CSE 331
Software Design & Implementation

Spring 2023

Section 2 – Functional Programming I

Inductive Data Types – Review

- Describe a set by ways of creating its element
  - Each is a "constructor"

$$\textbf{type}\ \mathrm{T} := \mathrm{A}(\mathrm{x} : \mathbb{Z}) \mid \mathrm{B}(\mathrm{x} : \mathbb{Z},\ \mathrm{y} : \mathrm{T})$$

  - Second constructor is recursive
  - Can have any number of parameters

Ex:

base case

recursive case

$$\textbf{type}\ \mathrm{List} := \mathrm{nil} \mid \mathrm{cons}(\mathrm{x} : \mathbb{Z},\ \mathrm{L} : \mathrm{List})$$

- **Inductive definition of lists of integers**

nil	≈ []
cons(3, nil)	≈ [3]
cons(2, cons(3, nil))	≈ [2, 3]
cons(1, cons(2, cons(3, nil)))	≈ [1, 2, 3]

array notation

Structural Recursion – Review

- **Inductive types:** builds new values from existing ones
- **Structural recursion:** recurse on smaller parts
  - Call on n recurses on n.val
  - Guarantees no infinite loops
  - Note: Only kind of recursion used for this class

Ex: **type** List := nil | cons(hd: $\mathbb{Z}$, tl: List)

- **Mathematical definition of length**

func len(nil) := 0
len(cons(x, L)) := 1 + len(L) for any x ∈ $\mathbb{Z}$ and any L ∈ List

  - **any list is either nil or cons(x, L) for some x and L**
  - **so one of these two rules always applies**

Testing – Strict vs Deep

```
describe('example', function() {
    it('testBar' function() {
       .../* assert statements */
    })
})
```

Assertion	Failure Condition
assert.strictEqual(expected, actual)	expected !== actual
assert.deepEqual(expected, actual)	If properties of child objects are not equal

```
const v1 : Vector = {x: 1, y : 1}
const v2 : Vector = {x: 1, y: 1}
it('add_vector_deep', function() {
    assert.deepEqual(add_vector(v1, v2), {x: 2, y: 2})
})

it('add_vector_strict', function() {
    assert.strictEqual(add_vector(v1, v2), {x: 2, y: 2})
})
```

Testing – Documenting

- Make sure to document which subdomain/domain you are testing for each test

Ex:

Name of class being tested

```
describe('example', function() {
```

Name of function being tested

```
    it('testBar' function() {
       /* comment describing subdomain being tested */
       assert...
    })
})
```

Question 1

We are asked to write a function "twice" that takes a list as an argument and "returns a list of the same length but with every number in the list multiplied by 2".

(a) This is an English definition of the problem, so our first step is to formalize it. Let's start by writing this out in more detail. Fill in the blanks showing the result of applying twice to lists of different lengths.

nil	_____
cons(a, nil)	_____
cons(a, cons(b, nil))	_____
cons(a, cons(b, cons(c, nil)))	_____
...	

(b) The previous list of examples is not a formal definition. It does not tell us, for example, what twice does to a list of length 4. More generally, any time we see "...", the definition is probably not formal.

Write a formal definition of twice using recursion.

(c) If we translated this into TypeScript code, what tests (if any) should we include to make sure that we did it correctly?

Question 1

We are asked to write a function "twice" that takes a list as an argument and "returns a list of the same length but with every number in the list multiplied by 2".

(a) This is an English definition of the problem, so our first step is to formalize it. Let's start by writing this out in more detail. Fill in the blanks showing the result of applying twice to lists of different lengths.

nil	nil
cons(a, nil)	cons(2a, nil)
cons(a, cons(b, nil))	cons(2a, cons(2b,nil))
cons(a, cons(b, cons(c, nil)))	cons(2a, cons(2b,cons(2c, nil)))
...	

(b) The previous list of examples is not a formal definition. It does not tell us, for example, what twice does to a list of length 4. More generally, any time we see "...", the definition is probably not formal.

Write a formal definition of twice using recursion.

(c) If we translated this into TypeScript code, what tests (if any) should we include to make sure that we did it correctly?

Question 1

We are asked to write a function "twice" that takes a list as an argument and "returns a list of the same length but with every number in the list multiplied by 2".

(a) This is an English definition of the problem, so our first step is to formalize it. Let's start by writing this out in more detail. Fill in the blanks showing the result of applying twice to lists of different lengths.

nil	nil
cons(a, nil)	cons(2a, nil)
cons(a, cons(b, nil))	cons(2a, cons(2b,nil))
cons(a, cons(b, cons(c, nil)))	cons(2a, cons(2b,cons(2c, nil)))
...	

(b) The previous list of examples is not a formal definition. It does not tell us, for example, what twice does to a list of length 4. More generally, any time we see "...", the definition is probably not formal.

