CSE 331
Software Design & Implementation

Autumn 2023

Section 3 – Functional Programming I

Administrivia

- HW3 released later tonight, due next Wednesday at **11pm**
  - Will be **more difficult** than HW1 and HW2. + has more weight in the gradebook
  - Please! start early and be prepared for a challenge!
  - Give yourself time to come to OH and ask questions on Ed
  - Working on the same issue for hours when you’re stuck won’t help, ask for help!

Review – Inductive Data Types

- Describe a set by ways of creating an element of the type
  - Each is a “constructor”
  - Second constructor is recursive
  - Can have any number of parameters

Ex: base case / recursive case

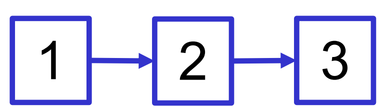
$$\textbf{type}\ \text{List} := \text{nil} \mid \text{cons}(x : \mathbb{Z},\ L : \text{List})$$

nil

cons(3, nil)

cons(2, cons(3, nil))

cons(1, cons(2, cons(3, nil)))



Review – Structural Recursion

- **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)

func len(nil) := 0

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

- Any List is either nil or of the form cons(x, L) for some number x and List L
- Cases of function are exclusive and exhaustive based on ⤴️

Testing

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

- Use assertions to compare expected and actual output for each test case
  - `assert.deepStrictEqual(expected, actual);` should be used generally
- Keep your tests simple! Don't want to have to write tests for your tests

Testing – Strict vs Deep

Assertion	Failure Condition
`assert.strictEqual(expected, actual)`	expected !== actual
`assert.deepStrictEqual(expected, actual)`	values/types of child objects are not equal

```
const v1: Vector = {x: 1, y: 1};
const v2: Vector = {x: 1, y: 1};
```

two different objects, but same record values

```
it('assert_strict', function() {
  assert.strictEqual(v1, v2);
});
```

this will fail

```
it('assert_deep_strict', function() {
  assert.deepStrictEqual(v1, v2);
});
```

this will pass

Testing – Documenting

- Document which subdomain you are testing. A justification: heuristic used, part of code it tests.

Name of class being tested

Ex:

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

Name of test (can be function being tested)

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

Definitions

$$\textbf{type}\ \mathsf{List} := \mathsf{nil} \mid \mathsf{cons}(\mathsf{hd} : \mathbb{Z},\ \mathsf{tl} : \mathsf{List})$$

- Len – returns the length of a list:

$$\begin{aligned} \textbf{func}\ \mathsf{len}(\mathsf{nil}) &:= 0 \\ \mathsf{len}(\mathsf{cons}(a, L)) &:= 1 + \mathsf{len}(L) \quad \text{for any } a : \mathbb{Z} \text{ and } L : \mathsf{List} \end{aligned}$$

- Sum – returns the sum of the integers in the list:

$$\begin{aligned} \textbf{func}\ \mathsf{sum}(\mathsf{nil}) &:= 0 \\ \mathsf{sum}(\mathsf{cons}(a, L)) &:= a + \mathsf{sum}(L) \quad \text{for any } a : \mathbb{Z} \text{ and } L : \mathsf{List} \end{aligned}$$

Question 1

twice takes a list and returns a list of the same length but with every number in the list multiplied by 2

Show the result of applying twice to each list:

nil	____________________
$\mathsf{cons}(a, \mathsf{nil})$	____________________
$\mathsf{cons}(a, \mathsf{cons}(b, \mathsf{nil}))$	____________________
$\mathsf{cons}(a, \mathsf{cons}(b, \mathsf{cons}(c, \mathsf{nil})))$	____________________

$\ldots$

Question 1

(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.

Question 1

$$\begin{aligned} \textbf{func}\ \text{twice}(\text{nil}) &:= \text{nil} \\ \text{twice}(\text{cons}(a, L)) &:= \text{cons}(2a, \text{twice}(L)) \quad \text{for any } a : \mathbb{Z} \text{ and } L : \text{List} \end{aligned}$$

(c) If we translated this into TypeScript code in the most direct manner (level 0), what heuristic should we use to get a set of subdomains? What specific tests should we use to make sure that everything is correct?

