

# Defining Functions

- Functions with a **finite** domain can be described by specifying for each element in the domain the associated element in the codomain.

- **Examples:**

- 

$$f(x) = \begin{cases} 1 & \text{if } x = 1 \\ 0 & \text{if } x = 0 \text{ or } x = 3 \end{cases}$$

- Let  $x$  a real.  $f(x) = 1$  if  $0 \leq x \leq 3$

- The two basic mechanisms for defining functions on **infinite** domains are
  - **explicit** definitions and
  - **recursive** definitions.

# Explicit Definitions

- An **explicit definition** of a function  $f$  consists of giving an expression that indicates for each domain element  $x$  how  $f(x)$  is obtained from previously defined functions (including constants) by composition.
- **Examples**

$$\begin{aligned} \mathit{zero}(x) &= 0 \\ \mathit{add3}(x) &= x + 3 \\ \mathit{gt}(x, y) &= \text{if } x > y \text{ then } 1 \text{ else } 0 \\ \chi_A(x) &= \text{if } x \in A \text{ then } 1 \text{ else } 0 \end{aligned}$$

## *Note*

- The last function is called the *characteristic function* of the set  $A$ .
- If-then-else may be used for case distinctions in function definitions.

# Recursive Definitions

- A **recursive definition** of a function consists of giving an expression for every domain element  $x$  that indicates how  $f(x)$  is obtained from previously defined functions **and** values of  $f$  for “smaller” arguments (by composition).

→ **Self-references**

- The **recursion principle** specifies under which conditions such definitions with self-references are **well-formed**.

- **Example**

The number of *permutations* of  $n$  elements is  $n!$  (or  $fact(n)$ , read *n factorial*).

→ **Order**

This function can be defined recursively by:

$$fact(n) = \text{if } n = 0 \text{ then } 1 \text{ else } n * fact(n - 1).$$

The values  $fact(n)$ , for all  $n > 0$ , depend on values  $fact(k)$ , where  $k$  is smaller than  $n$ . Here  $k = n - 1$ . This case is called the **general case**.

$n = 0$  is called the **exit condition** or the **basis condition**.

## Well-Formed Recursive Definitions

- A **well-formed recursive definition** of a function  $f$  consists of two parts:
  - the **basis case** defines the function  $f$  for the “smallest” arguments in terms of previously defined functions (including constants), (*no  $f$* ).
  - the **general case** defines values  $f(x)$  in terms of previously defined functions and values  $f(y)$  for “smaller” arguments  $y$ .
- In the case of definitions of functions over the natural numbers, smaller is interpreted in the usual sense.

Later on we will see recursive definitions of functions on other domains, such as lists, where “smaller” necessarily has to be interpreted differently. We use an **ordering** on the elements we consider.

## Computing Values of Recursively Defined Functions

- The **evaluation** of a recursively defined function for a specific argument involves two kinds of operations:
  - **substitutions** use the function definition to “expand” an application, whereas
  - **simplifications** use knowledge about previously defined (or primitive) functions to “reduce” an expression.
- The evaluation process will **terminate** if the definition is **well-formed**.
- **Example:**

$$\begin{aligned} fact(5) &= 5 * fact(5 - 1) && \text{(substitution)} \\ &= 5 * fact(4) && \text{(simplification)} \\ &= 5 * (4 * fact(4 - 1)) && \text{(substitution)} \\ &= 20 * fact(3) && \text{(simplification)} \\ &\vdots \\ &= 120 \end{aligned}$$

## Example: Squares

- There are different ways to define a function.
- For instance, the function that squares its argument can be defined **explicitly** in terms of multiplication,

$$\mathit{square}(x) = x * x,$$

or by **recursion**:

$$\mathit{square}(x) = \begin{array}{l} \text{if } x = 0 \text{ then } 0 \\ \text{else } \mathit{square}(x - 1) + 2x - 1 \end{array}$$

From the recursive definition we get the following function values:

$$\begin{array}{l} \mathit{square}(0) = 0 \\ \mathit{square}(1) = \mathit{square}(0) + 1 = 1 \\ \mathit{square}(2) = \mathit{square}(1) + 3 = 4 \\ \mathit{square}(3) = \mathit{square}(2) + 5 = 9 \\ \mathit{square}(4) = \mathit{square}(3) + 7 = 16 \\ \vdots \end{array}$$

