

### Question 6.1c

Derive the type of function `c` declared using

```
fun c(f) = fn y => f(y)
```

begin by naming the type of each symbol and subexpression:

```
c: 'c (which is what we are looking for)
f: 'f
y: 'y
f(y) : 'a
fn y => f(y) : 'b
```

now write equations expressing the relationships between these:

```
'f = 'y -> 'a (because the term f(y) tells us that 'f is the type of a function applied
to a value of type 'y yielding a value of type 'a)
'b = 'y -> 'a (because fn y => f(y) is a function taking a value of type 'y and
yielding a value of the type of f(y) which is 'a.)
'c = 'f -> 'b = ('y->'a) ->'y->'a
```

Then renaming the type variables and expressing the type as the compiler would gives

```
c: ('a->'b) ->'a->'b
```

### Question 7.8

a) Under static scoping, `x` on line 2 is bound to 2, on line 4 is bound to 7 and on line 5 is bound to 7. The result of the ML program is 16 (the C program isn't an expression so it doesn't have a value, per se.)

b) Under dynamic scoping, all three uses of `x` would yield 7 as the value and the overall result would be 21.

(There are syntax errors on lines 1 and 2 of the problem:

```
let x=2 in
  let val fun f(y) = x+y in
```

```
    should say
    let val x=2 in
      let fun f(y) = x+y in
```

```
)
```

### Question 6.8

One expects that the type of `(reduce plus [1,2,3])` would be `int` but the type given in the problem says it would be `'int list`. You also expect that the type of the function passed to `reduce` should be `('a*'a->'a)` but in the type given for `reduce` it is `('a*'a list->'a list)`.

We might also ask ourselves what is wrong with the function definition. There is an obvious mistake having to do with the first pattern. If `reduce` is applied to *any* two

arguments of the appropriate types the result is just the second argument because the first clause of the definition,  $\text{fun reduce } (f,x) = x$ , will match. (The assertion that this matches only a 1-element list is flat-out wrong.) While Moscow ML will accept the definition and gives it the type shown in the book, Standard ML of New Jersey flags the definition as erroneous because the second clause of the definition can never be used.

Exercise: show how the type given in the book is derived from the definition of reduce.

### Question 6 from the homework handout

Strategies for this kind of problem:

Look at the values of the variables for 0, 1, 2, ... iterations

<i>Iteration</i>	<i>i</i>	<i>product</i>
0	k+1	1
1	k+2	(k+1)
2	k+3	(k+1)(k+2)
3	k+4	(k+1)(k+2)(k+3)

In a problem like this, the invariant will often have two parts, one that lets you conclude the value of  $i$  at the end of the loop and the other that lets you conclude the value of product. Consider  $i$  first. If you just think informally about what the loop is doing you should see that  $i$  will have value  $n+1$  when the loop terminates. What invariant would let you formally conclude that?  $i \leq n+1$ . When the loop terminates the while rules says that  $(i \leq n+1)$  and  $(\text{not } i \leq n)$  which together say  $(i = n+1)$ . So  $(i \leq n+1)$  is one part of the proposed invariant.

For the other part, we need something that will let us conclude that  $(\text{product} = n! / k!)$ . Look at the product column above. Observe that at each step  $(\text{product} = (i-1)! / k!)$ . This becomes the second part of the proposed invariant, so the complete proposed invariant is  $((i \leq n+1) \text{ and } (\text{product} = (i-1)! / k!))$ . Let's check parts d) and e):

e)  $((i \leq n+1) \text{ and } (\text{product} = (i-1)! / k!) \text{ and } (\text{not } i \leq n)) \Rightarrow (i = n+1 \text{ and } (\text{product} = (i-1)! / k!)) \Rightarrow \text{product} = (n! / k!)$   
 QED

d)  $\text{wp}(\text{product} = \text{product} * i; i = i + 1, ((i \leq n+1) \text{ and } (\text{product} = (i-1)! / k!))) == \text{wp}(\text{product} = \text{product} * i, ((i+1 \leq n+1) \text{ and } (\text{product} = (i)! / k!))) == ((i+1 \leq n+1) \text{ and } (\text{product} * i = (i)! / k!))$

So now, we ask does

$((i \leq n+1) \text{ and } (\text{product} = (i-1)! / k!) \text{ and } i \leq n) \Rightarrow (i+1 \leq n+1) \text{ and } (\text{product} * i = (i)! / k!))?$

Well,  $((i \leq n+1) \text{ and } (\text{product} = (i-1)! / k!) \text{ and } i \leq n) \Rightarrow ((\text{product} = (i-1)! / k!) \text{ and } i+1 \leq n+1) \Rightarrow$

$(\text{product} * i == (i-1)! * i / k!) \text{ and } i+1 \leq n+1) ==$   
 $(\text{product} * i == i! / k!) \text{ and } i+1 \leq n+1)$

QED

Now part a: determine a precondition for the program as a whole by finding

$\text{wp}(i=k+1; \text{product}=1, ((i \leq n+1) \text{ and } (\text{product} == (i-1)! / k!))) ==$   
 $\text{wp}(i=k+1, (i \leq n+1) \text{ and } (1 == (i-1)! / k!)) ==$   
 $((k+1 \leq n+1) \text{ and } (1 == (k+1-1)! / k!)) ==$   
 $((k \leq n) \text{ and } 1 == 1) ==$   
 $k \leq n$