OCaml: Tail Recursion

Programming Languages

William Killian

Millersville University

Recursion

- A function is recursive when it calls itself
- Two requirements:
  1. **A base case** – the part of the function in which the problem can be trivially solved. Does not "call itself"
  2. **A recursive case** – the part of the function in which we make a smaller version of the same problem and often *combine*

- We can have multiple base cases
- We can have multiple recursive cases

Tail Recursion

- A function is **tail recursive** when its recursive call is the last thing executed by the function
- This means that there cannot be any additional operations done with the recursive call
- Tail recursion is harder to write at first but easy to adapt to over time

A Recursive Problem: sumToN

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

Is this tail recursive? Why or why not?

A Recursive Problem: sumToN

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

- What is the “state” of this function?
- If we had to write it with a loop, what variable(s) would we introduce?

Keeping Track of the State

```
let rec sumToN sum n =
  if n = 0 then
    (* base case: return the sum *)
    sum
  else
    (* recursive: change sum and n *)
    sumToN (sum + n) (n - 1)
```

Keeping Track of the State

```
let rec sumToN sum n =
   ...
      sumToN (sum + n) (n - 1)
```

This is **equivalent** to the following code:

```
sum = sum + n
n = n - 1
```

Problem: The Extra Parameter

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

We now must call our function like this:

```
    sumToN 0 15
```

Solution #1: Rebind (Don't do this)

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

let sumToN = sumToN 0
```

This is confusing

Solution #2: Create a Local Binding

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

This is better, but the names are confusing

Solution #3: Renamed Local Binding

```
let sumToN n =
  let rec sumHelper sum n =
    if n = 0 then
      sum
    else
      sumHelper (sum + n) (n - 1)
  in sumHelper 0 n
```

This is (likely) the best

Bonus: Removing arguments

```
let sumToN =
  let rec sumHelper sum n =
    if n = 0 then
      sum
    else
      sumHelper (sum + n) (n - 1)
  in sumHelper 0
```

This also works. Why?

So Why Even do Tail Recursion?

- Languages like when we use tail-recursion for two big reasons

    1. It is a while-loop hidden in disguise

    2. We only need to “update” the parameters and “jump” back to the top of the function

So we get our loops back, just not in the way you expected