;;;Tail Recursion

(define f

(lambda (x y)

(g(u x)))

Not much of a procedure.

Look at “machine code” for F.

* The return from g will bring us back to f2.
* How to optimise?
* Return pops the value that returns there.
* When g returns, it returns to caller f, not f.

Peephole option

1. Push address
2. Jump X
3. Address
4. Return → {Jump X}

Note that “ret” = pop address off stack and jump to it.

We can write a loop.

For example, look at fact1. What is the space requirements? Stack is o(n), it is o(1) for each stack frame holding an invocation of fact1.

facR is a different factorial function.

* It calls fact2-aux.
* Fact2-aux is recursive, 2 paameters
* If 0, calls a. If not, it calls itself with n-1 and n\*a.

Theorem

(fact2-aux n a) returns n! \* a.

*Proof by induction:*

If n is 0, a=1 which is correct.

N!\*a = (fact2-aux (-n 1)(\*n a)), assume true for values less than n.

=(n-1)!\*(n\*a)

=n\*(n-1)!\*a

=n!\*a

QED

Proof by induction on n

1. Case n =0, (fact2-aux 0 a) = a
2. Case n>0, the proof above.

Fact2-aux answer is immediately returned (more efficient). The stack doesn’t have to grow. Answer immediately returned to caller, stack frame doesn’t have to grow.

Space requirement is now o(1).

Technically tail recursion call sight.

In body of f, the call to u is not tail recursive, the call to g is tail recursive. The tail is the last bit.

Tail call in “fact2-aux” is fact2-aux. All tail calls required to use stack frame.

Note

Tail call useful in Web production

A→B→C and C→B→A. Instead use A→B→C→A…

What is a procedure call?

1. Marshall argument
2. Push return address
3. Jump (to procedure address)

But if there is a tail call, skep step b). If no args, skip step a).

Theefore, Tail call to procedure without args is GOTO.

An example is binary search.

We look for x in sorted array a[N].

In lambda calculus, all have variables in binary search.

Floor is used as in scheme, division is exact.