15-150 Assignment 06

Jack Kasbeer
jkasbeer@andrew.cmu.edu
Section K
October 13, 2015

2: Polymorphic Universe

1. ML runtime system says...
   (i) `val map = fn : ('a -> 'b) -> 'a list -> 'b list`

   (ii) `val map1 = fn : 'a list -> ('a -> 'b) -> 'b list`

   (iii) `val map2 = fn : ('a -> 'b) list -> 'a -> 'b list`

   (iv) `val map3 = fn : ('a -> 'b) * 'a list -> 'b list`

   (v) `map4... Error: operator and operand don't agree... in expression: map4 f`

2. `map3` has type `('a -> 'b) * 'a list -> 'b list`, so the expression `(f x) :: map3 (f, L)` has type `'b list` since (1) `(f x)` has type `'b` (`f` has type `'a -> 'b` and `x` has type `'a`), and (2) `map3 (f, L)` has type `'b list` since `f : 'a -> 'b` and `L : 'a list`, so `map 3`'s type requirements are satisfied by this tuple. Hence, `(f x) :: map3 (f, L)` has type `'b list` (`(f x)` has type `'b` and `map3 (f, L)` has type `'b list`, by definition).

   `map4` has the same type as `map3`, but it doesn't evaluate on run-time since the expression `map4 f L` is not well-typed. `map4` expects a tuple of type `('a -> 'b) * 'a list`, but in this expression, `f L` has type `('a -> 'b) -> 'b list`, which doesn't fit the type of `map4` so a run-time error occurs for an operator and operand mismatch.

3. For each expression, indicate: (1) Well-typed? (2) If so, type? (3) Summary of behavior

   (i) `map (fn x => x ^ "!") ["Functions ", "are ", "Values"]` is well typed, with type

   `(string -> string) -> (string list -> string list)`, and is an expression that calls `map` on `(fn x => x ^ "!") ["Functions ", "are ", "Values"]`, which evaluates to a `string list` containing the elements of the input string list concatenated with "!"
   (`["Functions !", "are !", "Values!"]`).

   (ii) `map (fn (x,y) => (x-1, y ^ y)) [(1, "f"), ("o", 2), ("o", 3)]` is not well

   typed. This is due to the input list's inconsistency of elements; the function `(fn (x,y) => (x-1, y ^ y))` applies the `int` operation `-` to the first item in the tuple, and the string operation `^` to the second item in the input tuple, but the input list in the expression, `[(1, "f"), ("o", 2), ("o", 3)]`, has strings in the first item position and integers in the second item position in some of its elements. Hence, an operator and operand mismatch occurs when trying to evaluate this expression.

3: High Order Warp Jump

1. Prove: (LEMMA 1): For all types `t1` and `t2`, total funtion `f : t1 * t2 -> t2`, and values `z : t2` and `L : t1 list`,

   `foldr f z (L@[y]) = foldr f (f(y,z)) L`

   Proof. Let $P(n)$ be the proposition stated above, where n is the length of `L`. We will prove $\forall n \geq 0. P(n)$ by strong induction on n.

   Base Case $n = 0$: `length L = 0` $\Rightarrow P(0):$ `foldr f z ([] @ [y]) = foldr f (f(y,z)) L`.
   Evaluating the left side...
   `foldr f z ([] @ [y]) => foldr f z [y] = foldr f z (x::[y]) = f(y, foldr f z [])`

   Evaluating the right side...
   `foldr f z [] => f(y,z)`
   `foldr f (f(y,z)) [] => f(y,z)`

   Thus, when $n = 0$, `foldr f z (L@[y]) = f(y,z) = foldr f (f(y,z)) L`.
   Hence, $P(0)$ holds.

   Inductive Case $n > 0$: We have `length L > 0`. Assume that $P(1), ..., P(k-1)$ hold for some integer k.
   Inductive Hypothesis: $P(k)$ WWTS that $P(k)$ holds, given that $P(1), ..., P(k-1)$ hold.

