Lecture 5

Tuesday, September 7, 2021 2:09 PM

let rec sum n = if n = 0 then 0 else n + sum(n-1)

sum 4 ⟹ if $\underbrace{4 = 0}_{\text{evaluates this first}}$ then $\underbrace{0}$ else $\underbrace{4 + \text{sum}(4-1)}$

"Thunks"

⟹ if false then 0 else 4 + sum (4-1)

⟹ 4 + sum (4 - 1)

(+) 4 (sum ((-) 4 1))

(In more verbose notation)

⟹ 4 + sum 3

⟹ 4 + (if 3 = 0 then 0 else 3 + sum(3-1))

⟹ 4 + (if false then 0 else 3 + sum(3-1))

⟹ 4 + (3 + sum (3-1))

⟹ 4 + (3 + sum 2)

⟹ ...

⟹ 4 + (3 + (2 + (1 + 0)))

⟹ 4 + (3 + (2 + 1))

⟹ 4 + (3 + 3)

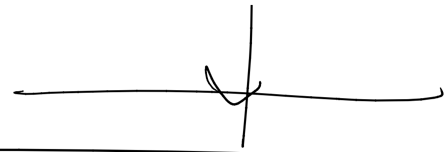
⟹ 4 + 6

⟹ 10

Buildup

Unwinding

$\Rightarrow 10$



let rec sum n = if n=0 then 0 else n+ $\underbrace{\text{sum (n-1)}}$

(1) Make recursive call

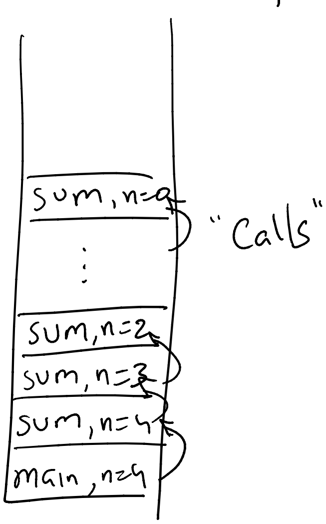
"Continuation"

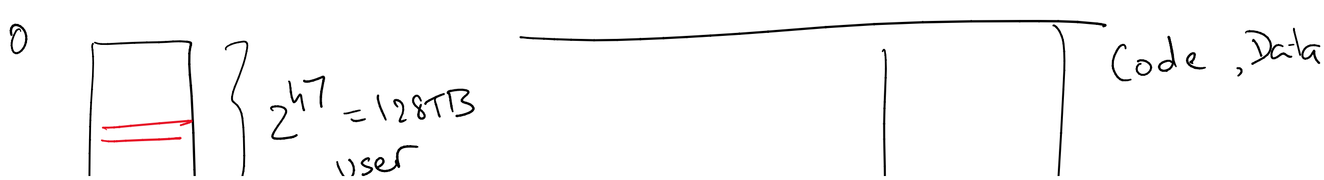
"What to do with the result of a fn call"