Write a formal definition of twice using recursion.

func twice(nil) := nil

twice(cons(a, L)) := cons(2a, twice(L)) for any a : Z and *L* : List

(c) If we translated this into TypeScript code, what tests (if any) should we include to make sure that we did it correctly?

Question 1

We are asked to write a function "twice" that takes a list as an argument and "returns a list of the same length but with every number in the list multiplied by 2".

(a) This is an English definition of the problem, so our first step is to formalize it. Let's start by writing this out in more detail. Fill in the blanks showing the result of applying twice to lists of different lengths.

nil	nil
cons(a, nil)	cons(2a, nil)
cons(a, cons(b, nil))	cons(2a, cons(2b,nil))
cons(a, cons(b, cons(c, nil)))	cons(2a, cons(2b,cons(2c, nil)))
...	

(b) The previous list of examples is not a formal definition. It does not tell us, for example, what twice does to a list of length 4. More generally, any time we see "...", the definition is probably not formal.

Write a formal definition of twice using recursion.

func twice(nil) := nil

twice(cons(a, L)) := cons(2a, twice(L)) for any a : Z and *L* : List

(c) If we translated this into TypeScript code, what tests (if any) should we include to make sure that we did it correctly?

We should apply 0-1-many heuristic (ex: test length of 0, 1, 3

Question 2

func twice(nil) := nil

twice(cons(a, L)) := cons($2a$, twice(L)) for any $a : \mathbb{Z}$ and L : List

(a) Using the fact that $L = \text{cons}(1, \text{cons}(2, \text{nil}))$, prove by calculation that twice(L) = R, i.e., that

$$\text{twice}(L) = \text{cons}(2, \text{cons}(4, \text{nil}))$$

Question 2

func twice(nil) := nil

twice(cons(a, L)) := cons($2a$, twice(L)) for any $a : \mathbb{Z}$ and L : List

(a) Using the fact that $L = \text{cons}(1, \text{cons}(2, \text{nil}))$, prove by calculation that twice(L) = R, i.e., that

$$\text{twice}(L) = \text{cons}(2, \text{cons}(4, \text{nil}))$$

twice(L)

= twice(cons(1, cons(2, nil))) Def of L

= cons(2, twice(cons(2, nil))) Def of twice

= cons(2, cons(4, twice(nil))) Def of twice

= cons(2, cons(4, nil)) Def of twice

Question 2

func twice(nil) := nil

twice(cons(a, L)) := cons($2a$, twice(L)) for any $a : \mathbb{Z}$ and L : List

twice(L) = cons(2, cons(4, nil))

(b) Using the facts that $L = \text{cons}(1, \text{cons}(2, \text{nil}))$ and $R = \text{cons}(2, \text{cons}(4, \text{nil}))$, prove by calculation that the code above returns the correct value, i.e., that

$$\text{twice}(\text{cons}(0, L)) = \text{cons}(0, R)$$

Feel free to cite part (a) in your calculation.

Question 2

func twice(nil) := nil

twice(cons(a, L)) := cons($2a$, twice(L)) for any $a : \mathbb{Z}$ and L : List

twice(L) = cons(2, cons(4, nil))

(b) Using the facts that L = cons(1, cons(2, nil)) and R = cons(2, cons(4, nil)), prove by calculation that the code above returns the correct value, i.e., that

$$\text{twice}(\text{cons}(0, L)) = \text{cons}(0, R)$$

Feel free to cite part (a) in your calculation.

twice(cons(0, L))

= cons(0, twice(L)) Def of twice

= cons(0, cons(2, cons(4, nil))) Part(a)

= cons(0, R) Def of R

Question 3

We are asked to write a function that takes a list as an argument and "returns a list of the same length but with *every other* number in the list, *starting with the first number*, multiplied by 2".

The first number in the list is at index 0, which is even; the second number in the list is at index 1, which is odd; the third number in the list is at index 2, which is even; and so on. Hence, we will call this function twice-evens because it multiples the numbers at even indexes by two and leaves those at odd indexes unchanged.

(a) The definition of the problem was in English, so our first step is to formalize it. Let's start by writing this out in more detail. Fill in the blanks showing the result of applying twice-even to lists of different lengths.

nil	_____
cons(a, nil)	_____
cons(a, cons(b, nil))	_____
cons(a, cons(b, cons(c, nil)))	_____
...	

Question 3

We are asked to write a function that takes a list as an argument and "returns a list of the same length but with *every other* number in the list, *starting with the first number*, multiplied by 2".

The first number in the list is at index 0, which is even; the second number in the list is at index 1, which is odd; the third number in the list is at index 2, which is even; and so on. Hence, we will call this function twice-evens because it multiples the numbers at even indexes by two and leaves those at odd indexes unchanged.