   Now consider `length L = k`.
   Then, `foldr f z (`L_k`@[y]) => foldr f z ((x::`L_{k-1}`)@[y])`, and by Lemma a,
   this evaluates to `foldr f z (x::(`L_{k-1}`@[y])) = f(x, foldr f z (`L_{k-1}`@[y]))`
   By our IH, `foldr f z (`L_{k-1}`@[y])) = foldr f (f(y,z))` L_{k-1}
   $\therefore$ `f(x, foldr f z (`L_{k-1}`@[y])) = f(x, foldr f (f(y,z))` L_{k-1}

   Thus, $P(k)$ has been shown to be true, so by SPMI, it follows that $\forall n \geq 0. P(n)$.
   Hence, (LEMMA 1) has been proven. □

2. Prove: For all types `t` and values `L : t list list`,

   `flattenleft L = flattenright (rev L)`

   By definition, `flattenleft L = foldl (fn (x,A) => A@x) [] L`, and
   `flattenright L = foldr (fn (x,A) => A@x) [] L`. To prove the above proposition, it is sufficient to show that
   `(foldl (fn (x,A) => A@x) z L) = (foldr (fn (x,A) => A@x) z (rev L))` since
   `flattenleft L = flattenright (rev L)` is an instance of the above equality where
   `z = []`.

   Proof. Let $P(n)$ be the strengthened proposition stated above, where n is the length of `L`. We will prove $\forall n \geq 0. P(n)$ by strong induction on n. Also let `g` denote `fn (x,A) => A@x`.

Base Case $n = 0$: `length L = 0`
$\Rightarrow P(0):$ `flattenleft [] = foldl g [] [] => []` and `flattenright [] = foldr g [] [] => []`.
So $P(0)$ holds trivially.
Base Case $n = 1$: `length L = 1`
$\Rightarrow P(1):$ For some singleton `[x]` (`length [x] = 1`),
`flattenleft [x] = foldl g [] [x]` and
`foldl g [] [x] => foldl g (f(x,[])) [] => foldl g [x] [] = [x]`.
Similarly, `flattenright [x] = foldr g [] [x]` and
`foldr g [] [x] => f(x, foldr g [] []) => f(x, []) = [x]`.
Hence, $P(1)$ also holds.
Inductive Case $n > 1$: We have `length L > 1`. Assume that $P(1), ..., P(k-1)$ hold for some integer k.
Inductive Hypothesis $P(k)$: WWTS that $P(k)$ holds, given that $P(1), ..., P(k-1)$ hold.

Now consider `length L = k`, and let `L = x::(R@[y])` where `x, R, [y]` are subsets of `L`.
Then, `foldl g z (x::(R@[y])) = foldl g (g(x,z)) (R@[y]) => foldl g (z@x) (R@[y])` (Lemma c). Notice that `length (R@[y]) < k`.
This means `foldl g (z@x) (R@[y]) => foldr g (g(x,z)) (rev (R@[y]))` (Lemma c and IH).
By Lemma 1, this then evaluates to

`foldr g z (rev (R@[y]) @ [x]) => foldr g z rev(x::(R@[y]))` (Lemma b)

Notice that `L = x::(R@[y])`, so we have `foldr g z (rev L)`, and we're done.
Thus, $P(k)$ has been shown to be true, so by SPMI, it follows that $\forall n \geq 0. P(n)$.
Hence, the proposition has been proven. □

3. `costleft, costright` are defined in `hw06.sml`
4. Through testing `flattenleft L` and `flattenright L` with various lengths, it can be shown that both `flattenleft` and `flattenright` are $O(n^3)$.

4: Short Circuitry

1. Prove or disprove: (THM 1): For all types `t1, t2` and values `f : t1 -> bool`, `g : t2 -> bool`, and `x : t1`, `y : t2`,

   `(fn (a,b) => a orelse b) (f x, g y) = f x orelse g y`

   Proof. Suppose that `f` is total and `g` is not (i.e. it may fail to terminate) and assume that `f(x) => true`. Since `orelse` uses short circuit evaluation, the RHS of the proposition will evaluate to `true`, but the LHS will fail to terminate if `y` is such that `g(y)` fails to terminate (we know that such a `y` exists since `g` is assumed to not be a total function). □

5: Big Data

1. `merge_histogram` is defined in `hw06.sml`
2. `create_histogram` is defined in `hw06.sml`
3. `corpus_histogram` is defined in `hw06.sml`

6: The Real GridWorld

1. `path` is defined in `hw06.sml`