COL 100 Lecture 9

More Iteration and Lists

S(N)=Sum of First N Fibonacci Numbers

Develop an algorithm to compute the sum of the first n Fibonacci numbers. The algorithm should work in O(n) time and O(1) space.

SOSML Editor Files

SML

fib Store Share

```
1  fun fib (n) =
2          let fun fib_iter(n,a,b,count)= if count=n then a+b
3                                         else fib_iter(n,b,a+b,count+1);
4          in  if n<=2 then 1 else fib_iter(n,1,1,3)
5  end
6  val test= fib(8);
7
8  Output
9  > val fib = fn: int → int;
10 > val test = 21: int;
11
```

INV a=fib(count-2) b=fib(count-1)

$F_1 = 1$
$F_2 = 1$
$F_3 = 2$
$F_4 = 3$
$F_5 = 5$
$F_6 = 8$
$F_7 = 13$
$F_8 = 21$
⋮

How about using Sum_iter

```
fun sum(a,b) =
      let fun sum_iter(c,cf,s) =
                if c=cf+1 then s
                else sum_iter(c+1,cf,s+fib(c));
      in
          sum_iter(a,b,0)
end;
sum(1,8);
```

Output

```
> val fib = fn: int → int;
> val test = 21: int;
> val sum = fn: int * int → int;
> val it = 54: int;
```

Time Complexity:

Sum calls fib(1), fib(2)….. fib(n) -> 1 + 1 + 2 +3 …+n = $O(n^2)$

S(N)=Sum of First N Fibonacci Numbers

Develop an algorithm to compute the sum of the first n Fibonacci numbers. The algorithm should work in O(n) time and O(1) space.

$F_1 = 1$ — $S_1=1$

$F_2 = 1$ — $S_2=2$

$F_3 = 2$ — $S_3=4$

$F_4 = 3$ — $S_4=7$

$F_5 = 5$ — $S_5=12$

$F_6 = 8$ — $S_6=20$

$F_7 = 13$ — $S_7=33$

$F_8 = 21$ — $S_8=54$

$\vdots$

S(N)=Sum of First N Fibonacci Numbers

Develop an algorithm to compute the sum of the first n Fibonacci numbers. The algorithm should work in O(n) time and O(1) space.

$F_1 = 1$
$F_2 = 1$
$F_3 = 2$
$F_4 = 3$
$F_5 = 5$
$F_6 = 8$
$F_7 = 13$
$F_8 = 21$
$\vdots$

ITERATIVE STEP

$S_1=1$
$S_2=2$
$S_3=4$
$S_4=7$
$S_5=12$
$S_6=20$
$S_7=33$
$S_8=54$

a	b	sum	count
b	a+b	sum+a+b	count+1

S(N)=Sum of First N Fibonacci Numbers

Develop an algorithm to compute the sum of the first n Fibonacci numbers. The algorithm should work in O(n) time and O(1) space.

SOSML Editor Files

SML fib_sum Store Share

```
1
2  fun fib_sum (n) =
3          let fun fib_iter(n,a,b,s,count)= if count=n then s+a+b
4                                          else fib_iter(n,b,a+b,s+a+b,count+1);
5          in  if n=1 then 1 else if n=2 then 2 else fib_iter(n,1,1,2,3)
6  end
7  val test= fib_sum(8);
8
9
```

Introduce new variable s

INV a=fib(count-2) b=fib(count-1), s=S(count-1)

$F_1 = 1$	$S_1=1$
$F_2 = 1$	$S_2=2$
$F_3 = 2$	$S_3=4$
$F_4 = 3$	$S_4=7$
$F_5 = 5$	$S_5=12$
$F_6 = 8$	$S_6=20$
$F_7 = 13$	$S_7=33$
$F_8 = 21$	$S_8=54$
⋮	

Example – Summation of a function

Iterative computation of $\sum_a^b f(n)$

$$sum(a, b) = sum\_iter(a, b, 0)$$

where, the auxiliary function $sum\_iter : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is given as

$$sum\_iter(c, c_f, s) = \begin{cases} s & \text{if } c = c_f + 1 \\ sum\_iter(c + 1, c_f, s + f(c)) & \text{otherwise} \end{cases}$$

