Lab 05

15-150 Fall 2024

Due: Thu 3rd Oct, 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 Warmup: Types in SML

Do these tasks without the help of the SML interpreter.

Task 1.1 (6 points) For each of the following expressions, determine its type, if it has one. If the expression does not typecheck, answer “DNT”.

1. `[]`
2. `[[]]`
3. `NONE`
4. `if false then 1.0 else 1`
5. `fun w x = if h x then f x else g (f x)`
6. `fun f (x,y) = if x=1 then y else "asdf"`

Task 1.2 (4 points) Determine the value and type of each of the following expressions. Also, determine the values of the variables at the end of each line.

```
fun pet u = let
  fun fst (a, _) = a
  fun snd (_, a) = a
  fun cat v = fst (u, v)
  fun dog w = snd (u, w)
in
  (cat, dog)
end

let
  val x = 0
  val y = 7
in
  (let
     val x = 5
     val y = nil
   in
     x::y
   end) @ [x - y]
end
```

2 Polymorphic Trees

For these tasks, we will use the following datatype:

```
datatype 'a tree = Empty
                 | Node of 'a tree * 'a * 'a tree
```

You can find the code for the function `inorder: 'a tree -> 'a list` in the starter file `trees.sml`. It carries out an in-order traversal of its argument.

Coding Task 2.1 (4 points) Write a function

`merge: 'a tree * 'a tree -> 'a tree`

such that the call `merge(t1,t2)` returns a tree `t` so that `(inorder t1) @ (inorder t2)` is equivalent to `(inorder t)`.

Task 2.2 (10 points) Prove that

For all `t,t': 'a tree`*,*

`inorder(merge(t,t'))` $\cong$ `(inorder t) @ (inorder t')`

The polymorphic type variable `'a` can be instantiated to any type. Equality, however, is not meaningful on all types (you can't use `=` on expressions of type `real` for example — we will soon encounter other types that do not admit equality). However, in SML we can write

```
''a
```

for a type variable that can only be instantiated to types that admit equality (so that replacing it with `real` is disallowed). While it slightly restricts polymorphism, this gives us a way to write a lot of useful functions that, directly or indirectly, rely on equality. Here are some.

Coding Task 2.3 (2 points) Write the function

`member: ''a * ''a list -> bool`

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

Coding Task 2.4 (2 points) Write the function

`indexOf: ''a * ''a list -> int option`

defined so that `indexOf(x,l)` returns `NONE` if `x` is not in `l`, and otherwise returns `SOME n` where `n` is the index of the first occurrence of `x` in `l`.

Coding Task 2.5 (2 points) Write a function

`remove: ''a * ''a list -> ''a list`

such that `remove(x,l)` returns the list obtained by removing **all** occurrences of `x` from `l`.