CS450

Structure of Higher Level Languages

Lecture 5: Lists; quoting

Tiago Cogumbreiro
Today we will learn...

- Being successful in CS 450
- Defining user data-structures
- Serializing code with quote
- Exercises with lists
User data-structures
User data-structures

We can represent data-structures using pairs/lists. For instance, let us build a 3-D point data type.

```scheme
(require rackunit)
(define p (point 1 2 3))
(check-true (point? p))
(check-equal? (list 1 2 3) p)
(check-equal? 1 (point-x p))
(check-equal? 2 (point-y p))
(check-equal? 3 (point-z p))
(check-true (origin? (list 0 0 0)))
(check-false (origin? p))
```
User data-structures

We can represent data-structures using pairs/lists. For instance, let us build a 3-D point data type.

(require rackunit)
(define p (point 1 2 3))
(check-true (point? p))
(check-equal? (list 1 2 3) p)
(check-equal? 1 (point-x p))
(check-equal? 2 (point-y p))
(check-equal? 3 (point-z p))
(check-true (origin? (list 0 0 0)))
(check-false (origin? p))
On data-structures

- We only specified **immutable** data structures
- The effect of updating a data-structure is encoded by **creating/copying** a data-structure
- This pattern is known as a **persistent data structure**
Serializing code
Quoting: a specification

Function \( \text{quote} \ e \) **serializes** expression \( e \). Note that expression \( e \) is **not** evaluated.

- A variable \( x \) becomes a symbol \('x\). You can consider a **symbol** to be a special kind of string in Racket. You can test if an expression is a symbol with function **symbol?**
- A function application \( (e_1 \cdots e_n) \) becomes a list of the serialization of each \( e_i \).
- Serializing a \( \text{(define } x \ \text{e}) \) yields a list with: symbol \('\text{define}'\), the serialization of variable \( x \), and the serialization of \( e \). Serializing \( \text{(define } (x_1 \cdots x_n) \ e) \) yields a list with symbol \('\text{define}'\) followed by a nonempty list of symbols \('x_i'\) followed by serialized \( e \).
- Serializing \( \text{(lambda } (x_1\ldots x_n) \ e) \) yields a list with symbol \('\text{lambda}'\), followed by a possibly-empty list of symbols \( x_i \), and the serialized expression \( e \).
- Serializing a \( \text{(cond } (b_1 \ e_1) \cdots (b_n \ e_n)) \) becomes a list with symbol \('\text{cond}'\) followed by a serialized branch. Each branch is a list with two components: serialized expression \( b_i \) and serialized expression \( e_i \).
Quoting exercises:

- We can write 'term rather than (quote term)
- How do we serialize term (lambda (x) x) with quote?
- How do we serialize term (+ 1 2) with quote?
- How do we serialize term (cond [(> 10 x) x] [else #f]) with quote?
- *Can we serialize a syntactically invalid Racket program?*
Quoting exercises:

- We can write 'term rather than (quote term)
- How do we serialize term (lambda (x) x) with quote?
- How do we serialize term (+ 1 2) with quote?
- How do we serialize term (cond [(> 10 x) x] [else #f]) with quote?
- **Can we serialize a syntactically invalid Racket program? No!** You would not be able to serialize this expression (. Quote only accepts a S-expressions (parenthesis must be well-balanced, identifiers must be valid Racket identifiers, number literals must be valid).
- **Can we serialize an invalid Racket program?**
Quoting exercises:

- We can write 'term rather than (quote term)
- How do we serialize term (lambda (x) x) with quote?
- How do we serialize term (+ 1 2) with quote?
- How do we serialize term (cond [(> 10 x) x] [else #f]) with quote?
- **Can we serialize a syntactically invalid Racket program? No!** You would not be able to serialize this expression (. Quote only accepts a S-expressions (parenthesis must be well-balanced, identifiers must be valid Racket identifiers, number literals must be valid).
- **Can we serialize an invalid Racket program? Yes.** For instance, try to quote the term: (lambda)
Quote example

#lang racket
(require rackunit)
(check-equal? 3 (quote 3)) ; Serializing a number returns the number itself
(check-equal? 'x (quote x)) ; Serializing a variable named x yields symbol 'x
(check-equal? (list '+ 1 2) (quote (+ 1 2))) ; Serialization of function as a list
(check-equal? (list 'lambda (list 'x) 'x) (quote (lambda (x) x)))
(check-equal? (list 'define (list 'x)) (quote (define (x))))
Manipulating quoted terms

Specification

\[ \text{function-dec} = (\text{lambda} (\text{variable}^*) \text{term}^+) \]

- How do we get the parameter list?
- How do we get the body?
- What does \textit{variable}^* mean?
- What does \textit{term}^+ mean?

On HW1 Q.4

- The input format of the quoted term are \textit{precisely} described in the slides of Lecture 3
- You do \textbf{not} need to test recursively if the terms in the body of a function declaration or definition are valid.
- A list, with one symbol \texttt{lambda} followed by zero or more symbols, and one or more terms.
Exercises with lists
Lists: example 1

Summation of all elements of a list

Spec

```
(re require rackunit)
(check-equal? 10 (sum-list (list 1 2 3 4)))
(check-equal? 0 (sum-list (list)))
```
Lists: example 1

Summation of all elements of a list

Spec

```
(require rackunit)
(check-equal? 10 (sum-list (list 1 2 3 4)))
(check-equal? 0 (sum-list (list)))
```

Solution

```
#lang racket
; Summation of all elements of a list
(define (sum-list l)
  (cond [(empty? l) 0]
        [else (+ (first l) (sum-list (rest l)))]))
```
Lists: example 2

Returns a list from n down to 1

Spec

```
(require rackunit)
(check-equal? (list) (count-down 0))
(check-equal? (list 3 2 1) (count-down 3))
```
Lists: example 2

Returns a list from n down to 1

Spec

```
(require rackunit)
(check-equal? (list) (count-down 0))
(check-equal? (list 3 2 1) (count-down 3))
```

Solution

```
#lang racket
(define (count-down n)
  (cond [(<= n 0) (list)]
        [else (cons n (count-down (- n 1)))]))
```
Lists: example 3
Point-wise pairing of two lists

Spec

(require rackunit)
(check-equal? (list (cons 3 30) (cons 2 20) (cons 1 10))
(zip (list 3 2 1) (list 30 20 10)))
(check-equal? (list (cons 3 30) (cons 2 20) (cons 1 10))
(zip (list 3 2 1) (list 30 20 10 5 4 3 2 1)))
(check-equal? (list (cons 3 30) (cons 2 20) (cons 1 10))
(zip (list 3 2 1 90 180 270) (list 30 20 10)))
Lists: example 3

Point-wise pairing of two lists
Lists: example 3

Point-wise pairing of two lists

Solution

```racket
#lang racket
(define list-add cons) (define pair cons)
(define (zip l1 l2)
  (cond [(empty? l1) (list)]
        [(empty? l2) (list)]
        [else
         (list-add
          (pair (first l1) (first l2))
          (zip (rest l1) (rest l2)))]))
```