FIT2004: Tutorial 1 (held in Week 3)

Objectives: The tutorials, in general, give practice in problem solving, in analysis of algorithms and data-structures, and in mathematics and logic useful in the above.

Instructions to the class: Prepare your answers to the questions **before** the tutorial! It will probably not be possible to cover all questions unless the class has prepared them in advance. There is 0.5 mark worth for this Tute towards active participation. 0.25 marks is towards answering the starred questions (*) indicated below, before attending your assigned tutorial. You will have to hand in your work on these starred questions to your tutor at the very start of the tutorial. Remaining 0.25 mark is for participating during the rest of the tutorial.

Instructions to Tutors:

i. The purpose of the tutorials is not to solve the practical exercises!

ii. The purpose is to check answers, and to discuss particular sticking points, not to simply make answers available.

1. Use mathematical induction to prove the following:

$$\sum_{i=0}^{N} ar^i = a + ar + ar^2 + ar^3 + \cdots + ar^N = \frac{a\left(r^{N+1}-1\right)}{r-1}$$

Base case: N = 0

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

Inductive step

Assume it holds for a value k, i.e., $\sum_{i=0}^{k} ar^i = \frac{a(r^{k+1}-1)}{r-1}$. We show that it holds for $k+1$, i.e.,$\sum_{i=0}^{k+1} ar^i = \frac{a(r^{k+2}-1)}{r-1}$

$\sum_{i=0}^{k+1} ar^i = a + ar + ar^2 \cdots + ar^k + ar^{k+1}$

$= ar^{k+1} + \sum_{i=0}^{k} ar^i$

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

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

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

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

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

2. * Listed below is the abstract data type definition for a binary-tree data structure (similar to what's been introduced in the lecture 1.2):

```
type tree e = nilTree | fork e * (tree e) * (tree e)
```

Further, below are definitions of two algorithm on this data structure:

`M` applicable to trees of any kind, defined as:

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

`SUM` on trees of numbers (integers or floating-point):

```
SUM(nilTree) = 0
|  SUM( fork(e,L,R) ) = e +  SUM(L) + SUM(R)
```

Your task in this tutorial is prove some properties of the algorithms:

(a) Prove **formally** that M(M(T)) = T, for every tree T.

(b) Prove **formally** that SUM(M(T)) = SUM(T), for every tree T of numbers.

2a. Show that M(M(T)) = T for all trees T

$$T = \quad nilTree \quad | \quad forke*(treee)*(treee) \tag{1}$$

$$M(nilTree) = \quad nilTree \quad | \quad M(fork(e,L,R)) = fork(e,M(R),M(L)) \tag{2}$$

We are 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 is 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 are 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 have shown that both the base case is true and the step that moves us closer to the base case is true, we have shown that `M(M(T)) = T` for every tree `T`.

2b. 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 are 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 âĂŤ 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 are 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 have shown that both the base case is true and the step that moves us closer to the base case is true, we have shown that `SUM(M(T)) = SUM(T)` for every `tree T`.

3. * Solve the following recurrence relationship:

$$T(N) = \begin{cases} 2 * T(N-1) + a, & \text{if } N > 0 \\ b & \text{if } N = 0. \end{cases}$$

$$T(N) = 2T(N-1) + a \tag{3}$$

Our goal is to reduce the term on R.H.S. to $T(0)$.

$$T(N-1) = 2T(N-2) + a \tag{4}$$

Replace the value of $T(N-1)$ from the above equation in Eq. (3).

$$T(N) = 2(2T(N-2) + a) + a = 2^2 T(N-2) + 2^1 a + 2^0 a \tag{5}$$

Similarly, we have $T(N-2) = 2 * T(N-3) + a$. Replacing this value of $T(N-2)$ in the above equation gives

$$T(N) = 2^2(2T(N-3) + a) + 2^1 a + 2^0 a = 2^3 T(N-3) + 2^2 a + 2^1 a + 2^0 a \tag{6}$$

Do you see the pattern? For k, we have

$$T(N) = 2^k T(N-k) + 2^{k-1} a + 2^{k-2} a + \cdots + 2^0 a \tag{7}$$

or

$$T(N) = 2^k T(N-k) + \sum_{i=0}^{k-1} 2^i a \tag{8}$$

$$T(N) = 2^k T(N-k) + a(2^k - 1) \tag{9}$$

Setting $N-k=0$ or $k=N$

$$T(N) = 2^N T(0) + a(2^N - 1) = 2^N(b+a) - a \tag{10}$$

Hence,

$$T(N) = 2^N(b+a) - a \tag{11}$$

4. * Show that the following recurrence relation has a solution $T(N) = b + c\log_2 N$.

$$T(N) = \begin{cases} T(N/2) + c, & \text{if } N > 1 \\ b & \text{if } N = 1. \end{cases}$$

Base case: N = 1
$T(1) = b + c\log_2 1 = b$
Inductive Step
Assume that $T(k) = b + c\log_2 k$. Prove this for the next case, i.e., show $T(2k) = b + c\log_2 2k$.
$T(2k) = T(k) + c$
Since $T(k) = b + c\log_2 k$, we have
$T(2k) = b + c\log_2 k + c = b + c(\log_2 k + \log_2 2) = b + c\log_2 2k$

5. Let $F(n)$ denote the $n-th$ number in the Fibonacci series. Prove the following by induction.

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{bmatrix}$$

See `https://www.nayuki.io/page/fast-fibonacci-algorithms`

6. Let $F(n)$ denote the $n-th$ number in the Fibonacci series. Prove that we can compute $F(2k)$ and $F(2k+1)$ using $F(k)$ and $F(k+1)$ as follows.

$$F(2k) = F(k)[2F(k+1) - F(k)] \tag{12}$$

$$F(2k+1) = F(k+1)^2 + F(k)^2 \tag{13}$$

See `https://www.nayuki.io/page/fast-fibonacci-algorithms`

-=END=-