The two definitions above define the same function, as

$$x * x = (x - 1) * (x - 1) + 2x - 1.$$

## Addition and GCD

- Addition

$$add(a, b) = \begin{cases} a & \text{if } b = 0 \\ add(a, b - 1) + 1 & \text{otherwise} \end{cases}$$

- Greatest Common Divisor

$$gcd(a, b) = \begin{cases} a & \text{if } b = 0 \\ gcd(b, a \bmod b) & \text{otherwise} \end{cases}$$

# Fibonacci Numbers

- The recursive definition of the following well-known function (*Fibonacci function*) employs the function values for **several** smaller arguments:

$$fib(n) = \begin{cases} 1 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ fib(n-1) + fib(n-2) & \text{if } n > 1 \end{cases}$$

- The corresponding function values are called **Fi-bonacci numbers**:

$$\begin{aligned} fib(0) &= 1 \\ fib(1) &= 1 \\ fib(2) &= fib(1) + fib(0) = 2 \\ fib(3) &= fib(2) + fib(1) = 3 \\ fib(4) &= fib(3) + fib(2) = 5 \\ fib(5) &= fib(4) + fib(3) = 8 \dots \end{aligned}$$

- The Fibonacci numbers were originally defined to count the number of rabbits after  $n$  generations, but they pop up in an amazing variety of places:
  - The *Golden Ratio* of architecture,  $\phi \approx fib(n)/fib(n-1) = (1 + \sqrt{5})/2 \approx 1.618$
  - The angles between leaves in spiral pine cones grow as ratios of Fibonacci numbers.
  - They arise in the analysis of computer algorithms.

# Well-defined Functions

- A key requirement of a recursive definition is that it be formulated in terms of function values for **smaller** arguments.
- A recursive function is said **well-defined**, if it is possible to compute  $f(n)$  for all  $n$  for which the function is defined. Otherwise it is said **partially defined**.
- Consider this definition,

$$F(x) = \text{if } x = 0 \text{ then } 0 \text{ else } F(x + 1) + 1$$

and corresponding attempts at computing function values,

$$\begin{aligned} F(0) &= 0 \\ F(1) &= F(2) + 1 \\ &= F(3) + 2 \\ &= F(4) + 3 \\ &\vdots \end{aligned}$$

This function is defined for one argument only. So  $F$  is not well-defined.

- What about the function  $G$ , defined for positive integers by

$$G(n) = \begin{cases} 0 & \text{if } n = 1 \\ 1 + G(n/2) & \text{if } n \text{ is even} \\ G(3n - 1) & \text{if } n \text{ is odd and } n > 1 \end{cases}$$

## Study of G

- $G$  is not well-defined for all arguments.

We have

$$G(1) = 0$$

$$G(2) = 1 + G(1) = 1$$

$$G(3) = G(8) = 1 + G(4) = 1 + (1 + G(2)) = 3$$

$$G(4) = 1 + G(2) = 2$$

$$G(5) = G(14) = 1 + G(7) = 1 + G(20)$$

$$= 1 + (1 + G(10)) = 3 + G(5)$$

⋮

Thus, if  $G(5)$  was defined, we could infer the contradictory statement that  $0 = 3$ ! In other words,  $G(5)$  must be undefined.

# A New Function $H$

- It has been *conjectured* (and shown up to one trillion) that a slight modification,

$$H(n) = \begin{cases} 0 & \text{if } n = 1 \\ 1 + H(n/2) & \text{if } n \text{ is even} \\ H(3n + 1) & \text{if } n \text{ is odd and } n > 1 \end{cases}$$

defines a well-defined function on all positive integers.

– **H(2):** H(1)

– **H(10):** H(5)–H(16)–H(8)–H(4)–H(2)–H(1)

– **H(17):** H(52)–H(26)–H(13)–H(40)–H(20)–H(10)–H(5)–H(16)–H(8)–H(4)–H(2)–H(1)

– **H(21):** H(64)–H(32)–H(16)–H(8)–H(4)–H(2)–H(1)

– **H(35):** H(106)–H(53)–H(160)–H(80)–H(40)–H(20)–H(10)–H(5)–H(16)–H(8)–H(4)–H(2)–H(1)

$H$  counts the number of downward steps this path takes.

$$H(2) = 1$$

$$H(17) = 9$$

$$H(21) = 6$$

$$\text{and } H(35) = 10.$$

## More General Recursive Definitions

- **Example:**

$$M(n) = \begin{cases} n - 10 & \text{if } n > 100 \\ M(M(n + 11)) & \text{if } n \leq 100 \end{cases}$$

- This function is known as “McCarthy’s 91 function.”

