CSE 331
Software Design & Implementation

Spring 2023

Section 3 – Functional Programming II

Administrivia

- HW3 released later today
  - Due Wednesday (4/19) @ 11:00pm
- Google Form for Student GitLab Repo (here) – Optional
  - If you would like to have a dedicated student repo for this course to maintain version control

Structural Induction – Review

- Let P(S) be the claim
- To Prove P(S) holds for any list S, we need to prove two implications:
  - **Base Case:** prove P(nil)
    - Use any know facts and definitions
  - **Inductive Step:** prove P(cons(x, L)) for any x : *Z,* L : List
    - Direct proof
    - Use know facts and definition and Inductive Hypothesis
  - **Inductive Hypothesis:** assume P(L) is true
    - Use this in the inductive step ONLY
- Assuming we know P(S), if we prove P(cons(x, L)), we then prove recursively that P(S) holds for any List

Defining Function By Cases – Review

- Sometimes we want to define functions with other cases
  - E.g. define f(n) where n : *Z*

$$
\begin{aligned}
&\textbf{func}\ \ \mathrm{f(n)} := \mathrm{2n} + 1 && \textbf{if}\ \mathrm{n} \geq 0 \\
&\phantom{\textbf{func}}\ \ \mathrm{f(n)} := 0 && \textbf{if}\ \mathrm{n} < 0
\end{aligned}
$$

  - To use the definition f(m), we need to know if m > 0 or not
  - Because of this structure, the proof needs to look different

Proof By Cases – Review

- New code structure means we need new proof structures
- Can split a proof into cases:
  - E.g. a = True and a = False
  - E.g. n >= 0 and n < 0
  - These cases needs to be exhaustive
- Ex:

$$\textbf{func } f(n) := 2n + 1 \qquad \textbf{if } n \geq 0$$
$$f(n) := 0 \qquad \textbf{if } n < 0$$

Prove that $f(n) \geq n$ **for any** $n : \mathbb{Z}$

Case $n \geq 0$:

$$f(n) = \ldots \geq n$$

Since these 2 cases are exhaustive, f(n) >= n holds in general

Case $n < 0$:

$$f(n) = 0 \qquad \textbf{def of } f \textbf{ (since } n < 0\textbf{)}$$
$$> n \qquad \textbf{since } n < 0$$

Question 1

We are asked to write a function pseudo-sort that takes a list as an argument, “looks at the first two numbers, moves the smaller of those to the front, and then continues on the rest of the list after the first element“.

(a) Write a formal definition for this English definition?

(b) Show by example that pseudo-sort does not actually sort the list.

Question 1

We are asked to write a function pseudo-sort that takes a list as an argument, "looks at the first two numbers, moves the smaller of those to the front, and then continues on the rest of the list after the first element".

(a) Write a formal definition for this English definition?

func pseudo-sort(nil)	:= nil	
pseudo-sort(cons(a, nil))	:= cons(a, nil)	for any a : *Z*
pseudo-sort(cons(a, cons(b, L)))	:= cons(a, pseudo-sort(cons(b, L)))	B
pseudo-sort(cons(a, cons(b, L)))	:= cons(b, pseudo-sort(cons(a, L)))	C

Where B is "for any a, b: *Z* and *L*: List with a <= b" and C is "for any a, b: *Z* and *L*: List with a > b"

(b) Show by example that pseudo-sort does not actually sort the list.

Question 1

We are asked to write a function pseudo-sort that takes a list as an argument, "looks at the first two numbers, moves the smaller of those to the front, and then continues on the rest of the list after the first element".

(a) Write a formal definition for this English definition?

func pseudo-sort(nil)	:= nil	
pseudo-sort(cons(a, nil))	:= cons(a, nil)	for any a : *Z*
pseudo-sort(cons(a, cons(b, L)))	:= cons(a, pseudo-sort(cons(b, L)))	B
pseudo-sort(cons(a, cons(b, L)))	:= cons(b, pseudo-sort(cons(a, L)))	C

Where B is "for any a, b: *Z* and *L*: List with a <= b" and C is "for any a, b: *Z* and *L*: List with a > b"

(b) Show by example that pseudo-sort does not actually sort the list.

We can see that:

func pseudo-sort(cons(2, cons(3, cons(1, nil))))	
= cons(2, pseudo-sort(cons(3, cons(1, nil))))	Def of pseudo-sort
= cons(2, cons(1, pseudo-sort(cons(3, nil))))	Def of pseudo-sort
= cons(2, cons(1, cons(3, nil)))	Def of pseudo-sort

However, the sorted list is cons(1, cons(2, cons(3, nil)))

Question 2

You see following snippet in some TypeScript code:

```
const s = sum(L);
...
return 2 * s;  // = sum(twice(L))
```

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

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

This code claims to calculate the answer $\mathsf{sum}(\mathsf{twice}(L))$, but it actually returns $2\,\mathsf{sum}(L)$. Prove this code is correct by showing that $\mathsf{sum}(\mathsf{twice}(L)) = 2\,\mathsf{sum}(L)$ holds for any list L by structural induction.

Question 2

You see following snippet in some TypeScript code:

```
const s = sum(L);
...
return 2 * s;   // = sum(twice(L))
```

func $\text{sum}(\text{nil}) := 0$

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

func $\text{twice}(\text{nil}) := \text{nil}$

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

This code claims to calculate the answer $\text{sum}(\text{twice}(L))$, but it actually returns $2\,\text{sum}(L)$. Prove this code is correct by showing that $\text{sum}(\text{twice}(L)) = 2\,\text{sum}(L)$ holds for any list L by structural induction.

(1) Define P(L) to be claim that sum(twice(L)) = 2sum(L). We will prove the claim by structural induction

(2) Base Case (nil):

sum(twice(nil))

= sum(nill)	Def of twice
= 0 = 2*0	Algebra
= 2 * sum(nil)	Def of sum

(3) Inductive Hypothesis. Suppose that P(L) holds for a list L. (i.e. suppose that sum(twice(L)) = 2sum(L)

Question 2 continued…

You see following snippet in some TypeScript code:

```
const s = sum(L);
...
return 2 * s;  // = sum(twice(L))
```

func $\mathsf{sum}(\mathsf{nil}) := 0$

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

func $\mathsf{twice}(\mathsf{nil}) := \mathsf{nil}$

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

This code claims to calculate the answer $\mathsf{sum}(\mathsf{twice}(L))$, but it actually returns $2\,\mathsf{sum}(L)$. Prove this code is correct by showing that $\mathsf{sum}(\mathsf{twice}(L)) = 2\,\mathsf{sum}(L)$ holds for any list L by structural induction.

(4) Inductive Step. Show P(cons(a, L)) for any integer *a*

Let *a* be any integer. Then we can calculate,

sum(twice(cons(a, L))

= sum(cons(2a, twice(L)))	Def of twice
= 2a + sum(twice(L))	Def of sum
= 2a + 2sum(L)	I.H.
= 2(a + sum(L))	
= 2sum(cons(a, L))	Def of sum

(5) Conclusion. P(L) holds for any L by structural induction

Question 3

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

$$\text{sum}(\text{twice-evens}(L)) + \text{sum}(\text{twice-odds}(L)) = 3\,\text{sum}(L)$$

Use structural induction to prove that this holds for any list L.

Question 3

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

$$\text{sum}(\text{twice-evens}(L)) + \text{sum}(\text{twice-odds}(L)) = 3\,\text{sum}(L)$$

Use structural induction to prove that this holds for any list L.

(1) Let P(L) be the claim above. We will prove this claim by structural induction

(2) Base Case (nil)

sum(twice-evens(nil)) + sum(twice-odds(nil)) = 3sum(nil)

= sum(nil) + sum(twice-odds(nil))	Def of twice-evens
= sum(nil) + sum(nil)	Def of twice-odds
= 0 = 3 * 0	Algebra
= 3sum(nil)	Def of sum

(3) Inductive Hypothesis. Suppose P(L) holds for a list *L*

Question 3 continued…

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

(4) Inductive Step. Show P(cons(a, L)) for any integer a

Let a be any integer. Then we can calculate

sum(twice-evens(cons(a, L))) + sum(twice-odds(cons(a, L)))

= sum(cons(2a, twice-odds(L)))
+ sum(twice-odds(cons(a, L))) — Def of twice-evens

= 2a + sum(twice-odds(L))
+ sum(twice-odds(cons(a, L))) — Def of sum

= 2a + sum(twice-odds(L))
+ sum(cons(a, twice-evens(L))) — Def of twice-odds

= 2a + sum(twice-odds(L))
+ a + sum(twice-evens(L)) — Def of sum

= 3a + sum(twice-odds(L)) + sum(twice-evens))

= 3a + 3sum(L) — By I.H.

= 3(a + sum(L))

= 3sum(cons(a, L)) — Def of sum

(5) Conclusion. P(L) holds for any list *L* by structural induction

Question 4

func swap(nil)	:=	nil	
swap(cons$(a$, nil))	:=	cons$(a$, nil)	for any $a : \mathbb{Z}$
swap(cons$(a$, cons$(b, L)))$	:=	cons$(b$, cons$(a$, swap$(L)))$	for any $a, b : \mathbb{Z}$ and L : List

Prove by cases that swap(cons$(a, L)) \neq$ nil for any integer a and list L.

Question 4

func swap(nil)	:=	nil	
swap(cons(a, nil))	:=	cons(a, nil)	for any $a : \mathbb{Z}$
swap(cons(a, cons(b, L)))	:=	cons(b, cons(a, swap(L)))	for any $a, b : \mathbb{Z}$ and L : List

Prove by cases that swap(cons(a, L)) $\neq$ nil for any integer a and list L.

Let a be any integer and *L* be any list. We argue by cases on *L*

First, suppose that *L* = nil

swap(cons(a, L))

= swap(cons(a, nil))

= cons(a, nil)   Def of swap

≠ nil

Next, suppose that *L* ≠ nil. That means *L* = cons(b, R) for some b: *Z* and R: List

swap(cons(a, L))

= swap(cons(a, cons(b, R)))

= cons(b, cons(a, swap(R)))   Def of swap

≠ nil

Attendance

Please fill out the Google Form at the following link: