Lab 03

15-150 Fall 2024

Due: Thu 12th Sept, 2024 at 5:00PM

Interview

This assignment will be evaluated in an interview with the course staff on the date above. The points in the tasks are meaningless and submission of solutions is not required.

The instructions for the interview are in the interview guide.

1 List permutations

One useful notion when you have two lists is comparing if one list is a *permutation* of the other list. That is, if the two lists both contain the same elements, possibly in different order. In this section, we will present an algorithm for checking if this property holds, and we will implement it in two different ways.

1.1 Using Destructors

To check if some list l_1 is a permutation of l_2, it suffices to show that each element of l_1 corresponds to an element of l_2 without any leftovers. This means that we need to have a function that checks that an element is in a list, a second function that removes it from that list, and a third function that combines them to determine if a list is a permutation of another.

Let's do it!

Coding Task 1.1 (2 points) Implement the SML function

```
member: int * int list -> bool
```

such that `member (x, l)` returns `true` if the number `x` occurs in list `l`, and `false` otherwise.

Coding Task 1.2 (2 points) Implement the SML function

```
remove: int * int list -> int list
```

such that, given a list `l` and an integer `x` that occurs in `l`, the call `remove (x, l)` returns the list `l'` obtained by removing the first occurrence of `x` from `l`.

Coding Task 1.3 (4 points) Implement the SML function

```
isPermutation: int list * int list -> bool
```

such that `isPermutation (l1, l2)` returns `true` if `l1` is a permutation of `l2`, and `false` otherwise.

2 The Flat List Society

The recently-founded Flat List Society has been closely following your progress in 15-150. In the previous homework, they (like you) saw the `int list` type, and saw proofs of functions that take `int lists` as input and produce `int lists` as output. However, they have recently learned that...

The same techniques that we have been using to program with `int lists` can be used for `lists` containing other types of elements - in particular, the elements themselves can be `int lists`! Values like this have the type `(int list) list`, which can also be written `int list list`, representing lists of lists of integers. For instance:

```
          [1]  ::  [] ≅ [[1]]                    : int  list  list
[1,2]  ::  [[3],[4,5]] ≅ [[1,2],[3],[4,5]]       : int  list  list
```

After having learned about this, the self-respecting members of the Flat List Society have become extremely offended - they only like 'flat' `lists`, that is, `lists` that do not have `lists` as elements. They have since threatened to use a `list` of weapons on the course! Help us appease them by writing a function that can flatten our `lists`!

2.1 We live in a (flat) society

Before we begin this heroic task, we will need to consider what it means for a list to be "flattened". The Flat List Society points you in the direction of an ancient tome that contains a function known to flatten lists.

Theorem 1. *A list `F : int list` is flattened with respect to another list `LL : int list list` if `oldFlat LL = F` where `oldFlat` is defined as follows:*

```
fun oldFlat [] = []
  | oldFlat (L::LS) = L @ (oldFlat LS)
```

Coding Task 2.1 (3 points)
Unfortunately, in today's anti-`@` world, this function has been deemed illegal. Armed with this ancient function as our definition, we must provide a new function allowing the Flat List Society to flatten their `int list lists`!

In `flatten.sml`, write the function `flatten (LL : int list list) -> int list`

You may not use or create any helper functions for this task. This includes `@`, `length` and any functions from the standard library.

Here are some example outputs:

```
flatten [] = []
flatten [[]] = []
flatten [[15, 150]] = [15, 150]
flatten [[15150]] = [15150]
flatten [[], [15, 15], [], [], [], [0]] = [15, 15, 0]
flatten [[1, 5], [1], [], [5, 0]] = [1, 5, 1, 5, 0]
```

2.2 We can actually prove it's flat (unlike some other societies)!

However, it is not enough to have written `flatten`. The Flat List Society has aggressively demanded that you prove to them that your function actually produces flattened lists! They will only accept your function if it behaves the same as their sacred function from an era long past. Help us fend off the impending attack!

Task 2.2 (10 points) Using your implementation, prove the following:

Theorem 2. *For all values `LL : int list list`, `flatten LL` $\cong$ `oldFlat LL`.*

You may assume that `@` and `oldFlat` have the following implementations:

```
fun []      @ L = L
  | (x::xs) @ L = x::(xs @ L)

  fun oldFlat [] = []
    | oldFlat (L::LS) = L @ (oldFlat LS)
```

Lemma 1. *You may cite that `oldFlat` is total.*

Hint: Think about the structure of the code when structuring your proof. Your proof should mirror the code.