Question 2

```
if (equal(L, cons(1, cons(2, nil)))) {
  const R = cons(2, cons(4, nil));  // = twice(L)
  return cons(0, R);                // = twice(cons(0, L))
}
```

Comments // are the spec, but the code isn't a direct translation of the spec (level 1)

Need to prove it does the same thing as the spec

Question 2

(a) Using the fact that $L = \text{cons}(1, \text{cons}(2, \text{nil}))$, prove by calculation that $\text{twice}(L) = R$, where R is the constant list defined in the code. I.e., prove that

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

$$\begin{aligned} &\textbf{func } \text{twice}(\text{nil}) && := \text{nil} \\ &\quad\;\;\, \text{twice}(\text{cons}(a, L)) && := \text{cons}(2a, \text{twice}(L)) \quad \text{for any } a : \mathbb{Z} \text{ and } L : \text{List} \end{aligned}$$

Question 2

$$\begin{aligned} \textbf{func}\ \mathsf{twice}(\mathsf{nil}) &:= \mathsf{nil} \\ \mathsf{twice}(\mathsf{cons}(a, L)) &:= \mathsf{cons}(2a, \mathsf{twice}(L)) \quad \text{for any } a : \mathbb{Z} \text{ and } L : \mathsf{List} \end{aligned}$$

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

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

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

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

Question 3

twice-evens takes a list and returns a list of the same length but with *every other* number (at even indices) in the list multiplied by 2

Show the result of applying twice-evens to each list:

nil ____________________

cons(a, nil) ____________________

cons(a, cons(b, nil)) ____________________

cons(a, cons(b, cons(c, nil))) ____________________

$\ldots$

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!

Question 3

(c) If we translated this into TypeScript code in the most direct manner (level 0), what tests (if any) should we include to make sure that everything is correct?

Question 4

$$\begin{aligned}
&\textbf{func}\ \text{twice-evens}(\text{nil}) &&:= \text{nil} \\
&\quad \text{twice-evens}(\text{cons}(a, L)) &&:= \text{cons}(2a, \text{twice-odds}(L)) \quad \text{for any } a : \mathbb{Z} \text{ and } L : \text{List}
\end{aligned}$$

$$\begin{aligned}
&\textbf{func}\ \text{twice-odds}(\text{nil}) &&:= \text{nil} \\
&\quad \text{twice-odds}(\text{cons}(a, L)) &&:= \text{cons}(a, \text{twice-evens}(L)) \quad \text{for any } a : \mathbb{Z} \text{ and } L : \text{List}
\end{aligned}$$

$$\begin{aligned}
&\textbf{func}\ \text{len}(\text{nil}) &&:= 0 \\
&\quad \text{len}(\text{cons}(a, L)) &&:= 1 + \text{len}(L) \quad \text{for any } a : \mathbb{Z} \text{ and } L : \text{List}
\end{aligned}$$

(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

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 the direct proof from part (a) shows that the code is correct according to the specification (written in the comment).

HW3 Reminders/Recommendations:

- **No mutation!** Make a new variable for new values you calculate
- Proofs by calculation require explanations/rules for **every** line (except basic algebra, you can say “math” if you want)
- Proofs by calculation can start with the left or right side of the = to prove
- We won’t penalize you for more test cases than the minimum required!
- If you get errors that “`property ___ does not exist on type ___`" it probably means you are missing a `nil` check

Proof by calculation LaTeX

- *Optional*, if you're using LaTeX feel free to use this to align proofs:

```
$$\begin{aligned}
      & first line of proof          && \\
      &\qquad = line of proof        && \text{rule} \\
      &\qquad = line of proof        && \text{rule} \\
\end{aligned}$$
```

- and to align functions:

```
$$\begin{aligned}
  \textbf{func } & \textsf{funcName}(case)
                         &&:= result &\text{side cond} \\
                 & \textsf{funcName}(case)
                         &&:= result &\text{side cond} \\
\end{aligned}$$
```