Induction, Recursion, Algorithmic Analysis

Problem 1
Prove by induction that

$$1 \cdot 1! + 2 \cdot 2! + \ldots + n \cdot n! = (n+1)! - 1 \quad \text{for } n \geq 1$$

Solution

Let $P(n)$ be the proposition that $1 \cdot 1! + 2 \cdot 2! + \ldots + n \cdot n! = (n+1)! - 1$. We will prove that $P(n)$ holds for all $n \geq 1$ by induction on n.

Base case $n = 1$. $1.1! = 1 = 2! - 1 = (1+1)! - 1$ so $P(1)$ holds.

Inductive case. Assume $P(k)$ holds for some $k \in \mathbb{N}_{>0}$. That is $1 \cdot 1! + 2 \cdot 2! + \ldots + k \cdot k! = (k+1)! - 1$. Then

$$\begin{aligned}
1 \cdot 1! + 2 \cdot 2! + \ldots + k \cdot k! + (k+1) \cdot (k+1)! &= (k+1)! - 1 + (k+1) \cdot (k+1)! \quad \text{(Induction hypothesis)} \\
&= (1 + k + 1)(k+1)! - 1 \\
&= ((k+1)+1)(k+1)! - 1
\end{aligned}$$

so $P(k+1)$ holds.
Therefore, by the Principle of Induction, $P(n)$ holds for all $n \geq 1$.

Problem 2
Let $\Sigma = \{1, 2, 3\}$.

(a) Give a recursive definition for the function $\mathsf{sum} : \Sigma^* \to \mathbb{N}$ which, when given a word over Σ returns the sum of the digits. For example $\mathsf{sum}(1232) = 8$, $\mathsf{sum}(222) = 6$, and $\mathsf{sum}(1) = 1$. You should assume $\mathsf{sum}(\lambda) = 0$.

(b) For $w \in \Sigma^*$, let $P(w)$ be the proposition that for all words $v \in \Sigma^*$, $\mathsf{sum}(wv) = \mathsf{sum}(w) + \mathsf{sum}(v)$. Prove that $P(w)$ holds for all $w \in \Sigma^*$.

(c) Consder the function $\mathsf{rev} : \Sigma^* \to \Sigma^*$ defined recursively as follows:

- $\mathsf{rev}(\lambda) = \lambda$
- For $w \in \Sigma^*$ and $a \in \Sigma$, $\mathsf{rev}(aw) = \mathsf{rev}(w)a$

Prove that for all words $w \in \Sigma^*$, $\mathsf{sum}(\mathsf{rev}(w)) = \mathsf{sum}(w)$

Solution

(a) We give a definition using the recursive nature of Σ^*:

$$\begin{aligned} \mathsf{sum}(\lambda) &= 0 \\ \mathsf{sum}(a.w) &= a + \sum(w). \end{aligned}$$

(b) We first need the recursive definition of concatenation:

$$\begin{aligned} \lambda.v &= v \\ (aw).v &= a(w.v) \end{aligned}$$

We will now prove $P(w)$ for all $w \in \Sigma^*$ by structural induction on w.

Base case ($w = \lambda$).

$$\begin{aligned} \mathsf{sum}(wv) &= \mathsf{sum}(\lambda.v) & \\ &= \mathsf{sum}(v) & \text{Definition of concatenation} \\ &= 0 + \mathsf{sum}(v) & \\ &= \mathsf{sum}(\lambda) + \mathsf{sum}(v) & \text{Definition of sum} \\ &= \mathsf{sum}(w) + \mathsf{sum}(v) & \end{aligned}$$

So $P(\lambda)$ holds.

Inductive case ($w = aw'$). Assume $P(w')$ holds, that is for all $v \in \Sigma^*$, $\mathsf{sum}(w'v) = \mathsf{sum}(w') + \mathsf{sum}(v)$. Then for all $v \in \Sigma^*$ and all $a \in \Sigma$:

$$\begin{aligned} \mathsf{sum}((aw')v) &= \mathsf{sum}(a(w'v)) & \text{Definition of concatenation} \\ &= a + \mathsf{sum}(w'v) & \text{Definition of sum} \\ &= a + \mathsf{sum}(w') + \mathsf{sum}(v) & \text{Inductive hypothesis} \\ &= \mathsf{sum}(aw') + \mathsf{sum}(v) & \text{Definition of sum} \end{aligned}$$

So $P(w')$ implies $P(aw')$ for all $a \in \Sigma$.

Therefore, by the Principle of Structural Induction, $P(w)$ holds for all $w \in \Sigma^*$.

(c) Let $P(w)$ be the proposition that $\mathsf{sum}(\mathsf{rev}(w)) = \mathsf{sum}(w)$. We will show that $P(w)$ holds for all words $w \in \Sigma^*$ by structural induction on w.

Base case ($w = \lambda$). From the definition of rev we have: $\mathsf{sum}(\mathsf{rev}(\lambda)) = \mathsf{sum}(\lambda)$. So $P(\lambda)$ holds.