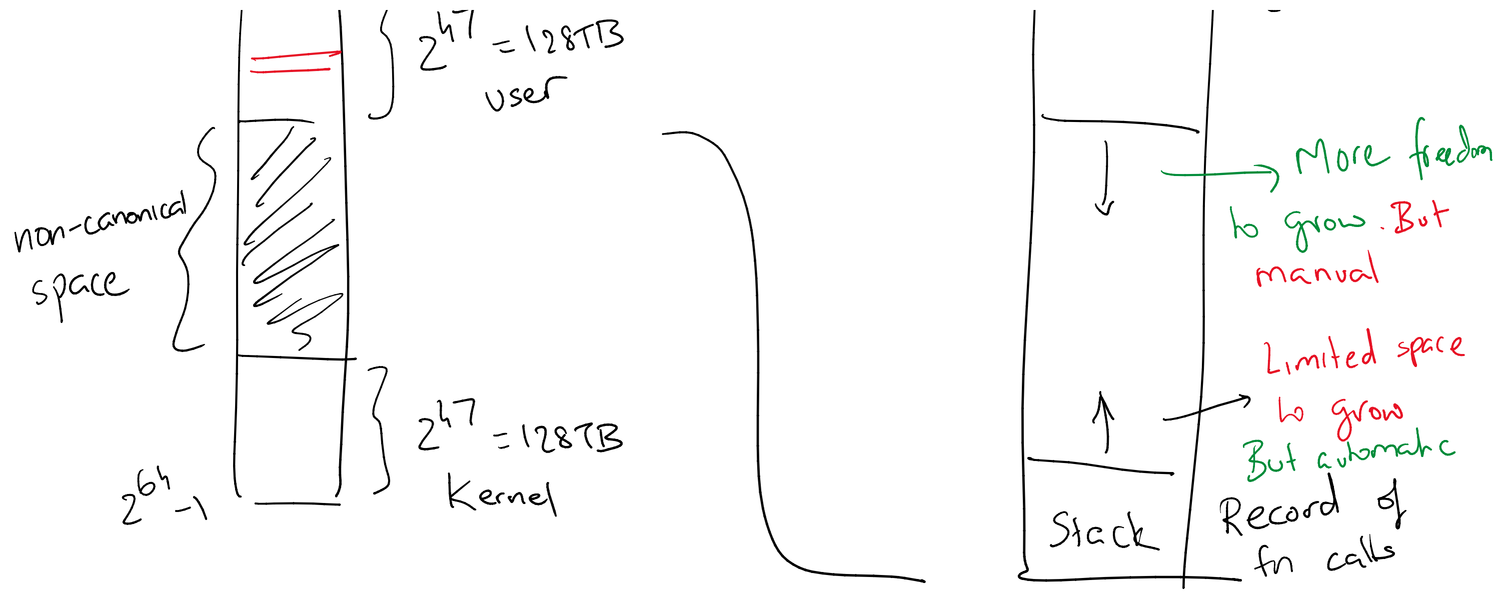
(2) Add n to the result

(3) Return.

Continuations : represented as a stack








```
let sum2 n =
    let _sum acc n = if n=0 then acc
                     else _sum (acc+n) (n-1) in
    _sum 0 n
```

Question: Why doesn't sum2 appear to run out of space?

```
_sum 0 4 => if 4=0 then 0 else _sum (0+4) (4-1)
  => if false then 0 else _sum (0+4) (4-1)
  => _sum (0+4) (4-1)
```

$\Rightarrow$ _sum 4 3

$\Rightarrow$ if 3=0 then 4 else _sum (4+3) (3-1)

$\Rightarrow$ ...

$\Rightarrow$ _sum 7 2

$\Rightarrow$ ...

$\Rightarrow$ _sum 9 1

$\Rightarrow$ ...

$\Rightarrow$ _sum 10 0

$\Rightarrow$ ...

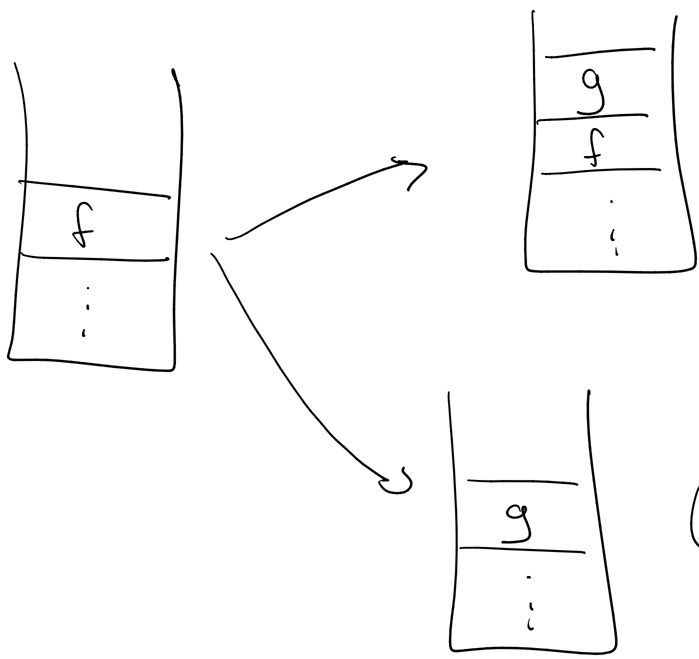
$\Rightarrow$ 10

- No unwinding necessary!
- If last activity performed by a call to f is to call g (Tail call)

then replace stack frame of f with

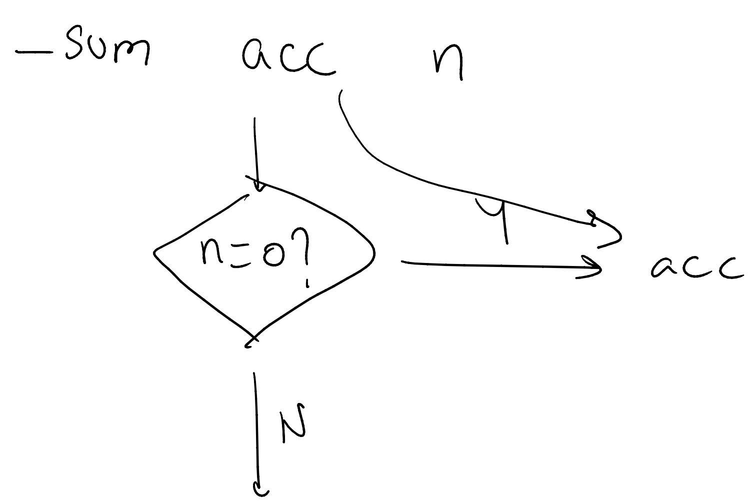
then replace stack frame of f with stack frame of g. (Tail call optimization)

- So size of stack is $O(1)$.

