5.3 Recursive Definitions

Sometimes it is difficult to define a function or object explicitly; it is easier to define the function or object in terms of the function or object itself. This process is called recursion.

Example 1. The Fibonacci Numbers \(\{F_n\}\), defined by
\[
F_1 = 1, \quad F_2 = 1, \quad F_n = F_{n-1} + F_{n-2}.
\]

Recurrsively Defined Functions

When we define a sequence recursively by specifying how terms of the sequence are found from previous terms, we can use induction to prove results about the sequence. Note that a sequence is basically a function on \(\mathbb{N}\).

Definition 1. A recursively defined function \(f\) with domain \(\mathbb{N}\) is a function defined by:

1. BASIS STEP: Specify the value of the function at zero.
2. RECURSIVE STEP: Give a rule for finding its value at an integer from its values at smaller integers.

Such a definition is called a recursive or inductive definition.

Example 2. Give a recursive definition of the sequence \(\{a_n\}\), \(n = 1, 2, 3, \ldots\), if

1. \(a_n = 4n\)
2. \(a_n = 4^n\)
3. \(a_n = 4\)

Solution. There may be more than one correct answer to each sequence. Following are examples for solutions.

1. Since each term in the sequence is 4 greater than the previous term, we may define the sequence by setting \(a_1 = 4\) and declaring that \(a_{n+1} = 4 + a_n\) for all \(n \geq 1\).
2. Each term is 4 times its predecessor. Thus we have \(a_1 = 4\) and \(a_{n+1} = 4a_n\) for all \(n \geq 1\).
3. We may just set \(a_1 = 4\) and declare that \(a_{n+1} = a_n\) for all \(n \geq 1\).

Example 3. Is the following a valid recursively defined function with domain \(\mathbb{N}\)?

1. \(f(0) = 0, f(n) = 2f(n-2)\) for \(n \geq 1\).
2. \(f(0) = 1, f(1) = 2, f(n) = 2f(n-2)\) for \(n \geq 2\).

Solution. 1. This is not valid, since if \(n = 1\), we would have \(f(1) = 2f(-1)\), but \(f(-1)\) is not defined.

2. The basis conditions specify \(f(0)\) and \(f(1)\), and the recursive step gives \(f(n)\) in terms of \(f(n-2)\) for \(n \geq 2\), so this is a valid definition. We may find a formula for \(f(n)\) when \(n\) is a nonnegative integer: The sequence of function values is 1, 2, 4, 4, 8, 8, 16, 16, ... And we can fit a formula to this if we use the floor function: \(f(n) = 2\left\lfloor \frac{n+1}{2} \right\rfloor\). For a proof, we check the base cases: \(f(0) = 1 = 2\left\lfloor \frac{0+1}{2} \right\rfloor\) and \(f(1) = 2 = 2\left\lfloor \frac{1+1}{2} \right\rfloor\). For the inductive step: \(f(k + 1) = 2f(k - 1) = 2 \cdot 2^{\left\lfloor \frac{k}{2} \right\rfloor} = 2^{\left\lfloor \frac{k+1}{2} \right\rfloor+1} = 2^{\left\lfloor \frac{(k+1)+1}{2} \right\rfloor}\).}

Example 4. Show that whenever \(n \geq 3\), \(F_n > \alpha^{n-2}\), where \(F_n\) is the \(n\)th Fibonacci number and \(\alpha = \frac{1 + \sqrt{5}}{2}\).

Proof. We can use strong induction to prove this inequality. Let \(P(n)\) be the statement \(F_n > \alpha^{n-2}\). We want to show that \(P(n)\) is true whenever \(n\) is an integer greater than or equal to 3.
1. BASIS STEP:

\[ \alpha < 2 = F_3, \quad \alpha^2 = \frac{3 + \sqrt{5}}{2} < 3 = F_4 \]

so \( P(3) \) and \( P(4) \) are true.

2. INDUCTIVE STEP: Assume that \( P(j) \) is true, namely, that \( F_j > \alpha^{j-2} \), for all integers \( j \) with \( 3 \leq j \leq k \), where \( k \geq 4 \). We must show that \( P(k+1) \) is true, that is, that \( F_{k+1} > \alpha^{k-1} \). Because \( \alpha \) is a solution of \( x^2 - x - 1 = 0 \) (as the quadratic formula shows), it follows that \( \alpha^2 = \alpha + 1 \). Therefore,

\[ \alpha^{k-1} = \alpha^2 \cdot \alpha^{k-3} = (\alpha + 1)\alpha^{k-3} = \alpha \cdot \alpha^{k-3} + 1 \cdot \alpha^{k-3} = \alpha^{k-2} + \alpha^{k-3}. \]

By the inductive hypothesis, because \( k \geq 4 \), we have

\[ F_{k-1} > \alpha^{k-3}, F_k > \alpha^{k-2}. \]

Therefore, it follows that

\[ F_{k+1} = F_k + F_{k-1} > \alpha^{k-2} + \alpha^{k-3} = \alpha^{k-1}. \]

Hence, \( P(k+1) \) is true.

