FIT2004 S1_2017 Tute Week 3 Solutions

1. $a + ar + ar^2 + ... + ar^N = \frac{a(r^{N+1} - 1)}{r-1}$

Base case: N = 0

$a = \frac{a(r^{0+1} - 1)}{r-1}$

$= \frac{a(r - 1)}{(r-1)}$

$a = a$

Assume true for a general case where N = k

$a + ar + ar^2 + ... + ar^k = \frac{a(r^{k+1} - 1)}{r-1}$

Consider N= k+1 case:

$a + ar + ar^2 + ... + ar^k + ar^{k+1} = \frac{a(r^{k+1} - 1)}{r-1} + ar^{k+1}$

$$= \frac{a(r^{k+1} - 1)}{r-1} + ar^{k+1}$$

$$= \frac{a(r^{k+1} - 1) + ar^{k+1}(r-1)}{r-1}$$

$$= \frac{ar^{k+1} - a + ar^{k+2} - ar^{k+1}}{r-1}$$

$$= \frac{ar^{k+2} - a}{r-1}$$

$$= \frac{a(r^{k+2} - 1)}{r-1}$$

Q.E.D.

3.

For this question we wish to prove that the append operation is associative; in essence that `append(L1,append(L2,L3))` yields the same as `append(append(L1,L2),L3)`

if we start with a base case whereby L1 is the null list

```
append(L1,append(L2,L3))
append(null,append(L2,L3))
append(L2,L3)
append(append(null,L2),L3)
```

now for the inductive case; here we assume it works for the tail of L1 and will hence show it works for L1

```
append(cons(h,T_L1),append(L2,L3))
```

as usual for the append operation, it transfers moves along the list element by element so we take the head out

```
cons(e,append(T_L1,append(L2,L3)))
```

given that it is assumed to work for the tail of L1 we can restate this as

```
cons(e,append(append(T_L1,L2),L3))
```

appreciating (from the definition) that appending L2 to the tail of L1 is still the same as appending L2 to L1 this becomes

```
append(cons(e,append(T_L1,L2),L3))
append(append(cons(e,T_L1),L2),L3)
```

and cons(e,T_L1) is merely the original list L1

```
append(append(L1,L2),L3)
```

and here we find that which we were hoping to prove

4a. Show that M(M(T)) = T for all trees T

`T = nilTree | fork e * tree e * tree e`

`M(nilTree) = nilTree | M(fork(e, L, R)) = fork(e, M(R), M(L))`

Note that `M(T)` mirrors the left and right subtrees when `T != nilTree`.

We're going to show that our objective is true in a trivial case, then show that it holds in the general recursive case — proof by structural induction.

Base Case

Given our rules, the rule is trivially shown to hold when `T = nilTree`.

`M(M(nilTree))`

`= M(nilTree)` by substituting M(nilTree) with nilTree

`= nilTree` by substituting m(nilTree) with nilTree

General Case

The general form for a tree is given as `T = fork(e, L, R)`, where we note that `L` and `R` are both of type `tree e` (see the above definition).

`M(M(fork(e, L, R)))`

`= M(fork(e, M(R), M(L)))` by substituting the inner M(fork(e, L, R)) with fork(e, M(R), M(L))

At this stage, it's worth restating the definition of fork given in the lecture slides to make a subtle point explicit: `fork : e * tree e * tree e —> tree e`

Given this, we know `M(R)` and `M(L)` are both of type `tree`. And, alleluia, the following substitution is possible:

`M(fork(e, M(R), M(L)))`

`= fork(e, M(M(L)), M(M(R)))` by definition of M(T)

Note that `M` has again swapped the positions of the subtrees.

Crucially, we're now going to now assume that substructures `M(M(L)) = L` and `M(M(R)) = R` and use this to show that the larger structure `M(M(T)) = T`. This is the inductive step that is *only relevant after having proved that the base case holds*.

`fork(e, M(M(L)), M(M(R)))`

`= fork(e, L, R)` given by the above assumption

`= T` by our our initial definition of T

As we've shown that both the base case is true *and* the step that moves us closer to the base case is true, we've shown that `M(M(T)) = T` for every tree `T`.

4b. Show that SUM(M(T)) = SUM(T) for all trees T

```
T = nilTree | fork e * tree e * tree e
SUM(nilTree) = 0 | SUM(fork(e, L, R) = e + SUM(L) + SUM(R)
M(nilTree) = nilTree | M(fork(e, L, R)) = fork(e, M(R), M(L))
```

As before, we're going to show that our objective is true in a trivial case, then show that it holds every time we step closer to the base case — proof by structural induction.

Base Case

Given our rules, the rule is trivially shown to hold when `T = nilTree`.

`SUM(M(nilTree))`
`= SUM(nilTree)` by substituting M(nilTree) with nilTree
`= 0`

General Case

The general form for a tree is given as `T = fork(e, L, R)`, where we note that `L` and `R` are both of type `tree e`.

`SUM(M(fork(e, L, R)))`
`= SUM(fork(e, M(R), M(L)))` by substituting `M(fork(e, L, R))` with `fork(e, M(R), M(L))`
`= e + SUM(M(R)) + SUM(M(L))` by the definition of SUM
`= e + SUM(M(L)) + SUM(M(R))` as addition is commutative

We're going to assume that substructures `SUM(M(L)) = SUM(L)` and `SUM(M(R)) = SUM(R)` and use this to show that the larger structure `SUM(M(T)) = SUM(T)`. This is the inductive step that is only relevant after having shown that the base case holds.

`e + SUM(M(L)) + SUM(M(R))`
`= e + SUM(L) + SUM(R)` by the indicative hypothesis described above
`= SUM(T)` by the definition of `SUM(T)`

As we’ve shown that both the base case is true and the step that moves us closer to the base case is true, we’ve shown that `SUM(M(T)) = SUM(T)` for every tree `T`.

5.

For a recurrence of the kind

```
         /
         |    2 T_{N1}  + a    N > 0
T_N =
         |    b                  N = 0
         \
```

the solution is `T_N= 2^N(a+b)a`

Solution:

```
T_N  =  2 (T_{N1}          )  + a
       by recursively decomposing T_{N1} = 2*T_{N2}+a, we get
     =  2 (2(T_{N2}   ) + a)  + a   = 2^2 T_{N2} + 3a
      again recursively decomposing T_{N2} = 2*T_{N3}+a, we
get
     =  2 (2(2T_{N3}+a) + a)  + a   = 2^3 T_{N3} + 7a
      we can begin to see a pattern in these decomposition
     = . . .
     = . . .
     = . . .
      after some k decompositions, we get
T_N  = 2^k T_{Nk} + (2^k 1)a

When k = N:

T_N = 2^N T_0  + (2^N 1)a
Since T_0 = b by the recurrence given above,
T_N = 2^N b    + (2^N 1)a

Rearranging
T_N = 2^N(b+a)a        Q.E.D.

 This recurrence suggests an exponential growth.
```