Recursive Definitions and Structural Induction

Section 5.3

Recursion

Recursion – defining an object (or function, algorithm, etc.) in terms of itself.

Recursion can be used to define sequences

- Previously sequences were defined using a specific formula, e.g., \( a_n = 2^n \) for \( n = 0,1,2,... \)

- This sequence can also be defined by giving the first term of the sequence, namely \( a_0 = 1 \), and a rule for finding a term of the sequence for the previous one, namely, \( a_{n+1} = 2a_n \) for \( n = 0,1,2,... \)
Recursively Defined Functions

- A recursive or inductive definition of a function consists of two steps.
  - BASIS STEP: Specify the value of the function at zero.
  - RECURSIVE STEP: Give a rule for finding its value at an integer from its values at smaller integers.

- A function $f(n)$ is the same as a sequence $a_0, a_1, \ldots$, where $a_i$, where $f(i) = a_i$. This was done using recurrence relations in Section 2.4.

Fibonacci Numbers

- Example: The Fibonacci numbers are defined as follows:
  
  \[ f_0 = 0 \]
  \[ f_1 = 1 \]
  \[ f_n = f_{n-1} + f_{n-2} \]

  Find $f_2, f_3, f_4, f_5$.

- Solution:
  
  \[ f_2 = f_1 + f_0 = 1 + 0 = 1 \]
  \[ f_3 = f_2 + f_1 = 1 + 1 = 2 \]
  \[ f_4 = f_3 + f_2 = 2 + 1 = 3 \]
  \[ f_5 = f_4 + f_3 = 3 + 2 = 5 \]
Recursively Defined Functions

- Example: Give a recursive definition of the factorial function \( n! \) (\( n \) is a nonnegative integer):

- Solution:
  \[
  f(0) = 1 \\
  f(n + 1) = (n + 1) \cdot f(n)
  \]

Recursively Defined Sets and Structures

- Recursive definitions of sets have two parts:
  - **basis step**: specifies an initial collection of elements.
  - **recursive step**: gives the rules for forming new elements in the set from those already known to be in the set.

- Sometimes the recursive definition has an **exclusion rule**, which specifies that the set contains nothing other than those elements specified in the basis step and generated by applications of the rules in the recursive step.

- Assume the exclusion rule holds, even if it is not explicitly mentioned.

- **Structural induction** is used to prove results about recursively defined sets.
Recursively Defined Sets and Structures

_example: Subset of Integers S:
   BASIS STEP: 3 ∈ S.
   RECURSIVE STEP: If x ∈ S and y ∈ S, then x + y is in S.
   Initially 3 is in S, then 3 + 3 = 6, then 3 + 6 = 9, etc. All positive multiples of 3.
_example:
   BASIS STEP: 0 ∈ N.
   RECURSIVE STEP: If n is in N, then n + 1 is in N.
_solution:
The natural numbers N. Initially 0 is in S, then 0 + 1 = 1, then 1 + 1 = 2, etc.

---

Recursively Defined Sets and Structures

_example: Give a recursive definition of
   The set of even positive integers.
   BASIS STEP: 2 ∈ S.
   RECURSIVE STEP: If x ∈ S, then x + 2 is in S.
   The set of positive integer powers of 2
   BASIS STEP: 2 ∈ S.
   RECURSIVE STEP: If x ∈ S, then x * 2 is in S.
Strings

Definition 1: The set $\Sigma^*$ of strings over the alphabet $\Sigma$:

- **BASIS STEP:** $\lambda \in \Sigma^*$ ($\lambda$ is the empty string)
- **RECURSIVE STEP:** If $w$ is in $\Sigma^*$ and $x$ is in $\Sigma$, then $wx \in \Sigma^*$.

Example: $\Sigma = \{0,1\}$,

- the strings in $\Sigma^*$ are the set of all bit strings, $\lambda, 0, 1, 00, 01, 10, 11$, etc.

Example: If $\Sigma = \{a,b\}$, show that $aab$ is in $\Sigma^*$.

