CHAPTER 2

Section 2.4

Recursive Definitions

- A **recursive definition** is one in which the item being defined is included as part of the definition
  - Also called an **inductive definition**
- 2 parts to a recursive definition
  1. A basis, where some simple cases (1 or more) of the item being defined are explicitly given
  2. A recursive or inductive step, where new cases of the item being defined are given in terms of previous cases

Recursively Defined Sequences

- Sequence S
  - A list of objects that are enumerated in some order
  - $S(k)$ denotes the k^{th} object in the sequence
- Defining a sequence recursively
  - First, name one or more base cases
  - Then, define later values in terms of earlier ones
- Example 29: Sequence S is defined recursively by

1. $S(1) = 2$
2. $S(n) = 2S(n\text{-}1) \quad \text{for } n \geq 2$

Recursively Defined Sequences

- Fibonacci Sequence of Numbers

  $F(1) = 1$

  $F(2) = 1$

  $F(n) = F(n\text{-}2) + F(n\text{-}1)$ for $n > 2$

- What is the basis?
- Write the first 10 values of the sequence.

Fibonacci Numbers

- Problem 14: Prove the given property of the Fibonacci numbers directly from the definition.

  $F(n+3) = 2F(n+1) + F(n)$ for $n \geq 1$

- Recall that the definition (recursive part) is

  $F(k) = F(k\text{-}2) + F(k\text{-}1)$

- Since we're trying to show a property $F(n+3)$, substitute $n+3$ into the definition for k ($k=n+3$).

  $F(n+3) = F(n+3\text{-}2) + F(n+3\text{-}1) = F(n+1) + F(n+2)$

Fibonacci Numbers

- Now, prove the same formula using the 2nd principle of induction (Problem 20).

  $P(n)$: $F(n + 3) = 2F(n + 1) + \mathrm{F}(n)$ for $n \geq 1$

- Step 1: Base Case
  - Since we are using 2 previous values to compute the next value, use 2 base cases

Problem 20 continued

- Step 2.a: Assume

  $P(r)$: $F(r+3) = 2F(r+1) + F(r)$ for $1 \leq r \leq k$

  - What does this mean? We can assume
    - $F(1+3) = 2F(1+1) + F(1) \Rightarrow F(4) = 2F(2) + F(1)$
    - $F(2+3) = 2F(2+1) + F(2) \Rightarrow F(5) = 2F(3) + F(2)$
    - …
    - $F(k\text{-}1+3) = 2F(k\text{-}1+1) + F(k\text{-}1) \Rightarrow$

      $F(k+2) = 2F(k) + F(k\text{-}1)$
    - $F(k+3) = 2F(k+1) + F(k)$

Problem 20 continued

- Step 2.b: Prove

  $P(k+1)$: $F(\mathrm{k}+1+3)$?= $2F(k+1+1) + F(k+1)$

  $\Rightarrow F(k+4)$?= $2F(k+2) + \mathrm{F}(k+1)$

  - Start with the definition of Fibonacci numbers

    $F(n) = F(n\text{-}2) + F(n\text{-}1)$

    - Since we are trying to prove $F(k+4)$, substitute $k+4$ for n in the definition formula

      $F(k+4) = F(k+4\text{-}2) + F(k+4\text{-}1) = F(k+2) + F(k+3)$

    - Now, do we have any information we can use about $F(k+2)$ and $F(k+3)$?

Fibonacci Sequence

- Example 31: Prove that in the Fibonacci sequence

  $$F(n+4) = 3F(n+2) - F(n) \qquad \text{for all } n \geq 1$$

- Proof by induction

Fibonacci Sequence

- Prove the formula without induction

$$F(n+4) = 3F(n+2) - F(n) \qquad \text{for all } n \geq 1$$

- Use the recurrence relation from the definition of Fibonacci numbers

$$F(n) = F(n-2) + F(n-1)$$

Recursively Defined Sets

- A ***sequence*** is a collection of objects which have a specific order
- A ***set*** is a collection of object with no order imposed
- Example: A recursive definition for the set of propositional wffs
    1. Any statement letter is a wff.
    2. If P and Q are wffs, so are $(P \vee Q)$, $(P \wedge Q)$, $(P \rightarrow Q)$, (P'), and $(P \leftrightarrow Q)$
- Show how to build $((A \vee (B')) \rightarrow C)$

Recursively Defined Sets

- Example 34: The set of all (finite-length) strings of symbols over a finite alphabet A is denoted by A^*. The recursive definition of A^* is
  1. The empty string λ (the string with no symbols) belongs to A^*.
  2. Any single member of A belongs to A^*.
  3. If x and y are strings in A^*, so is xy, the concatenation of strings x and y.
- If x = 1011 and y = 001, write the strings xy, yx, and $yx\lambda x$.

Recursively Defined Set

- Practice 17
  - Give a recursive definition for the set of all *binary* strings that are **palindromes**.
    - A **palindrome** is a string that reads the same forwards and backwards.

Backus-Naur Form

- BNF notation allows you to recursively define a set of strings
  - Angle brackets < > indicate items that are defined in terms of other items
  - Items without brackets cannot be further broken down
  - The vertical line | means or

Backus-Naur Form

- A BNF definition of an identifier
  - <identifier> ::= <letter> | <identifier><letter> |<identifier><digit>
  - <letter> ::= a | b | c | ... | z
  - <digit> ::= 1 | 2 | ... | 9
- How would the identifier $tmp1$ be built from the definition?

Recursively Defined Operations

- Some operations can also be defined recursively
  - Example 36: Exponentiation operation a^n on a nonzero real number a, where n is a nonnegative integer
  - Recursive definition

    1. $a^0 = 1$

    2. $a^n = (a^{n-1})a$ for $n \geq 1$

Recursively Defined Operations

- Practice 18
  - Let x be a string over some alphabet
  - Give a recursive definition for the operation x^n (concatenation of x with itself n times) for $n \geq 1$.

Recursively Defined Algorithms

- Write a computer algorithm to calculate $S(n)$ from Example 29

  1. $S(1) = 2$
  2. $S(n) = 2S(n-1)$ for $n \geq 2$

- Iterative vs. recursive

Binary Search

- Input
  - List of items sorted in nondecreasing order
  - An item x which you would like to find
- Basic Idea
  - Compare x to the middle item in list
  - If it matches, you’re done.
  - If x is less than middle item
    - Search first half of list
  - If x is greater than middle item
    - Search second half of list

Binary Search Algorithm

```
BinarySearch(list L; integer i; integer j; itemtype x)
// searches sorted list L from L[i] to L[j] for item x
  if (i > j) then
        write ("not found")
  else
        find the index k of the middle item in the list L[i]-L[j]
        if x = middle item then
                write("found")
        else
                if x < middle item then
                        BinarySearch(L, I, k-1, x)
                else
                        BinarySearch(L, k+1, j, x)
                end if
        end if
  end if
end function BinarySearch
```

Binary Search

- Apply the binary search algorithm to the list
  - 3, 7, 8, 10, 14, 18, 22, 34