Recursive Definitions and Structural Induction

Introduction

If it is difficult to define an object explicitly, it may be easy to define this object in terms of itself (i.e., the current term could be given in terms of the previous terms). This process is called recursion.

Recursion can be used to define sequences, functions and sets.

Recursively Defined Functions

Let \( f \) be a function with the domain \( \mathbb{Z}^+ = \mathbb{Z} \setminus \{-1, -2, \ldots\} \). To define the function \( f \) we use two steps:

**BASIS STEP:** Specify the value of the function at zero, \( f(0) = ? \)

**RECURSIVE STEP:** Give a rule for finding the function value \( f(n) \) from \( f(0), f(1), \ldots, f(n-1) \).

Such a definition is called recursive or inductive definition.

**Example 1** If \( f \) is defined recursively by \( f(0) = 3, f(n+1) = 2f(n) + 3 \), find \( f(1), f(2), f(3), f(4) \).

**Solution:** from the definition

\[
\begin{align*}
  f(1) &= 2f(0) + 3 = 2\cdot 3 + 3 = 9, \\
  f(2) &= 2f(1) + 3 = 2\cdot 9 + 3 = 21, \\
  f(3) &= 2f(2) + 3 = 2\cdot 21 + 3 = 45, \\
  f(4) &= 2f(3) + 3 = 2\cdot 45 + 3 = 93.
\end{align*}
\]

**Example 2** Give an inductive/recursive definition of \( F(n) = n! \).

**Solution:** we use the two steps:

**BASIS STEP:** Specify the value of the function at zero, \( F(0) = 0! = 1 \).

**RECURSIVE STEP:** Give a rule for finding the function value \( F(n) \) from \( F(0), F(1), \ldots, F(n-1) \). The rule is \( F(n+1) = (n+1)F(n) \).
Example 3 Give a recursive definition of $a^n$, $a \in R$ and $a \neq 0$, $n \in Z^+$.

Solution: we use the two steps:
BASIS STEP: Specify the value of the function at zero, $F(0) = a^0 = 1$.
RECURSIVE STEP: Give a rule for finding the function value $F(n)$ from $F(0), F(1), \ldots, F(n-1)$. The rule is $F(n+1) = aF(n)$.

Example 4 Give a recursive definition of $\sum_{k=0}^{n} a^k$.

Solution: we use the two steps:
BASIS STEP: Specify the value of the function at zero, $S(0) = a^0$.
RECURSIVE STEP: Give a rule for finding the function value $S(n)$ from $S(0), S(1), \ldots, S(n-1)$. The rule is $S(n+1) = S(n) + a^{n+1}$.

The Fibonacci numbers $f_0, f_1, f_2, \ldots$, are defined by the equations $f_0 = 0, f(1) = 1$ and $f_n = f_{n-1} + f_{n-2}$ for $n = 2, 3, 4, \ldots$.

Example 5 Find the Fibonacci numbers $f_2, f_3, f_4, f_5$ and $f_6$.

Solution: from the recursive step we get

\begin{align*}
  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, \\
  f_6 &= f_5 + f_4 = 5 + 3 = 8.
\end{align*}

Example 6 Show that $f_n > \alpha^{n-2}$ whenever $n \geq 3$, where $\alpha = [1 + \sqrt(5)]/2$.

Solution: We use strong induction to prove this inequality. Let $P(n)$: $f_n > \alpha^{n-2}$. We have to show that $P(n) = \text{T}$ whenever $n \geq 3$.
BASIS STEP:

