tail recursion vs. forward recursion

Can someone give me the difference between these two kinds recursions and example (specifically in OCaml)?

recursion functional-programming ocaml

edited Sep 9 '14 at 0:58
AstroCB
5,664 ● 9 ● 26 ● 45

asked Jun 15 '10 at 6:22
REALFREE
1,328 ● 1 ● 17 ● 39

2 Answers

A tail recursive function is a function where the only recursive call is the last one in the function. A non-tail recursive function is a function where that is not the case.

A backward recursion is a recursion where in each recursive call the value of the parameter is less than in the previous step. A forward recursion is a recursion where it grows bigger with each step.

Those are two orthogonal concepts, i.e. a forward recursion may or may not be tail-recursive and the same applies to backward recursions.

For example the factorial function is often written like this in imperative languages:

```
fac = 1
for i from 1 to n:
    fac := fac * i
```

The common recursive version of factorial counts backwards (i.e. it calls itself with `n-1` as the parameter), however if you'd directly translate the above imperative solution, you'd come up with a recursive version that counts upwards. It would look something like this:

```
let fac n =
  let rec loop i =
    if i >= n
    then i
    else i * loop (i+1)
  in
    loop 1
```

This is a forward recursion and as you can see it is slightly more cumbersome than the backward recursive variant as it requires a helper function. Now this is not tail recursive as the last call in `loop` is the multiplication, not the recursion. So to make it tail-recursive, you'd do something like this:

```
let fac n =
  let rec loop acc i =
    if i >= n
    then acc
    else loop (i*acc) (i+1)
  in
    loop 1 1
```

Now this is both a forward recursion and a tail recursion because the recursive call is a) a tail-call and b) calls itself with a greater value (`i+1`).

edited Sep 26 '12 at 14:57

answered Jun 15 '10 at 8:07
sepp2k
186k ● 20 ● 427 ● 494

Here's an example of a tail recursive factorial function:

```
let fac n =
let rec f n a =
    match n with
    0 -> a
    | _ -> f (n-1) (n*a)
in
f n 1
```

Here is it's non-tail recursive counterpart:

```
let rec non_tail_fac n =
match n with
0 -> 1
| _ -> (non_tail_fac n-1) * n
```

The tail recursive function uses an accumulator, a, to store the value of the result of the previous call. This allows OCaml to perform tail call optimization which results in the the stack not overflowing. Typically a tail recursive function will make use of an accumulator value to allow tail call optimization to occur.

answered Jun 15 '10 at 7:46

sashang
4,946 ● 1 ● 15 ● 28

Unless I misunderstood what forward recursion means (I admit I had to google it as I had never heard the term before), neither of your functions is forward recursive. – sepp2k Jun 15 '10 at 7:50

Looks like i missed the 'forward' recursion part of the question. – sashang Jun 15 '10 at 7:58