⟨*Iterative sum*⟩≡

```
fun sum (a, b) =
    let ⟨Code for sum_iter⟩
     in sum_iter (a, b, 0)
    end;
```

⟨*Code for* sum_iter⟩≡

```
fun sum_iter (c, cf, s) =
    if c = cf+1 then s
    else sum_iter (c+1, cf, s + f(c));
```

What if we want to find maximum of f() in range from a to b?

maximum of f() in range from a to b

ITERATIVE STEP

f(a)			
f(a+1)			
f(a+2)	cur_max	count_final	count
.....	max(f(count),cur_max)	count_final	count+1
f(b)			

maximum of f() in range from a to b

SML

max_iter | Store | Share

```
1
2  fun max(f,a,b) =
3        let fun max_iter(c,cf,m) =
4                  if c=cf+1 then m
5                  else  max_iter(c+1,cf,if f(c)>m then f(c) else m);
6        in
7           max_iter(a,b,f(a))
8  end;
9  fun foo(x) = x*x-6*x;
10 foo(8);
11 max(foo,1,8);
12 max(foo,1,5);
```

Output

```
> val max = fn: (int → int) * int * int → int;
> val foo = fn: int → int;
> val it = 16: int;
> val it = 16: int;
> val it = ~5: int;
```

Problems (exercise 3)

Define a tail-recursive (iterative) algorithm for the function

$$f : N \rightarrow Z \;\; is\ 1\ if\ n = 0 \;\; and\ is\ n - f(n - 1)\ otherwise$$

Define an invariant property for the above algorithm.

```
fun f (n) =
        if n=0 then 1
        else n - f(n-1);
end
```

By inspection

0 1 2 3 4 5 6 7 8 9

1 0 2 1 3 2 4 3 5 4

Excercise 3 (contd)

0 1 2 3 4 5 6 7 8 9

1 0 2 1 3 2 4 3 5 4

for n>=2

$f(n) = n - f(n-1) = n - [(n-1) - f(n-2)] = 1 + f(n-2)$

so:

$f(n) = 1 + f(n-2)$

$f[0]=1 \quad f[1]=0$

Power by repeated squaring (iterative)

Design iterative process that uses successive squaring to compute x^n in O(lg n)

- ex find x^{11}
- Repeated squaring gives x, x^2, x^4, x^8
- $x^{11} = x * x^2 * x^8$
- Hint write 11 in binary!

Lists

A list consists of one or more items of same type T

- val l=[1,2,7,2,12,~2];
- val ls=[fib(1),fib(7),fib(5),fib(4)];
- val lj=l@ls;
- val tlist=[#"H",#"E",#"L",#"L",#"O"];
- val x= String.implode tlist; gives string “HELLO”

Lists (cont'd)

```
val names = [ "Fred", "Jane", "Alice" ]    (* : string list *)
(* Even lists of lists of things *)
val groups = [ [ "Alice", "Bob" ],
               [ "Huey", "Dewey", "Louie" ],
               [ "Bonnie", "Clyde" ] ]     (* : string list list *)
val names_count = List.length names     (* gives 3 *)
(* You can put single values in front of lists of the same kind using the :: operator, called "the
cons operator" *)
val more_numbers = 13 :: numbers  (* gives [13, 1, 3, 3, 7, ...] *)
val more_groups  = ["Batman","Superman"] :: groups
(* Lists of the same kind can be appended using the @ ("append") operator *)
val guest_list = [ "Mom", "Dad" ] @ [ "Aunt", "Uncle" ]
(* This could have been done with the "cons" operator.  It is tricky because the
   left-hand-side must be an element whereas the right-hand-side must be a list
   of those elements. *)
val guest_list = "Mom" :: "Dad" :: [ "Aunt", "Uncle" ]
val guest_list = "Mom" :: ("Dad" :: ("Aunt" :: ("Uncle" :: [])))
```

List operations

`null` *l*

returns `true` if the list *l* is empty.

`length` *l*

returns the number of elements in the list *l*.

l1 `@` *l2*

returns the list that is the concatenation of *l1* and *l2*.