since $f_3 = 2 > \alpha$, we have $\alpha^2 = [3 + \sqrt(5)]/2 < 3 = f_4$. So $P(3)$ and $P(4)$ are true.
INDUCTIVE STEP:
assume that \( P(j) = T \), that is \( f_j > \alpha^{j-2} \) for all \( j \) with \( 3 \leq j \leq k, k \geq 4 \). We have to show that \( P(k+1) = T \), that is \( f_{k+1} > \alpha^{k-1} \). Since \( \alpha \) is a solution of \( x^2 - x - 1 = 0 \), we have \( \alpha^2 = \alpha + 1 \). So \( \alpha^{k-1} = \alpha^2 \alpha^{k-3} = (\alpha + 1) \alpha^{k-3} = \alpha^{k-2} + \alpha^{k-3} \). By the inductive hypothesis, for \( k \geq 4 \), we have \( f_{k-1} > \alpha^{k-3}, f_k > \alpha^{k-2} \). Thus we have \( f_{k+1} = f_k + f_{k-1} > \alpha^{k-2} + \alpha^{k-3} = \alpha^{k-1} \). So \( P(k+1) = T \).

Recursively Defined Sets and Structures

Functions and also sets can be defined recursively.
In the basis step an initial collection of elements is specified.
In the recursive step, rules for forming new elements from the old ones (those already known to be in the set) are given.

Example 7 Consider the subset \( S \subset \mathbb{Z} \) defined by
BASIS STEP: \( 3 \in S \).
INDUCEIVE STEP: If \( x \in S \) and \( y \in S \), then \( x + y \in S \).

The new elements in \( S \) are, 3 by the basis step and \( 3 + 3 = 6, 6 + 3 = 3 + 6 = 9 \) are first and second application of the recursive step.

Let \( \Sigma \) be an alphabet (set of finite number of symbols) and let \( \Sigma^* \) be the set of strings over \( \Sigma \).

Definition 2

The set \( \Sigma^* \) of strings over the alphabet \( \Sigma \) can be defined recursively by
BASIS STEP: \( \lambda \in \Sigma^* \) where \( \lambda \) is the empty string containing no symbols.
RECURSIVE STEP: If \( w \in \Sigma^* \) and \( x \in \Sigma \), then \( wx \in \Sigma^* \).

Example 8 If \( \Sigma = \{0, 1\} \), then after the first application of the recursive step, the elements 0, 1 of \( \Sigma^* \) are formed. By the application of the recursive step once more, we get 00, 01, 10, 11 as the elements of \( \Sigma^* \) and so on.
Definition 3

Two strings can be combined by the operation of *concatenation*. Let $\Sigma$ be a set of symbols and $\Sigma^*$ the set of strings formed from symbols in $\Sigma$. We can define the concatenation of two symbols recursively as follows.

**BASIS STEP**: If $w \in \Sigma^*$, then $w \cdot \lambda = w$, where $\lambda$ is the empty string.

**RECURSIVE STEP**: If $w_1 \in \Sigma^*$, $w_2 \in \Sigma^*$, and $x \in \Sigma$, then $w_1(w_2x) = (w_1w_2)x$.

Example 9 Length of a String  
Give a recursive definition of $l(w)$, the length of the string $w$.

**Solution**: The length of a string can be defined by

\[
l(\lambda) = 0, \quad \text{if } \lambda \text{ is empty string};
\]

\[
l(wx) = l(w) + 1, \quad \text{if } w \in \Sigma^*, x \in \Sigma.
\]

A tree is a special type of a graph; a graph is made up of vertices and edges connecting some pairs of vertices. They can be defined recursively.

Definition 4

The set of *rooted trees*, where a *rooted tree consisted of* a set of vertices containing a distinguished vertex called the *root*, and *edges* connecting these vertices, can be defined recursively by these steps:

**BASIS STEP**: A single vertex is a rooted tree.

**RECURSIVE STEP**: If $T_1, T_2, \ldots, T_n$ are disjoint rooted trees with roots $r_1, r_2, \ldots, r_n$, respectively, then the graph formed by starting with a root $r$, which is not in any of the rooted trees $T_1, T_2, \ldots, T_n$, and adding an edge from $r$ to each of the vertices $r_1, r_2, \ldots, r_n$ is also a rooted tree.

