Lecture 08: List Recursion and Recursion Diagrams
10:00 AM, Sep 21, 2018

Contents

1 Input/Output terminology for procedures
2 Practice Writing Recursive Procedures

2.1 Parity
2.1.1 I/O Specification
2.1.2 Non-Recursive Solution
2.1.3 Recursion Diagram
2.1.4 Recursive Solution
2.2 Double
2.2.1 I/O Specification
2.2.2 Recursion Diagram
2.2.3 Code
2.3 Is there a doctor in the house?
2.3.1 I/O Specification
2.3.2 Recursion Diagram
2.3.3 Code

1 Input/Output terminology for procedures

- “input”, “output”
- “take”, “yield”, ...

2 Practice Writing Recursive Procedures

2.1 Parity

2.1.1 I/O Specification

Input: natural number n
Output: the symbol odd if n is odd, the symbol even if n is even.
type signature: $\mathbb{N} \rightarrow \{\text{odd, even}\}$

2.1.2 Non-Recursive Solution

A good way to write this:

```scheme
(define parity
  (lambda (n)
    ((odd? n) (quote odd))
    ((even? n) (quote even))))
```

Now let’s pretend we didn’t have `odd?` and `even?` available to us. There is another good method: using arithmetic operators.

Just for kicks, let’s write it the wrong way: recursively!

2.1.3 Recursion Diagram

First, let’s draw some recursion diagrams.

Original input: 17

- Recursive input: 16
- Recursive output: even

Original output: odd

Note: We can see from above that the original output is the opposite of the recursive output.

2.1.4 Recursive Solution

```scheme
(define parity
  (lambda (n)
    (cond
      ((zero? n) (quote even))
      ((equal? (parity (- n 1)) (quote even)) (quote odd))
      ((equal? (parity (- n 1)) (quote odd)) (quote even))))
```

- Most of the time you should first try relating the case of $n$ to the case of $n - 1$ but every now and then you will find a more creative way of getting the recursive input from the original input.

2.2 Double

2.2.1 I/O Specification

First, let’s write our I/O specification:
Input: List $L$
Output: a list consisting of the elements of $L$, in the same order, but with each element occurring twice.

Example Input: (A (B) C)
Example Output: (A A (B) (B) C C)

2.2.2 Recursion Diagram

Original Input: (A B C)

Recursive Input: (B C)
Recursive Output: (B B C C)

Original Output: (A A B B C C)

How do we get from the recursive output to the original output? We need to add $A$ onto the recursive output, then add $A$ onto that.

2.2.3 Code

Here is the code for `double`:

```scheme
(define double
  (lambda (L)
    (cond
      ((empty? L) empty)
      ((cons? L) (cons (car L) (cons (car L) (double (cdr L)))))))))
```

2.3 Is there a doctor in the house?

2.3.1 I/O Specification

Procedure contains-doctor?

Input: a list $L$
Output: true if $L$ contains the symbol doctor

2.3.2 Recursion Diagram

Original input: (1 2 doctor)

Recursive input: (2 doctor) Recursive Output: true

Original output: true
We need to be more thoughtful in coming up with illustrative examples to go in our recursion diagrams. It helps to remember the usual structure of a list-processing procedure. To make sure we consider all the possibilities, we should try to make recursion diagrams for each of the possibilities:

<table>
<thead>
<tr>
<th>Original Output</th>
<th>Recursive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

2.3.3 Code

```scheme
(define has-doctor?
  (lambda (L)
    (cond
      ((empty? L) #false)
      ((cons? L) (or (equal? (car L) (quote doctor))
        (has-doctor? (cdr L)))))))
```

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS 17 document by filling out the anonymous feedback form: [http://cs.brown.edu/courses/csci0170/feedback](http://cs.brown.edu/courses/csci0170/feedback).