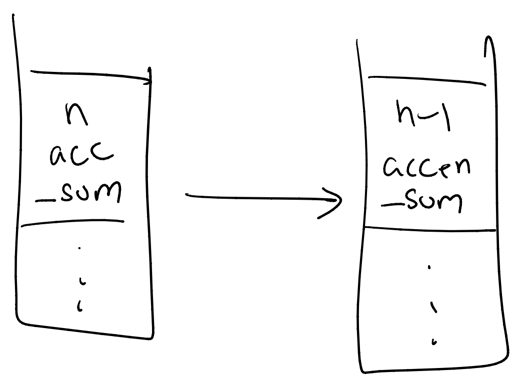


(By default)

(TCO)



_sum (acc+n) (n-1)




```
while (n>0) {
  acc:=acc+n;
  n:=n-1;
}
```

gcc ds.c

sum 4 ⇒ 4 + sum 3

⇒ 4 + (3 + sum 2)

⇒ ...

⇒ 4 + (3 + (2 + (1+0)))

gcc -O2 ds.c

acc:=acc+n

n:=n-1

$\Rightarrow \ldots (3 + (2 + (1+0)))$

acc	0	4	(4+3)	(4+3)+2	((4+3)+2)+1
n	4	3	2	1	0

Transformation — Ocaml sum $\Rightarrow$ sum2

— C ds.c $\Rightarrow$ ss.c

— Both only work because addition is associative.

n + sum (n-1) $\Longrightarrow$ _sum (acc + n) (n-1)

Question : What if + was not associative?

let rec even n = match n with

| 0 → true
| 1 → false
| _ → not (even (n-1))

---

even 4 ⇒ not (even 3)

⇒ . . .

⇒ not (not (even 2))

⇒ . .

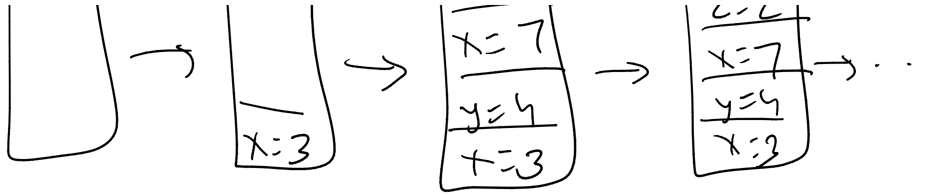
⇒ $\underbrace{\text{not}}_{e4}$ ($\underbrace{\text{not}}_{e3}$ ($\underbrace{\text{not}}_{e2}$ (even 1)))

Oldest not, last to be evaluated

Newest not, first to be evaluated

```
let x = 3 in
let y = 4 in
let x = x + y in
let z = 2 in
```

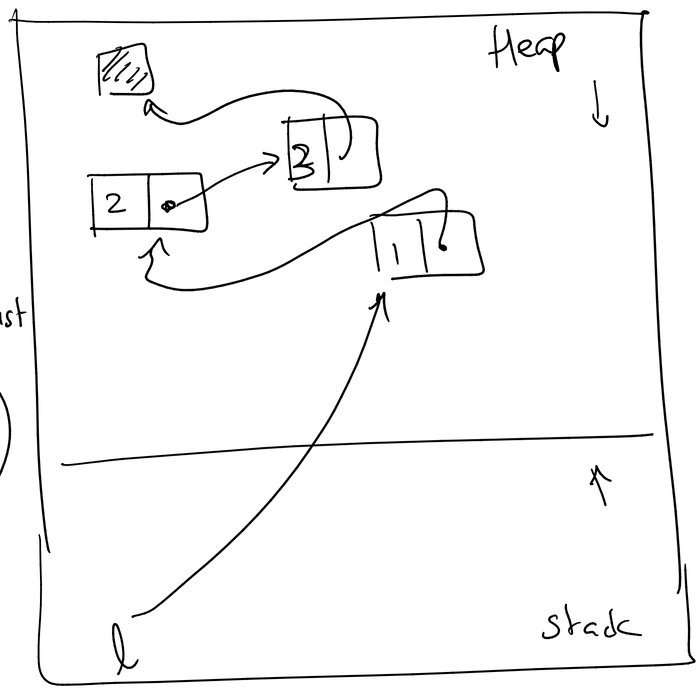
```
let x = x + y in
let z = 2 in
x + y + z
```



let l = [1,2,3]

type list = Nil
| Cons of int * list

C(1, C(2, C(3, Nil)))




```
let even2 n =
  let rec unwind stack acc =
    match stack with
    | [] -> acc
    | _ :: tl -> unwind tl (not acc) in
  let rec _even stack n = if n = 0 then (unwind stack true) else _even ("not" :: stack) (n - 1) in
  _even [] n
```

Lives on system heap

"Remember how many nots to apply."

let rec even n = match n with
| 0 -> true
| 1 -> false

| _ → not (even (n-1))

Result of recursive call

Deferred "action"