See Figure 2. Building rooted trees (basis step, recursive step applied twice)
Binary trees are a special type of rooted trees. Two types of binary trees—full binary trees and extended binary trees can be given by recursive definition.

**Definition 5**

The set of **extended binary trees**, can be defined recursively by these steps:

**BASIS STEP**: The empty set is an extended binary tree.

**RECURSIVE STEP**: If $T_1$ and $T_2$ are disjoint extended binary trees, there is an extended binary tree, denoted by $T_1.T_2$, consisting of a root $r$ together with edges connecting the root to each of the roots of the left subtree $T_1$ and the right subtree $T_2$ when these trees are nonempty.

See Figure 3. Building up extended binary trees (basis step, recursive step applied three times)

**Definition 6**

The set of **full binary trees**, can be defined recursively by these steps:

**BASIS STEP**: The is a full binary tree consisting only of a single vertex $r$.

**RECURSIVE STEP**: If $T_1$ and $T_2$ are disjoint full binary trees, there is a full binary tree, denoted by $T_1.T_2$, consisting of a root $r$ together with edges connecting the root to each of the roots of the left subtree $T_1$ and the right subtree $T_2$.

See Figure 4. Building up full binary trees (basis step, recursive step applied twice)

**Structural Induction**

To prove results about recursively defined sets, one can use some of mathematical induction and of course one can use a more convenient form of induction known as **structural induction**.
**BASIS STEP:** Show that the result holds for all elements of specified in the basis step of the recursive definition to be in the set.

**RECURSIVESTEP:** Show that if the statement is true for each of the elements used to construct new elements in the recursive step of the definition, the result holds for these new elements.

**Example 14** Use structural induction to prove that \( l(xy) = l(x) + l(y) \), where \( x, y \in \Sigma^* \), the set of strings over the alphabet \( \Sigma \).

**Solution:** Let \( P(y) : l(xy) = l(x) + l(y) \), whenever \( x \in \Sigma^* \).

We know that \( l(\lambda) = 0 \), when \( \lambda \) is an empty string in \( \Sigma^* \) and \( l(wx) = l(w) + 1 \) when \( w \in \Sigma^* \), \( x \in \Sigma \).

**BASIS STEP:** we have to show that \( P(\lambda) = T \), i.e., we must show that \( l(x\lambda) = l(x) + l(\lambda) \) for all \( x \in \Sigma^* \). Because \( l(x\lambda) = l(x) = l(x) + 0 = l(x) + l(\lambda) \) for every string \( x \), we see that \( P(\lambda) = T \).

**RECURSIVESTEP:** we assume that \( P(y) = T \) and show that this implies that \( P(ya) = T \) if \( a \in \Sigma \). We have to show that \( l(xya) = l(x) + l(ya) \) for every \( a \in \Sigma \).

By the recursive definition of \( l(w) \) we have \( l(xya) = l(xy) + 1 \) and \( l(ya) = l(y) + 1 \) and by the inductive hypothesis \( l(xy) = l(x) + l(y) \). This shows that \( l(xya) = l(x) + l(y) + 1 = l(x) + l(ya) \).

**Definition 7**

We define the *height* \( h(T) \) of a full binary tree \( T \) recursively:

**BASIS STEP:** The height of a full binary tree \( T \) consisting of only a root \( r \) is \( h(T) = 0 \).

**RECURSIVESTEP:** If \( T1 \) and \( T2 \) are full binary trees, then the full binary tree, \( T = T1.T2 \), has the height \( h(T) = 1 + \max(h(T1), h(T2)) \).

**Theorem 2** If \( T \) is a full binary tree \( T \), then \( n(T) \leq 2^{h(T)+1} - 1 \).
Chapter 7    Advanced Counting Techniques

7.1  Recurrence Relations

We have seen that a recursive definition of a sequence specifies one or more initial terms and a rule for determining subsequent terms from those that precede them. A rule similar to this is called a recurrence relation.

