val \(x = 1\)
fun \(f\ y = x + y\)
val \(x = 2\)
val \(y = 3\)
val \(z = f (x + y)\)

(* second example *)
val \(x = 1\)
fun \(f\ y =\)
let
val \(x = y + 1\)
in
\(\lambda z \Rightarrow x + y + z\)
end
val \(x = 3\)
val \(g = f 4\)
val \(y = 5\)
val \(z = g 6\)

(* third example *)
fun \(f\ g =\)
let
val \(x = 3\)
in
\(\lambda z \Rightarrow x + y + z\)
end
val \(x = 4\)
fun \(h\ y = x + y\)
val \(z = h\)

(* why lexical scope *)

fun \(f1\ y =\)
let
val \(x = y + 1\)
in
\(\lambda z \Rightarrow x + y + z\)
end
fun \(f2\ y =\)
let
val \(q = y + 1\)
in
\(\lambda z \Rightarrow q + y + z\)
end
val \(x = 17\) (* irrelevant *)
val \(a1 = (f1 7) 4\)
val \(a2 = (f2 7) 4\)

(* f3 and f4 are always the same, no matter what argument is passed in *)
fun \(f3\ g =\)
let
val \(x = 3\) (* irrelevant *)
in
\(g 2\)
end
fun \(f4\ g =\)
\(g 2\)

Jan 06, 18 14:22 lec08−code.sml Page 2/3

val \(x = 17\)
val \(a3 = f3 (\lambda y \Rightarrow x + y)\)
val \(a4 = f3 (\lambda y \Rightarrow 17 + y)\)

(* under dynamic scope, the call "g 6" below would try to add a string (from looking up \(x\)) and would have an unbound variable (looking up \(y\)), even though f1 type-checked with type int -> (int -> int) *)
val \(x = "hi"\)
val \(g = f1 7\)
val \(z = g 4\)

(* Being able to pass closures that have free variables (private data) makes higher-order functions /much/ more useful *)
fun \(filter\ (f,xs) =\)
\case \(xs\) of
\[\] ⇔ \[
| x::xs' ⇒ if \(f\ x\) then x::(filter(f,xs')) else filter(f,xs')
end
fun \(greaterThanX\ x = \lambda y \Rightarrow y > x\)
fun \(noNegatives\ xs = filter(greaterThanX -1, xs)\)
fun \(allGreater\ (xs,n) = filter (\lambda x \Rightarrow x > n, xs)\)
fun \(allShorterThan1\ (xs,s) = filter (\lambda x \Rightarrow String.size x < (print "!"); String.size s), xs)\)
fun \(allShorterThan2\ (xs,s) =\)
let
val \(i = (print "!"); String.size s\)
in
filter(\(\lambda x \Rightarrow String.size x < i), xs)\)
end
val _ = print "withAllShorterThan1: "
val \(x1 = allShorterThan1(["1","333","22","4444"],xxx)\)
val _ = print "withAllShorterThan2: "
val \(x2 = allShorterThan2(["1","333","22","4444"],xxx)\)
val _ = print "w"
fun f8 (xs, s) = 
  let val i = String.size s 
  in fold((λ (x, y) ⇒ x ∧ String.size y < i), true, xs) 
  end 
fun f9 (g, xs) = fold((λ(x, y) ⇒ x ∧ g y), true, xs) 
fun f8again (xs, s) = 
  let val i = String.size s 
  in 
    f9(λ y ⇒ String.size y < i, xs) 
  end