

CptS 355

Homework v. 2.3

19th October 2005

This homework is due Friday, October 21, **at the start of class**.

Changes 2.3: corrected 'define' to 'fun' in problem 3. Added needed initial value parameter to the use of fold in problem 3. Note that there is no 'fold' function – you have to choose foldl or foldr. Added hint for problem 1.

Changes 2.2: corrected : to :: in second clause of function g in question 6.

Changes 2.1: corrected a syntactic error in the function g in question 6. Added scoping question 7.

1. We will discuss Horner's rule for polynomial evaluation in class. Define a recursive function in Standard ML that takes a list of coefficients $[a_n, a_{n-1}, \dots, a_1, a_0]$ and a value, x , and computes its result, $\sum_{i=0}^n a_i x^i$, using Horner's rule. Make sure you understand what the right answers are by computing some examples by hand before writing your code. The form of your definition should be

```
fun evalpoly coeffs x = ...
```

Notice that the coefficients are in the list in reverse order – the coefficient of x^n is first.

Hint: You might find it easier to solve this first for a list in the form $[a_0, a_1, \dots, a_{n-1}, a_n]$, but ultimately you need to solve it in the form given. Think about using an accumulating parameter.

2. Evalpoly can be *partially applied* to a list of coefficients to yield a function that can be applied to different values of x . What is the type of the function returned by the partial application? Why is partial application useful in programming?
3. In languages that support higher-order functions, such as ML, the function *fold* (sometimes called *reduce*) can often be used to eliminate the need to write recursion explicitly. In ML the *fold* functions are named *foldl* and *foldr* and implement left- and right-associative folding respectively. A *fold* function takes 3 arguments (speaking loosely): a function taking a pair as its argument, a base value, and a list. Informally, fold inserts its function argument “between” each pair of elements of the list with the base value at the left of the sequence. For example, $\sum_{i=1}^5 i$ might be written $(foldl (op+) 0' [1, 2, 3, 4, 5])$ which would be evaluated as $(((((0+1)+2)+3)+4)+5)$. Define a function, *polystep*, that can be passed to a *fold* to accomplish the same effect as the function *evalpoly* in the previous example. Polystep will use take advantage partial evaluation. So the definition of *polystep* will look like

```
fun polystep x (a,b) = ...
```

and it will be used like

```
fold (polystep 3.4) 0.0 [1.0, 2.0, 3.0]
```

Hint: It is often helpful to think about the type of the function first. Does it matter whether *foldl* or *foldr* is used to evaluate polynomials using *polystep*?

4. What is the weakest precondition of each of the following sequences of assignment statements followed by postconditions?

- a) $a = 3*b + 4; b = a-4 \{b < 0\}$
- b) $a = n*a ; b = a+b \{b > 10\}$

5. Consider the following program and postcondition:

```
i = k+1;
product = 1;
while (i<=n) do
  product = product * i;
  i = i+1
end
{product = (n!/k!)}
```

You may assume that all values are integers and that k and n are positive integers with $k < n$. Determine an invariant, I , for the while statment and show that for that invariant

- a. $(I \text{ and not } i \leq n) \Rightarrow (\text{product} == n!/k!)$
- b. $(I \text{ and } i \leq n) \Rightarrow \text{wp}(\text{product}=\text{product}*i; i=i+1, \{I\})$

Note that part a. should be simply an algebraic and logic manipulation – it doesn't involve any program code. Part b. requires application of the statement sequence rule as well as algebraic and logical manipulation.

Determine a precondition, P , for the program as a whole and show that

- c. $P \Rightarrow \text{wp}(i=k+1; \text{product}=1, \{I\})$

When you have done all of these things you will have shown that

```
{P}
i = k+1;
product = 1;
{I}
while (i<=n) do
  {I and i<=n}
  product = product*i;
  i = i+1
  {I}
end
{I and not i<=n}
{product == n!/k!}
```

is a correctly asserted program using your assertions P and I .

6. What are the ML types of the following functions? Figure them out by hand then check with an ML system.

```
fun f [] = false
  | f (x::xs) = if x=7 then true else (f xs)
fun g [] = 0.0
  | g ((p,q)::xs) = (p*q)+g(xs)
```

7. What is the value of $(f\ 4)$ if static scoping is used? if dynamic scoping is used? Draw the call stack with static and dynamic links at the point marked by \leftarrow in the case that static scoping is used.

```
fun f x = let
  fun g z = x
  fun h y = let
```

```
    val x = y-1
  in
    g x          <-----
  end
in
  h x
end
```