Definition 1

A recurrence relation for the sequence \( \{a_n\} \) is an equation that expresses one or more of the previous terms of the sequence. A sequence called a solution of a recurrence if it satisfies the recurrence relation.

Example 1 Let \( \{a_n\} \) be a sequence that satisfies the recurrence relation 
\[
a_n = a_{n-1} - a_{n-2},
\]
for \( n=2,3,4, \ldots \), and suppose that \( a_0 = 3 \) and \( a_1 = 5 \). What are \( a_2 \) and \( a_3 \)?

Solution: Clearly \( a_2 = a_1 - a_0 = 5 - 3 = 2 \), \( a_3 = a_2 - a_1 = 2 - 5 = -3 \) and so on.

Modeling with Recurrence Relations

Example 3 Suppose that a person deposits $10,000 in a savings account at a bank yielding 11% per year with interest compounded annually. How much will be in the account after 30 years?

Solution: Let \( P_n \): the amount in the account after \( n \) years. Then \( P_0 = 10,000 \); 
\( P_1 = (1.11)P_0 \), \( P_2 = 1.11P_1 \), \ldots, \( P_n = 1.11P(n-1) \). That is 
\[
P_n = (1.11)^n P_0.
\]
We can prove the validity of this formula.

So \( n = 30 \), \( P_0 = $10,000 \) by the formula gives us 
\[
P_{30} = (1.11)^{30} \times 10,000 = $228,922.97.
\]

Example 4 Rabbits and Fibonacci Numbers A young pair of rabbits is placed on an island. A pair of rabbits does not breed until they are 2 month old. After they are 2 month old, each pair of rabbits produces another pair
each month. Find a recurrence relation for the number of pairs of rabbits on the island after n month, assuming that no rabbits ever die.

Solution: Let fn be the number of pairs of rabbits after n months. We will show that fn, for n=1, 2, 3, ..., are the terms of the Fibonacci sequence.

At the end of first month, the number of pairs of rabbits f1=1. At the end of second month, the number is still f2=1, because no breeding. From the third month on, after n months, the number of pairs fn is then the number of the pairs of the previous month f(n-1) plus the number of newborn pairs f(n-2), because each new born pair comes from a pair at least 2 months old. So

\[ f_n = f_{n-1} + f_{n-2} \quad \text{for } n \geq 3. \]

This is of course the Fibonacci sequence.

Example 5 The Tower of Hanoi
There is 3 pegs mounted on board together with disks of different sizes. Initially these disks are placed on the first peg in order of size, with the largest on the bottom. The disks are allowed to be moved one at a time from one peg to another as long as a disk is never placed on top of a smaller disk. The goal is to have all the disks on the second peg in order of size, with the largest on the bottom. See Figure 2

Let Hn be the number of moves needed to solve the problem with n disks.

Set up a recurrence relation for the sequence \( \{H_n\} \).

Solution: We can transfer the top n-1 disks, by the rules, to peg 3 using H(n-1) moves. See Figure 3. We did not move the largest disk. Now we move the largest disk to the second peg, this one move. We can transfer n-1 disks on peg 3 to peg 2 using additional H(n-1) moves, placing them on top of the largest disk, which stays fixed on the bottom of peg 2. This shows that

\[ H_n = 2H_{n-1} + 1. \]

The initial condition is H1=1. Moving one disk needs one step. For the rest

\[ H_n = 2(2H_{n-2} + 1) + 1 = 2H_{n-2} + 2 + 1. \]
\[ H_n = 2^2(2H_{n-3} + 1) + 2 + 1 = 2^3H_{n-3} + 2^2 + 2 + 1. \]

\[ H_n = 2^{n-1} + 2^{n-2} + \cdots + 2^2 + 2 + 1. \]
\[ H_n = 2^n - 1. \]