This completes the proof. \( \square \)

Recursively Defined Sets

We may define sets recursively. When we define a set recursively, we specify some initial elements in a basis step and provide a rule for constructing new elements from those we already have in the recursive step.

**Example 5.** Consider the subset \( S \) of the set of integers recursively defined by

1. BASIS STEP: \( 4 \in S \).
2. RECURSIVE STEP: If \( x \in S \) and \( y \in S \), then \( x + y \in S \).

The new elements found to be in \( S \) are 4 by the basis step, 4 + 4 = 8 at the first application of the recursive step, 4 + 8 = 12 = 8 + 4 and 8 + 8 = 16 at the second application of the recursive step, and so on. \( S \) is the set of all positive multiples of 4.

Recursive Algorithms

We may also define algorithms recursively.

**Definition 2.** An algorithm is a finite sequence of precise instructions for performing a computation or for solving a problem.

**Definition 3.** An algorithm is called recursive if it solves a problem by reducing it to an instance of the same problem with smaller input.

**Example 6.** Give a recursive algorithm for computing \( n! \), where \( n \) is a nonnegative integer.

**Solution.** We can build a recursive algorithm that finds \( n! \), where \( n \) is a nonnegative integer, based on the recursive definition of \( n! \), which specifies that \( n! = n \cdot (n-1)! \) when \( n \) is a positive integer, and that \( 0! = 1 \). To find \( n! \) for a particular integer, we use the recursive step \( n \) times, each time replacing a value of the factorial function with the value of the factorial function at the next smaller integer. At this last step, we insert the value of \( 0! \). The recursive algorithm we obtain is displayed as Algorithm 1.

To help understand how this algorithm works, we trace the steps used by the algorithm to compute \( 4! \). First, we use the recursive step to write \( 4! = 4 \cdot 3! \). We then use the recursive step repeatedly to write
3! = 3 · 2!, 2! = 2 · 1!, and 1! = 1 · 0!. Inserting the value of 0! = 1, and working back through the steps, we see that 1! = 1 · 1 = 1, 2! = 2 · 1! = 2, 3! = 3 · 2! = 3 · 2 = 6, and 4! = 4 · 3! = 4 · 6 = 24.

**ALGORITHM 1:** A Recursive Algorithm for Computing n!.

**Example 7.** Give a recursive algorithm for computing $a^n$, where $a$ is a nonzero real number and $n$ is a nonnegative integer.

**Solution.** We can base a recursive algorithm on the recursive definition of $a^n$. This definition states that $a^{n+1} = a \cdot a^n$ for $n > 0$ and the initial condition $a^0 = 1$. To find $a^n$, successively use the recursive step to reduce the exponent until it becomes zero. We give this procedure in Algorithm 2.

**ALGORITHM 2** A Recursive Algorithm for Computing $a^n$.

**Algorithm 1:** A Recursive Algorithm for Computing $n!$.

```
procedure factorial(n: nonnegative integer)
if n = 0 then return 1
else return n · factorial(n − 1)
{output is n!}
```

**Algorithm 2:** A Recursive Algorithm for Computing $a^n$.

```
procedure power(a: nonzero real number, n: nonnegative integer)
if n = 0 then return 1
else return a · power(a, n − 1)
{output is $a^n$}
```

Mathematical induction, and its variant strong induction, can be used to prove that a recursive algorithm is correct, that is, that it produces the desired output for all possible input values.

**Example 8.** Prove that Algorithm 2, which computes powers of real numbers, is correct.

**Solution.** We use mathematical induction on the exponent $n$.

1. **BASIS STEP:** If $n = 0$, the first step of the algorithm tells us that power $(a, 0) = 1$. This is correct because $a^0 = 1$ for every nonzero real number $a$. This completes the basis step.

2. **INDUCTIVE STEP:** The inductive hypothesis is the statement that power $(a, k) = a^k$ for all $a \neq 0$ for an arbitrary nonnegative integer $k$. That is, the inductive hypothesis is the statement that the algorithm correctly computes $a^k$. To complete the inductive step, we show that if the inductive hypothesis is true, then the algorithm correctly computes $a^{k+1}$. Because $k + 1$ is a positive integer, when the algorithm computes $a^{k+1}$, the algorithm sets power $(a, k + 1) = a \cdot power(a, k)$. By the inductive hypothesis, we have power $(a, k) = a^k$, so power $(a, k + 1) = a \cdot power(a, k) = a \cdot a^k = a^{k+1}$. This completes the inductive step.

We have completed the basis step and the inductive step, so we can conclude that Algorithm 2 always computes $a^n$ correctly when $a \neq 0$ and $n$ is a nonnegative integer.