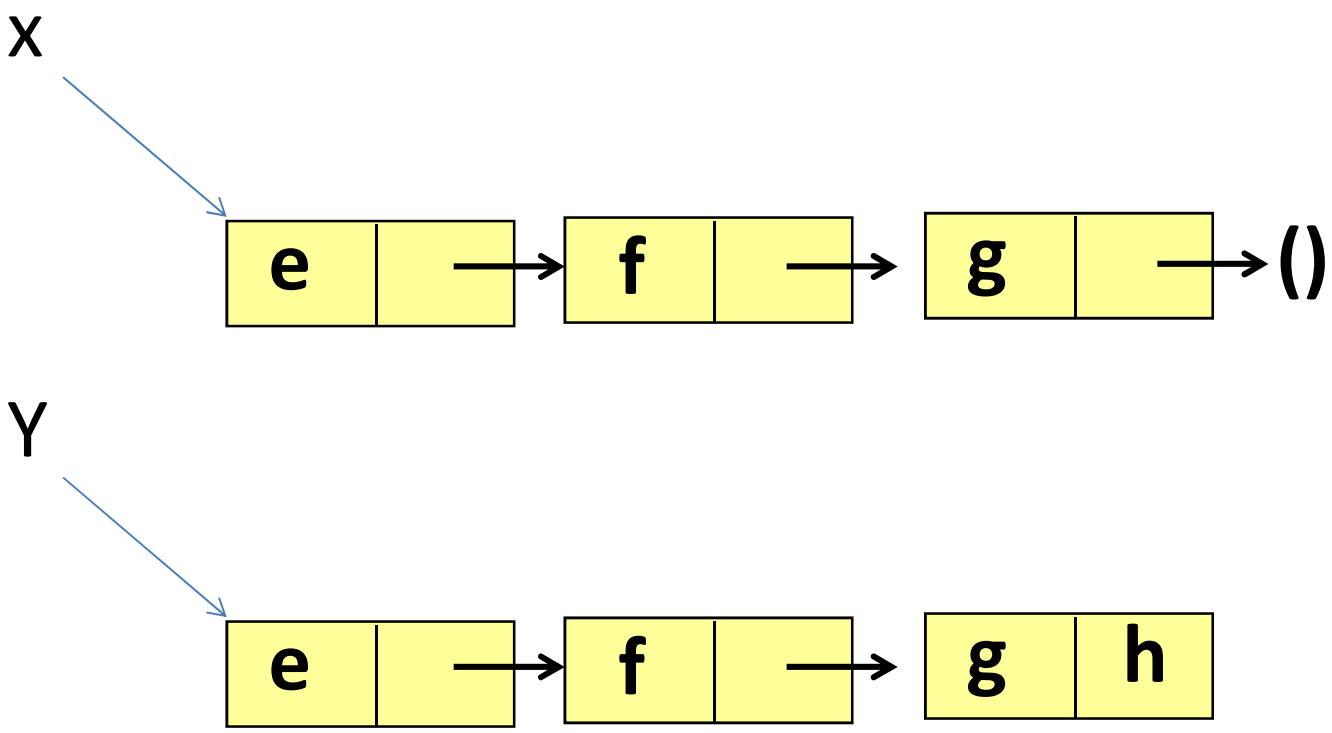
Plan for today (10:15 – 12:00)

- Short intro – 10 minutes
- Exercise 1.3:
  - Proper list predicate – 15 minutes
- Exercise 1.5:
  - Every second (n'th) element of a list – 15 minutes
- BREAK – 10 minutes
- Exercise 1.7 and 1.8 – 15 minutes
  - Property lists, transformations from a-list, get-prop
- Exercise 1.11 – 20 minutes
  - my-list-tail and list-prefix
- Short outro

