CS 162 Programming languages

Lecture 6: Recursion

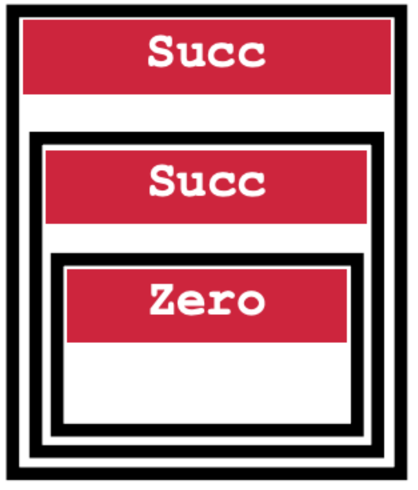
Yu Feng
Winter 2020

Recursive types

```
type nat = Zero | Succ of nat
```

What are values of nat ?
One nat contains another!

nat = recursive type



plus: nat*nat -> nat

Base pattern

Inductive pattern

```
type nat =
| Zero
| Succ of nat
```

Base pattern

Inductive pattern

```
let rec plus n m =
match m with
| Zero -> n
| Succ m’ -> Succ (plus n m’)
```

Base expression

Inductive expression

List datatype

```
type int_list =
  Nil
| Cons of int * int_list
```

Lists are a derived type: built using elegant core!

1. Each-of
2. One-of
3. Recursive

:: is just a syntactic sugar for “Cons”
[] is a syntactic sugar for “Nil”

List function: length

```
let rec len l =
    match l with
    | Nil -> 0
    | Cons(h,t) -> 1 + (len t)
```

Base pattern

Inductive pattern

Base expression

Inductive expression

List function: list_max

```
let rec list_max l =
    match l with
    | Nil -> 0
    | Cons(h,t) -> max h (list_max t)
```

Base pattern

Base expression

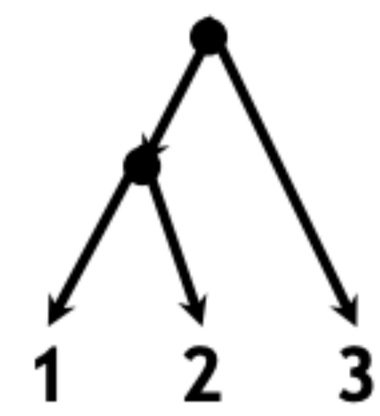
Inductive pattern

Inductive expression

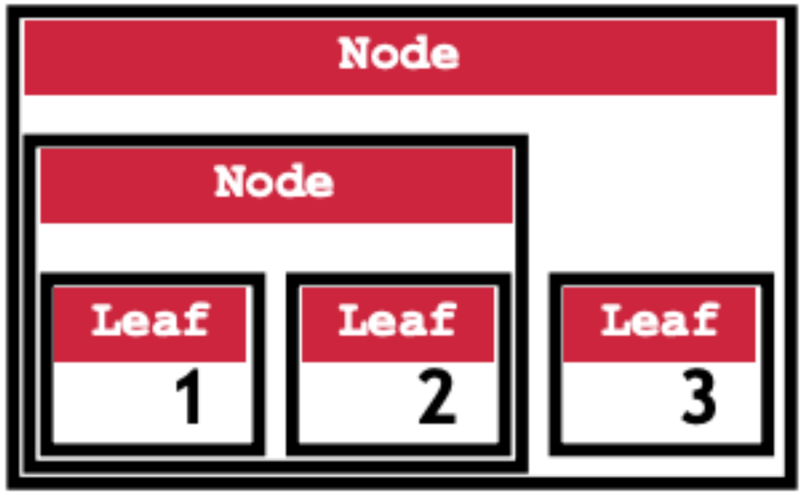
```
let max x y = if x > y then x else y;;
```

Representing Trees

```
type tree =
  Leaf of int
| Node of tree*tree
```



Node(Node(Leaf 1, Leaf 2), Leaf 3)