Inductive case ($w = aw'$). Suppose $P(w')$ holds, that is $\mathsf{sum}(\mathsf{rev}(w')) = \mathsf{sum}(w')$. For any $a \in \Sigma$ we have:

$$\begin{aligned} \mathsf{sum}(\mathsf{rev}(aw')) &= \mathsf{sum}(w'a) & \text{Definition of rev} \\ &= \mathsf{sum}(w') + \mathsf{sum}(a) & \text{From (b)} \\ &= \mathsf{sum}(w') + a + \mathsf{sum}(\lambda) & \text{Definition of sum} \\ &= a + \mathsf{sum}(w') + 0 & \text{Definition of sum} \\ &= \mathsf{sum}(aw') & \text{Definition of sum} \end{aligned}$$

So $P(w')$ implies $P(aw')$ for all $a \in \Sigma$.

Therefore, by the Principle of Structural Induction, $P(w)$ holds for all $w \in \Sigma^*$.

Problem 3
Define $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ recursively as follows: $f(m, 0) = 0$ for all $m \in \mathbb{N}$ and $f(m, n+1) = m + f(m, n)$.

(a) Let $P(n)$ be the proposition that $f(0, n) = f(n, 0)$. Prove that $P(n)$ holds for all $n \in \mathbb{N}$.

*(b) Let $Q(m)$ be the proposition $\forall n,\ f(m, n) = f(n, m)$. Prove that $Q(m)$ holds for all $m \in \mathbb{N}$.

Solution

1. We show that $P(n)$ holds for all $n \in \mathbb{N}$ by induction.

   Base case: $n = 0$. Since $f(0, 0) = f(0, 0)$, $P(0)$ holds.

   Inductive case. Now suppose $P(n)$ holds. Then

$$\begin{aligned} f(0, n+1) &= 0 + f(0, n) && \text{(Def)} \\ &= 0 + f(n, 0) && \text{(IH)} \\ &= 0 && \text{(Def)} \\ &= f(n+1, 0). && \text{(Def)} \end{aligned}$$

   So $P(n) \to P(n+1)$, and thus $P(n)$ holds for all $n \in \mathbb{N}$.

2. We will prove by induction that $f(m, n) = mn$, from which it follows that $f(m, n) = mn = nm = f(n, m)$. Let $R(n)$ be the proposition that: for all m, $f(m, n) = mn$.

   Base case: $n = 0$. From the definition of f, $f(m, 0) = 0 = 0.m$ for all m. So $R(0)$ holds.

   Inductive case. Suppose that $R(n)$ holds. That is, for all m, $f(m, n) = mn$. Then, for all m,

$$\begin{aligned} f(m, n+1) &= m + f(m, n) && \text{Definition of } f \\ &= m + mn && \text{Induction hypothesis} \\ &= m(n+1). \end{aligned}$$

   So $R(n+1)$ holds. Thus, $R(n)$ implies $R(n+1)$, so by the Principle of Induction $f(m, n) = mn$ for all m and n. Therefore $f(m, n) = f(n, m)$.

Problem 4
Analyse the complexity of the following algorithms to compute the n-th Fibonacci number

(a) **FibOne**(n):

   if $n \leq 2$ then return 1

   else return **FibOne**$(n-1)$ + **FibOne**$(n-2)$

(b) **FibTwo**(n):

$x = 1, y = 0, i = 1$

While $i < n$:

$t = x$

$x = x + y$

$y = t$

$i = i + 1$

return x

Solution

(a) Let $T(n)$ be the running time of **FibOne**(n). Then in the worst case, there are two recursive calls to smaller instances of **FibOne**, taking time $T(n-1)$ and $T(n-2)$ respectively. All other operations are constant time, so

$$\begin{aligned} T(n) &= O(1) + T(n-1) + T(n-2) \\ &\leq O(1) + 2.T(n-1). \end{aligned}$$

From the lectures, this means that $T(n) \in O(2^n)$.

(b) Let $T(n)$ be the running time of **FibTwo**(n). We have a while-loop which runs $O(n)$ times, and within the while loop there are several operations taking $O(1)$ time. All other operations are constant time, so the overall running time is $O(1) + O(n) \times O(1) = O(n)$.

Discussion

NB: It is possible to obtain better bounds for **FibOne**, however because of the $O(1)$ that appears in the recurrence equation, it is not quite as simple as $T(n) = \textsc{Fib}(n)$. A bound of $O(2^n)$ demonstrates a reasonable level of understanding, so would be sufficient in most assessable tasks.

Problem 5

Analyse the complexity of the following recursive algorithm to test whether a number x occurs in an *ordered* list $L = [x_1, x_2, \ldots, x_n]$ of size n. Take the cost to be the number of list element comparison operations.

BinarySearch$(x, L = [x_1, x_2, \ldots, x_n])$:

if $n = 0$ then return no

else

if $x_{\lceil \frac{n}{2} \rceil} > x$ then return **BinarySearch**$(x, [x_1, \ldots, x_{\lceil \frac{n}{2} \rceil - 1}])$

else if $x_{\lceil \frac{n}{2} \rceil} < x$ return **BinarySearch**$(x, [x_{\lceil \frac{n}{2} \rceil + 1}, \ldots, x_n])$

else return yes

Solution

Let $T(n)$ be the cost of running **BinarySearch** on a list of length n. In the worst case, we make $2 = O(1)$ element comparisons and recursively call **BinarySearch** on a list of length $\lceil \frac{n}{2} \rceil$. So we have:

$$T(n) = O(1) + T(n/2).$$

The Master Theorem applies to this recurrence: we have $a = 1$, $b = 2$, $c = 0$ and $d = \log_b(a) = 0$, so we are in Case 2. This tells us that $T(n) \in O(n^d \log n) = O(\log n)$.