15 minutes

Exercise 1.3

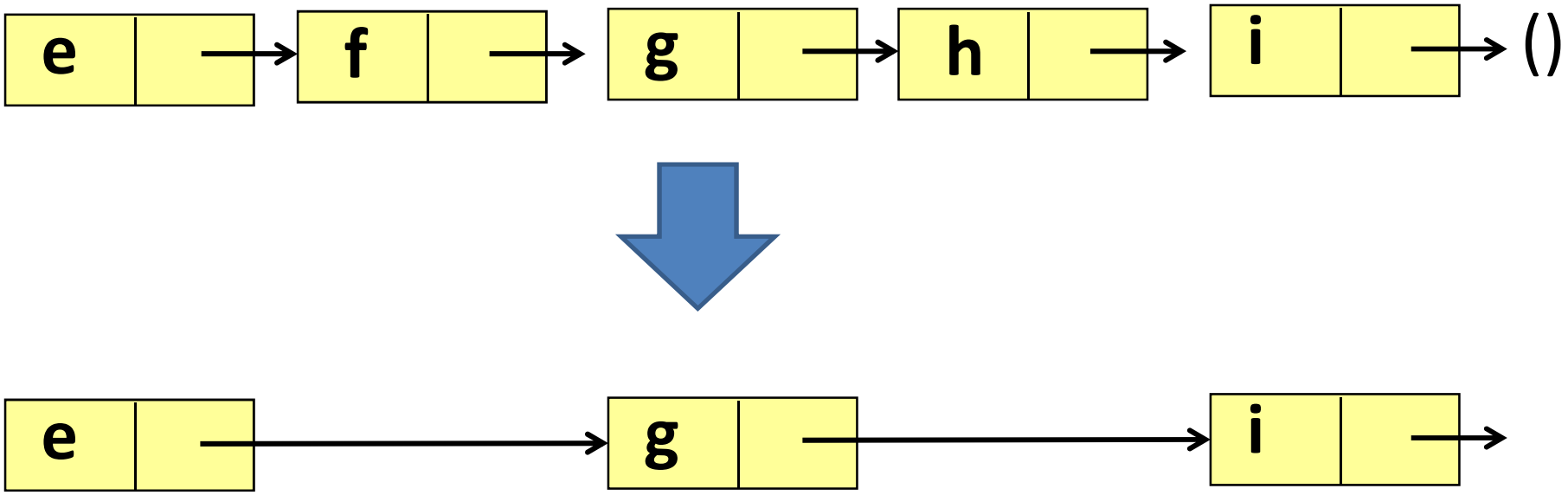
- A proper list predicate, like `list?`



15 minutes

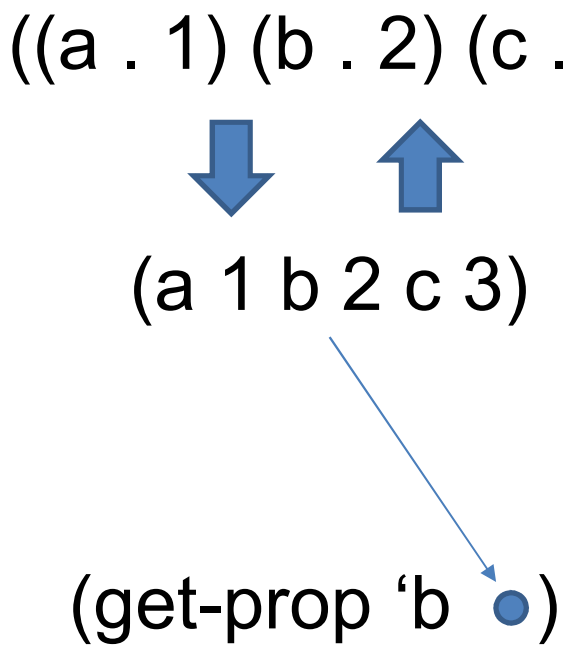
Exercise 1.5

- Every second element of a list



15 minutes

Exercise 1.7 and 1.8



20 minutes

Exercise 1.11

(a b c)

(d e f g)

```
(list-prefix lst 3)
```

```
(my-list-tail lst 3)
```

Outro
The PP1 miniproject

PP1 miniproject

- 15 hours of workload
- Submission deadline October 15
- PP1 miniproject working day: October 13
  - TA support
- The miniproject assignment of PP1 is already posted on Moodle

PP1 miniproject

- Given 200 students
- Various forms of group formations
- Concepts: student, group, grouping
  - List-based representation of these
  - Constructors, predicates and selectors
- A few variants of *group formations*
  - These serve as constructors for groupings
- A few simple functions on groupings