- Since $\lambda \in \Sigma^*$ and $a \in \Sigma$, $a \in \Sigma^*$.
- Since $a \in \Sigma^*$ and $a \in \Sigma$, $aa \in \Sigma^*$.
- Since $aa \in \Sigma^*$ and $b \in \Sigma$, $aab \in \Sigma^*$.

Balanced Parentheses

Example: Give a recursive definition of the set of balanced parentheses $P$.

Solution:

- **BASIS STEP:** () $\in P$
- **RECURSIVE STEP:** If $w \in P$, then () $\in P$, (w) $\in P$ and $w () \in P$.

Show that (()()) is in $P$. 
Recursive Algorithms

Section 5.4

Definition 1: An algorithm is called recursive if it solves a problem by reducing it to an instance of the same problem with smaller input.

For the algorithm to terminate, the instance of the problem must eventually be reduced to some initial case for which the solution is known.
Recursive Factorial Algorithm

Example: Give a recursive algorithm for computing \( n! \), where \( n \) is a nonnegative integer.

Solution: Use the recursive definition of the factorial function.

```plaintext
procedure factorial(n: nonnegative integer)
if    n = 0 then factorial(n):=1
else factorial(n):=n*factorial(n-1)
```

Proving Recursive Algorithms Correct

Both mathematical and strong induction are useful techniques to show that recursive algorithms always produce the correct output.

Example: Prove that the algorithm for computing the powers of real numbers is correct.

```plaintext
procedure power(a: nonzero real number, n: nonnegative integer)
if n = 0 then return 1
else return a*power(a, n-1)
{output is \( a^n \)}
```

Solution: Use mathematical induction on the exponent \( n \).

- BASIS STEP: \( a^0 = 1 \) for every nonzero real number \( a \), and \( power(a,0) = 1 \).
- INDUCTIVE STEP: The inductive hypothesis is that \( power(a,k) = a^k \), for all \( a \neq 0 \). Assuming the inductive hypothesis, the algorithm correctly computes \( a^{k+1} \), since

\[
power(a,k + 1) = a \cdot power(a, k) = a \cdot a^k = a^{k+1}.
\]
**Merge Sort**

- *Merge Sort* works by iteratively splitting a list (with an even number of elements) into two sublists of equal length until each sublist has one element.
- Each sublist is represented by a balanced binary tree.
- At each step a pair of sublists is successively merged into a list with the elements in increasing order. The process ends when all the sublists have been merged.
- The succession of merged lists is represented by a binary tree.

**Example:** Use merge sort to put the list 8,2,4,6,9,7,10,1,5,3 into increasing order.

**Solution:**
Recursive Merge Sort

Example: Construct a recursive merge sort algorithm.

Solution: Begin with the list of $n$ elements $L$.

```plaintext
procedure mergesort($L = a_1, a_2, \ldots, a_n$)
if $n > 1$ then
    $m := \lfloor n/2 \rfloor$
    $L_1 := a_1, a_2, \ldots, a_m$
    $L_2 := a_{m+1}, a_{m+2}, \ldots, a_n$
    $L := \text{merge(mergesort($L_1$), mergesort($L_2$))}$
{L is now sorted into elements in increasing order}

procedure merge($L_1, L_2$: sorted lists)
$L := \text{empty list}$
while $L_1$ and $L_2$ are both nonempty
    remove smaller of first elements of $L_1$ and $L_2$ from its list;
    put at the right end of $L$
if this removal makes one list empty
    then remove all elements from the other list and append them to $L$
return $L$. {L is the merged list with the elements in increasing order}
```

Merging Two Lists

Example: Merge the two lists 2, 3, 5, 6 and 1, 4.

Solution:

| Table 1 Merging the Two Sorted Lists 2, 3, 5, 6 and 1, 4. |
|----------------|----------------|-------------|-------------|
| First List     | Second List    | Merged List | Comparison  |
| 2 3 5 6         | 1 4            | 1           | 1 < 2       |
| 2 3 5 6         | 4              | 1 2         | 2 < 4       |
| 3 5 6           | 4              | 1 2 3       | 3 < 4       |
| 5 6             | 4              | 1 2 3 4     | 4 < 5       |
| 5 6             | 1 2 3 4 5 6    |             |             |