Pruning closures

in your environment-based interpreter

COS 326

Presented by: Andrew W. Appel

Princeton University

A remark about homework 4

WHY IT'S IMPORTANT TO PRUNE CLOSURE ENVIRONMENTS

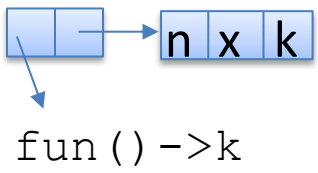
Pruning environments

```
let zeros i = if i=0 then [] else 0 :: s(i-1)

let h (n: int) : int =
  let f x =
    let k = List.length x in
    fun () -> k
  in
  let rec g i : (unit->int) list =
    if i=0 then [] else f (zeros n) :: g (i-1)
  in let bigdata = g n
  in List.fold_left (fun s u -> u()+s) 0 bigdata

let a = h 1000
```

Pruning environments

```
let zeros i = if i=0 then [] else 0 :: s(i-1)

let h (n: int) : int =
  let f x =
    let k = List.length x in
    fun () -> k
  in
  let rec g i : (unit->int) list =
    if i=0 then [] else f (zeros n) :: g (i-1)
  in let bigdata = g n
  in List.fold_left (fun s u -> u()+s) 0 bigdata

let a = h 1000
```

What variables are *in scope at this point* ?



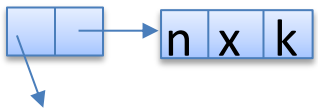
You *could* build a closure environment with all the variables currently in scope.

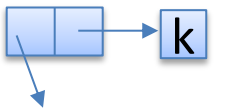
Pruning environments

```
let zeros i = if i=0 then [] else 0 :: s(i-1)

let h (n: int) : int =
  let f x =
    let k = List.length x in
    fun () -> k
  in
  let rec g i : (unit->int) list =
    if i=0 then [] else f (zeros n) :: g (i-1)
  in let bigdata = g n
  in List.fold_left (fun s u -> u()+s) 0 bigdata

let a = h 1000
```

What are the free variables of this function?

5 words of memory versus 3 words, what's the big deal?

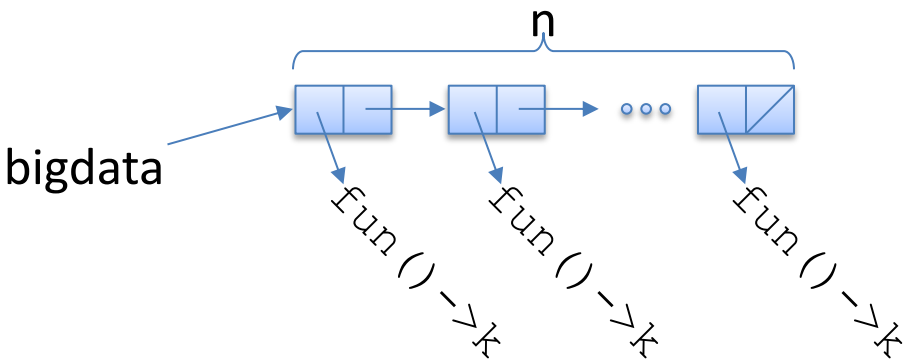




Pruning environments

```
let zeros i = if i=0 then [] else 0 :: s(i-1)

let h (n: int) : int =
  let f x =
    let k = List.length x in
    fun () -> k
  in
  let rec g i : (unit->int) list =
    if i=0 then [] else f (zeros n) :: g (i-1)
  in let bigdata = g n
  in List.fold_left (fun s u -> u()+s) 0 bigdata

let a = h 1000
```

Run the program to here, and what is in memory?

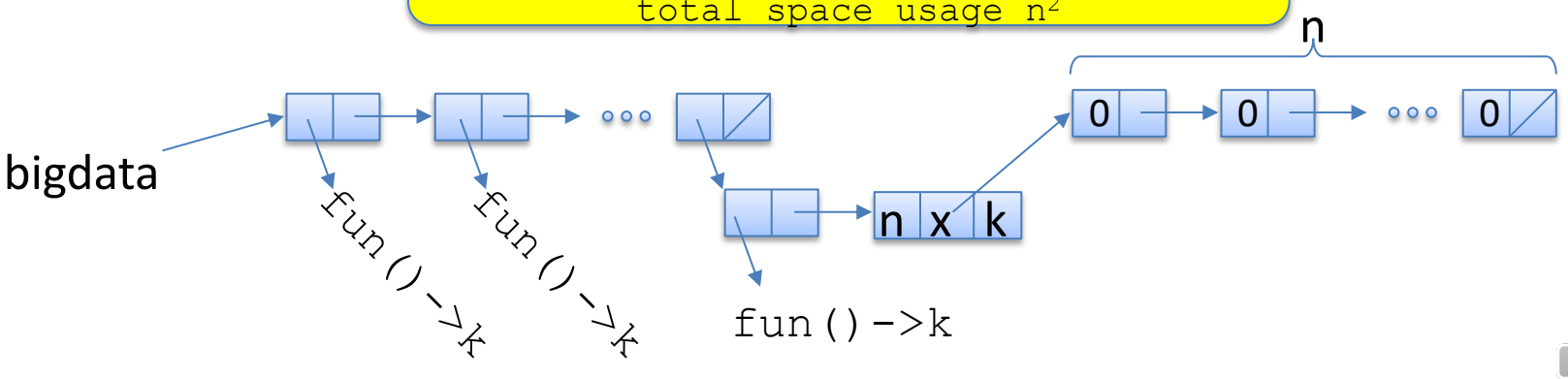


Pruning environments

```
let zeros i = if i=0 then [] else 0 :: s(i-1)

let h (n: int) : int =
  let f x =
    let k = List.length x in
    fun () -> k
  in
  let rec g i : (unit->int) list =
    if i=0 then [] else f (zeros n) :: g (i-1)
  in let bigdata = g n
  in List.fold_left (fun s u -> u()+s) 0 bigdata

let a = h 1000
```

What variables are *in scope at this point* ?

n closures for (fun()->k),
each is a list of length n,
total space usage n^2

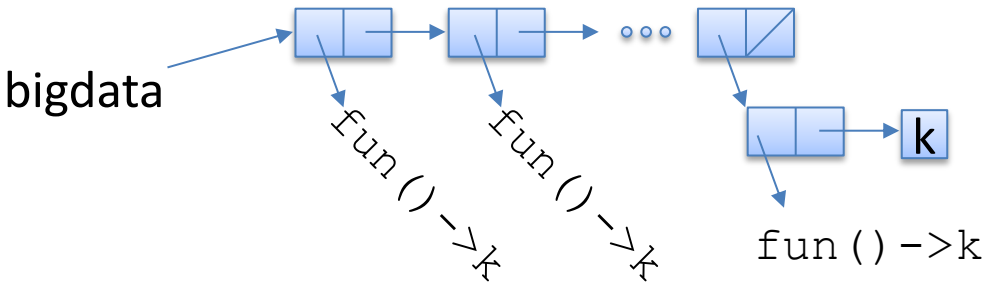


Pruning environments

```
let zeros i = if i=0 then [] else 0 :: s(i-1)

let h (n: int) : int =
  let f x =
    let k = List.length x in
    fun () -> k
  in
  let rec g i : (unit->int) list =
    if i=0 then [] else f (zeros n) :: g (i-1)
  in let bigdata = g n
  in List.fold_left (fun s u -> u()+s) 0 bigdata

let a = h 1000
```

What are the free variables of this function?

n closures for (fun()->k),
each is just a number k,
total space usage O(n)



Therefore

Closures should represent *only* the free variables of a function (not *all the variables currently in scope),*

otherwise the compiled program may use *asymptotically more space,*

such as $O(n^2)$ instead of $O(n)$