(a) The definition of the problem was in English, so our first step is to formalize it. Let's start by writing this out in more detail. Fill in the blanks showing the result of applying twice-even to lists of different lengths.

nil	nil
cons(a, nil)	cons(2a,nil)
cons(a, cons(b, nil))	cons(2a,cons(b,nil))
cons(a, cons(b, cons(c, nil)))	cons(2a,cons(b,cons(2c,nil)))
...	

Question 3

(b) The previous list of examples is not a formal definition (because of the "...").

Write a formal definition of this function, twice-evens, using recursion. In order to do so, you may need to define more than one function!

(c) If we translated this into TypeScript code, what tests (if any) should we include to make sure that we did it correctly?

Question 3

(b) The previous list of examples is not a formal definition (because of the "...").

Write a formal definition of this function, twice-evens, using recursion. In order to do so, you may need to define more than one function!

func twice-evens(nil) := nil
twice-evens(cons(a, L) := cons(2a, twice-odds(L)) for any a : Z and L : List

func twice-odds(nil) := nil
twice-odds(cons(a, L)) := cons(a, twice-evens(L)) for any a: Z and L : List

(c) If we translated this into TypeScript code, what tests (if any) should we include to make sure that we did it correctly?

Question 3

(b) The previous list of examples is not a formal definition (because of the "...").

Write a formal definition of this function, twice-evens, using recursion. In order to do so, you may need to define more than one function!

func twice-evens(nil) := nil
twice-evens(cons(a, L) := cons(2a, twice-odds(L)) for any a : Z and L : List

func twice-odds(nil) := nil
twice-odds(cons(a, L)) := cons(a, twice-evens(L)) for any a: Z and L : List

(c) If we translated this into TypeScript code, what tests (if any) should we include to make sure that we did it correctly?

For twice-evens, we need to test nil, a call to twice-odds base case, and a recursive call of twice-evens. This gives us 5 tests for twice-evens. We need to do the same for twice-odds, which gives a total of 10 tests.

Question 4

func twice-evens(nil) := nil

twice-evens(cons(a, L)) := cons($2a$, twice-odds(L)) for any $a : \mathbb{Z}$ and L : List

func twice-odds(nil) := nil

twice-odds(cons(a, L)) := cons(a, twice-evens(L)) for any $a : \mathbb{Z}$ and L : List

func len(nil) := 0

len(cons(a, L)) := $1 + \text{len}(L)$ for any $a : \mathbb{Z}$ and L : List

(a) Let a and b be any integers. Prove by calculation that

$$\text{len}(\text{twice-evens}(\text{cons}(a, \text{cons}(b, L)))) = 2 + \text{len}(\text{twice-evens}(L))$$

Question 4

func twice-evens(nil) := nil

twice-evens(cons(a, L)) := cons($2a$, twice-odds(L)) for any $a : \mathbb{Z}$ and L : List

func twice-odds(nil) := nil

twice-odds(cons(a, L)) := cons(a, twice-evens(L)) for any $a : \mathbb{Z}$ and L : List

func len(nil) := 0

len(cons(a, L)) := $1 +$ len(L) for any $a : \mathbb{Z}$ and L : List

(a) Let a and b be any integers. Prove by calculation that

$$\text{len}(\text{twice-evens}(\text{cons}(a, \text{cons}(b, L)))) = 2 + \text{len}(\text{twice-evens}(L))$$

len(twice-evens(cons(a, cons(b, L))))

= len(cons(2a, twice-odds(cons(b, L)))) Def of twice-evens

= len(cons(2a, cons(b, twice-evens(L)))) Def of twice-odds

= 1 + len(cons(b, twice-evens(L))) Def of len

= 1 + 1 + len(twice-evens(L)) Def of len

= 2 + len(twice-evens(L))

Question 4

Given this code:

```
return 2 + len(twice_evens(L));  // = len(twice-evens(cons(3, cons(4, L))))
```

And the fact we proved in (a):

$$\text{len}(\text{twice-evens}(\text{cons}(a, \text{cons}(b, L)))) = 2 + \text{len}(\text{twice-evens}(L))$$

(b) Explain why this shows that the code is correct.

Question 4

Given this code:

```
return 2 + len(twice_evens(L));  // = len(twice-evens(cons(3, cons(4, L))))
```

And the fact we proved in (a):

$$\text{len}(\text{twice-evens}(\text{cons}(a, \text{cons}(b, L)))) = 2 + \text{len}(\text{twice-evens}(L))$$

(b) Explain why this shows that the code is correct.

Applying part (a) with a = 3 and b=4 gives us a proof that:
len(twice-evens(cons(3, cons(4, L)))) = 2 + len(twice-evens(L))

Attendance

Please fill out the Google Form at the following link: