CSE 331: Software Design & Implementation

Section Cipher

The problems that follow make use of the following inductive type, representing lists of integers

$$\textbf{type}\ \text{List} := \quad \text{nil} \quad | \quad \text{cons}(\text{hd} : \mathbb{Z},\ \text{tl} : \text{List})$$

We will also use the function sum, which returns the sum of the integers in the list:

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

the function twice, which doubles each number in the list:

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

the functions twice-evens and twice-odds, which double the integers at even and odd indexes in the list:

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

the function swap, which swaps adjacent integers in the list:

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

and the function len, which finds the length of the list:

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

1. Any Sort in a Storm

We are asked to write a function pseudo-sort that takes a list of integers as an argument, "looks at the first two integers in the list, orders the pair to place the smaller of the two in the front, and then continues through the following pairs in the list after the first element".

For example: if the first two elements in a list are 5, 1 they would be reordered so the 1 comes first: 1, 5, and the next pair to look at would have the 5 as the first element.

(a) This is an English definition of the problem, so our first step is to formalize it.

Write a formal definition of pseudo-sort using recursion.

Note that, in order to do this, you will need to include include extra conditions on the patterns used in the definition. You can write extra conditions in the third column (where we usually write, e.g., "for any $a : \mathbb{Z}$ ...") by writing "with" or "if" and then the condition.

Also, if the conditions do not fit on one line, you can replace them with a label, e.g., "A", and then define that label on another line by writing "where A is ...". (Likewise, if any expression is too long, you can replace part of the expression by a new variable, e.g., "n", and then define that variable on another line by writing "where $n := \ldots$".)

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

Note that, when you are doing a calculation and applying a definition that has an extra condition, you should point out *why* that condition holds in your explanation in the right column.

2. Here Comes the Sum

You see following snippet in some TypeScript code:

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

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 $\mathsf{sum}(\mathsf{twice}(S)) = 2\,\mathsf{sum}(S)$ holds for any list S by structural induction.

3. Can You Sum a Few Bars?

In HW Quilt, you proved the following holds when L has length 3:

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

Prove that this holds for any list S by structural induction.

4. Swapaholic

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

5. Max Season

We are asked to write a couple functions that are different operations over a pair of lists. As in problem 1, you will need to use side conditions for these functions and if the conditions become long you can replace them with a label and define that label below.

(a) First write a function max-sums that takes two lists of integers as arguments, "compares the sums of the elements in each list and returns the maximum of the sums". This function will only compute the max if the lists are the same length, otherwise it will return undefined.

For example: if the given lists are L $=$ cons$(1,$ cons$(3,$ cons$(5,$ nil$)))$ and R $=$ cons$(2,$ cons$(4,$ cons$(6,$ nil$)))$, it will return 12, the sum of R which is larger than the sum of L. If L $=$ cons$(1,$ cons$(3,$ cons$(5,$ nil$)))$ and R $=$ cons$(10,$ nil$)$ it would return undefined, since the two lists are not the same length.

Write a formal definition of max-sums *without* using recursion. You will want your side conditions and outputs to include the result of calling other mathematical definitions.

(b) Now, we want to write a function, abs-difs that takes two lists of integers, "looks at the first number of each list and places the absolute difference of the two items in a result list unless the difference is 0, and then continues through the rest of the list." Again, this function will only compute the differences if the lists are the same length, otherwise it will return undefined.

For example: if the given lists are L $=$ cons$(-1,$ cons$(5,$ cons$(8,$ nil$)))$ and R $=$ cons$(2,$ cons$(4,$ cons$(8,$ nil$)))$, it will return the list cons$(3,$ cons$(1,$ nil$))$. If L $=$ cons$(-1,$ cons$(5,$ cons$(8,$ nil$)))$ and R $=$ cons$(-3,$ nil$)$ it would return undefined, since the two lists are not the same length.

Write a formal definition of abs-difs *using* recursion. Like problem 1, this is a problem where you can use pattern matching, but will need extra conditions on those patterns.