`hd` *l*

returns the first element of *l*. It raises `Empty` if *l* is `nil`.

`tl` *l*

returns all but the first element of *l*. It raises `Empty` if *l* is `nil`.

`last` *l*

returns the last element of *l*. It raises `Empty` if *l* is `nil`.

Tuples

ordered pair

- (“Rahul”,5) — types can be mixed
- n-tuple type1 * type2 * type 3 … *type n
  - (12,”cat”,-7,true,2.17)
- List of tuples
  - [(“Rahul”,5),(“Pratul”,15),…..,(“Reena”,~2)] OK
  - [(“Rahul”,5),(“Pratul”,1.5),…..,(“Reena”,~2)] NOT OK

Records

Records are tuples with named slots

```
val rgb = { r=0.23, g=0.56, b=0.91 } (* : {b:real, g:real, r:real} *)
(* You don't need to declare their slots ahead of time. Records with
   different slot names are considered different types, even if their
   slot value types match up. For instance... *)
val Hsl = { H=310.3, s=0.51, l=0.23 } (* : {H:real, l:real, s:real} *)
val Hsv = { H=310.3, s=0.51, v=0.23 } (* : {H:real, s:real, v:real} *)
(* ...trying to evaluate `Hsv = Hsl` or `rgb = Hsl` would give a type
   error. While they're all three-slot records composed only of `real`s,
   they each have different names for at least some slots. *)

(* You can use hash notation to get values out of records tuples. *)
val H = #H Hsv (* : real *)
val s = #s Hsl (* : real *)
```

Working with Lists

Find Largest number in list

```
fun largest [] = raise Empty
 | largest [x] = x
 | largest (x::xs) =
       let
          val y=largest(xs);
       in if x > y then x else y
end;
```

Tail Recursive version??

Working with Lists

Find Largest number in list — tail recursive

```
fun tlargest(x)=
     let
         fun largest_it (max, []) = max
         | largest_it (max, (x::xs)) =
           let val maxn= if x > max then x else max;
           in   largest_it(maxn,xs)
         end
     in
       largest_it(0,x)
end;
```

Largest number in List

```
7  val l=[1,2,7,2,12,~2];
8  val ls=[fib(1),fib(7),fib(5),fib(4)];
9  val jl=l@ls;
10
11 fun largest [] = raise Empty
12   | largest [x] = x
13   | largest (x::xs) =
14       let
15           val y=largest xs;
16       in if x > y then x else y
17 end;
18 largest(l);
19 fun tlargest(x)=
20      let
21          fun largest_it (max, []) = max
22          | largest_it (max, (x::xs)) =
23            let val maxn= if x > max then x else max;
24            in   largest_it(maxn,xs)
25          end
26      in
27        largest_it(0,x)
28 end;
29 tlargest(l);
30 tlargest(jl);
```

Output

```
> val fib = fn: int → int;
> val l = [1, 2, 7, 2, 12, ~2]: int list;
> val ls = [1, 13, 5, 3]: int list;
> val jl = [1, 2, 7, 2, 12, ~2, 1, 13, 5, 3]: int list;
> val largest = fn: int list → int;
> val it = 12: int;
> val tlargest = fn: int list → int;
> val it = 12: int;
> val it = 13: int;
```

Working with Lists

Find Average of number in list — tail recursive

```
fun average(x)=
     let
         fun sum_it (sum, []) = sum
         | sum_it (sum, (x::xs)) =
              sum_it(sum+x,xs)
         end
     in
       Real.fromInt(sum_it(0,x))/Real.fromInt(length(x));
end;
```

Find Average of number in list — tail recursive

SML

average Store

```
1
2
3
4  val l=[1,2,7,2,12,~2];
5
6  fun average(x)=
7      let
8          fun sum_it (sum, []) = sum
9          | sum_it (sum, (x::xs)) =
10             sum_it(sum+x,xs)
11     in
12       Real.fromInt(sum_it(0,x))/Real.fromInt(length(x))
13 end;
14 average(l);
15
16
17
```

Output

```
> val l = [1, 2, 7, 2, 12, ~2]: int list;
> val average = fn: int list → real;
> val it = 3.6666666666666665: real;
```