sum_leaf: tree -> int

```
type tree =
  Leaf of int
| Node of tree*tree
```

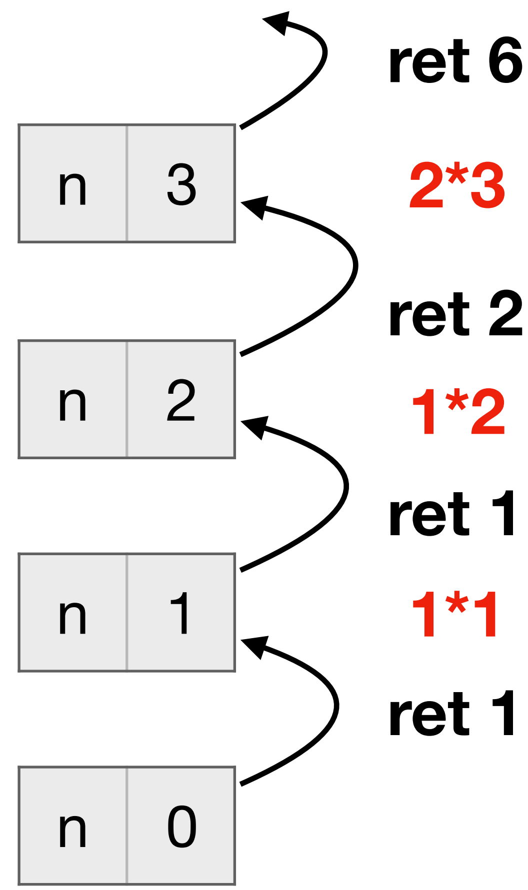
```
let rec sum_leaf t =
    match t with
    | Leaf n -> n
    | Node(t1,t2) -> (sum_leaf t1)
                    +(sum_leaf t2)
```

Factorial: int -> int

```
let rec fact n =
    if n<=0
    then 1
    else n * fact (n-1);;

fact 3;;
```

How does it execute?



Tail recursion

Tail recursion

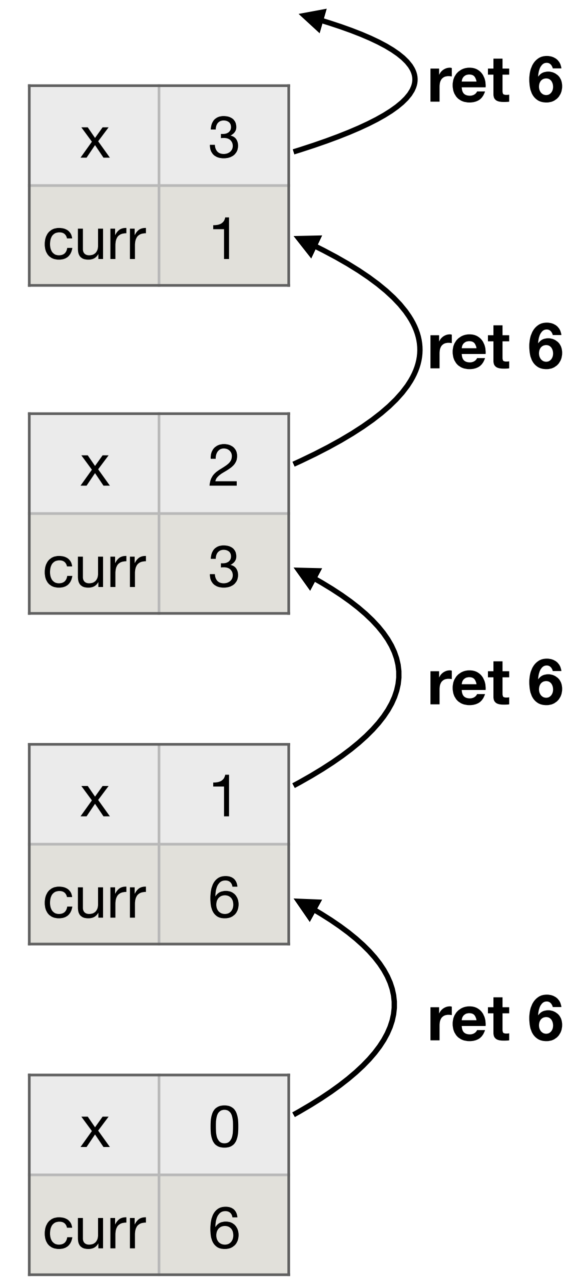
- Recursion where all recursive calls are immediately followed by a return

- In other words: not allowed to do anything between recursive call and return

Tail recursive Factorial

```
let fact x =
  let rec helper x curr =
     if x <= 0
     then curr
     else helper (x - 1) (x * curr)
  in
     helper x 1;;

fact 3;;
```



How does it execute?

Tail recursion

Tail recursion

- Recursion where all recursive calls are immediately followed by a return
- In other words: not allowed to do anything between recursive call and return

Why do we care about tail recursion?

- Tail recursion can be optimized into a simple loop

Compiler optimization

```
let fact x =
  let rec helper x curr =
    if x <= 0
    then curr
    else helper (x − 1) (x ∗ curr)
  in
    helper x 1;;
```

```
fact(x) {
  curr := 1;
  while (1) {
  if (x <= 0)
  then { return curr }
  else { x := x − 1;
         curr := (x ∗ curr) }}
}
```

Recursion

Loop