Lec 3 - COL765

Recap

1. Recursion - [longmult, fact, ...]
2. Correctness by induction

Today: More on recursion & induction (tail recursion)
+ little bit of Haskell syntax as well.

Recursion

- two version of Perfect numbers
- Tail recursion [start with fact, fast-pow]
- Computation styles [Eager & Lazy] → Std. computation is left to right & top to bottom
  Exception: when precedence, associativity & other such constraints are specified

Perfect number

$$n = \sum_{K=1}^{n \text{ quot } 2} i \mid \text{Divisor}(K)$$

$$i \mid \text{Divisor}(K) = \begin{cases} K & i \mid K \mid n \\ 0 & \text{otherwise} \end{cases}$$

$$\sum_{K=l}^{U} i \mid \text{Divisor}(K) = \begin{cases} 0 & i \mid l > U \\ i \mid \text{Divisor}(l) + \sum_{l+1}^{U} i \mid \text{Divisor}(K) & \text{otherwise} \end{cases}$$

```
Perfect n =
        if   n <= 0
        then    error "nonpositive"
        Else
            n =  sum_divisors (1, n `quot` 2)

Sum_divisors l u    =
                if  l > u
                then  0
                Else
                    if Divisor (l) + sum_divisors (l+1) u

if Divisor k =    if  n `rem` k == 0
                  then  k
                  Else  0
```

Tail Recursion

Motivation : To remove any inefficiencies of recursive computational process via "deferred computation"

Key idea : maintain **state** of computation at each invocation of the recursive function through the use of auxiliary variables

Collection of values of entities in a program
Eg: particles, mass-velocity-pos etc.

$$fact\ n = \begin{cases} 1 & n = 0 \\ n * fact(n-1) & n > 0 \end{cases}$$

- Auxiliary variable : C s.t.

$C_0 \leftarrow \boxed{0} \leq C \leq \boxed{n} \rightarrow C_m$

Motivation : C captures the counter upto n with one step increments

- Auxiliary variable : f s.t.

$f_0 = 1$ [o/p of fact function at Counter 0]

in general $f = f_0 * \prod_{i=C_0+1}^{C} i$ [o/p of factorial for any step C]

Idea of invariants : → Required to establish correctness

a cond^{n} on the state of computation that doesn't change throughout the computation,

usually expressed as a relationship bet^{n} state variables

For fact function

invariant $f_0 * \prod_{i=c_0+1}^{n} i = f * \prod_{i=c+1}^{n} i$

Alg. for factorial

$$\text{fact\_iter}(n, f, c) = \begin{cases} f & \text{if } c = n \\ \text{fact\_iter}(n, f*(c+1), c+1) & \text{if } 0 \le c < n \end{cases}$$

Q: Why is this style called tail-recursive?

In recursive fact

$n * \text{fact}(\ldots)$

mult.

In tail-recursive I.S. is purely recursive

fact_iter (5, 1, 0)

= fact_iter (5, 1, 1)

= " (5, 2, 2)

= " (5, 6, 3)

= " (5, 24, 4)

= " (5, 120, 5)

termination cond^

o/p of fact(5)

Proof of correctness

Basis : when $m = c$

$$\text{fact\_iter}(m, f, c) = f * \prod_{c+1}^{m} i$$

$$= f * 1 = f$$

I.H. : for some $k = m - c \geq 0$

$$\text{fact\_iter}(m, f, c) = f * \prod_{c+1}^{m} i$$

I.S. : Let $m - c = k+1 \geq 0$

then $\text{fact\_iter}(m, f, c) = \text{fact\_iter}(m, f*(c+1), c+1)$

$$= f*(c+1) * \prod_{c+2}^{m} i \quad [\text{I.H.}]$$

$$= f * \prod_{c+1}^{m} i$$

thus,

$$\text{fact\_iter}(n, 1, 0) = 1 * \prod_{i=1}^{n} i = n!$$

More tail recursion

pow_iter (x, n, f, c)] Same idea as before

$$= \begin{cases} f & \text{if } c = n \\ \text{pow\_iter}(x, n, f * x, c+1) & \text{if } c < n \end{cases}$$

fast_pow_iter $(x, n, p) =$

$$\begin{cases} p & \text{if } n = 0 \\ \text{fast\_pow\_iter}\ (x^2, n \text{ div } 2, p) & \text{if } n > 0 \ \& \text{ even} \\ \text{fast\_pow\_iter}\ (x^2, n \text{ div } 2, p \cdot x) & \text{if } n > 0 \ \& \text{ odd} \end{cases}$$

Correctness

- induct on n
- Base : $n = 0$, fast_pow_iter $(x, 0, p) = p = p \cdot x^0$
  I.H: : $\forall x$:Real, p:real $> 0 \wedge k$:int $\wedge\ 0 \le k < n$
  fast_pow_iter $(\ldots) = p \cdot x^k$
  I.S. in 2 cases