Its definition uses **nested** recursive function applications.

- Consider one instance,

$$\begin{aligned} M(99) &= M(M(110)) && \text{(since } 99 \leq 100\text{)} \\ &= M(100) && \text{(since } 110 > 100\text{)} \\ &= M(M(111)) && \text{(since } 100 \leq 100\text{)} \\ &= M(101) && \text{(since } 111 > 100\text{)} \\ &= 91 && \text{(since } 101 > 100\text{)} \end{aligned}$$

- Is this function defined for all arguments  $n \leq 100$ ?

The function is in fact defined for all positive integers and remarkably takes the value 91 for all arguments less than or equal to 101.

- M can also be defined explicitly by:

$$M(n) = \begin{cases} n - 10 & \text{if } n > 100 \\ 91 & \text{if } n \leq 100 \end{cases}$$

# Evaluation Schemes

- 

$$f(x, y) = \begin{cases} 0 & \text{if } x = 0 \\ f(x - 1, f(x, y)) & \text{otherwise} \end{cases}$$

- Consider  $f(1,1)$ .

## **Innermost evaluation**

$$f(1, 1) = f(0, f(1, 1)) = f(0, f(0, f(1, 1))) = \dots$$

## **Outermost evaluation**

$$f(1, 1) = f(0, f(1, 1)) = 0$$

## **Simultaneous**

$$f(1, 1) = f(0, f(1, 1)) = 0$$

- Innermost evaluation does not always terminate.
- Outermost evaluation does always terminate.
- Innermost evaluation is more efficient than outermost evaluation (*Convergence*)

# Sequences

- A **sequence** is formally a function on the natural numbers, or some initial segment of the natural numbers.

- For example, the infinite sequence

$$1, -\frac{1}{2}, \frac{1}{3}, -\frac{1}{4}, \frac{1}{5}, \dots$$

can be thought of as a function  $f$  that maps each natural number  $n$  to a rational number,  $f(n) = (-1)^n / (n + 1)$ .

- The finite sequence

$$2, 3, 5, 7, 11, 13$$

can be thought of as a function  $f : \{0, 1, 2, 3, 4, 5\} \rightarrow \mathbb{N}$  such that

$$\begin{aligned} f(0) &= 2 \\ f(1) &= 3 \\ f(2) &= 5 \\ f(3) &= 7 \\ f(4) &= 11 \\ f(5) &= 13 \end{aligned}$$

- Arrays are essentially finite sequences, though as a data type they also come with operations for accessing array elements, changing them, etc.

# Recursively Defined Sequences

- An infinite sequence

$$a_0, a_1, a_2, a_3, \dots$$

is formally a function  $a$  defined on the domain of natural numbers (i.e., nonnegative integers) by  $a(n) = a_n$ .

- Such sequences are often alternatively specified by recursive identities, or so-called **recurrence relations**.
- In a recurrence relation each term  $a_k$  is equated to an expression that may contain (some or all) of the  $i$  predecessors  $a_{k-1}, a_{k-2}, \dots, a_{k-i}$ , where  $i$  is a fixed integer.
- The terms  $a_0, \dots, a_{i-1}$ , which do not have  $i$  predecessors, need to be defined separately by so-called **initial conditions**.

- The recurrence relation for the function  $M$  consists of the following identities:

$$\begin{aligned}M_0 &= 0 \\M_n &= 2M_{n-1} + 1 \quad \text{if } n > 0\end{aligned}$$

- The Fibonacci numbers are characterized by the following recurrence relation:

$$\begin{aligned}F_1 &= 1 \\F_0 &= 1 \\F_k &= F_{k-1} + F_{k-2} \quad \text{if } k \geq 2\end{aligned}$$

## Arithmetic and Geometric Sequences

- A sequence  $a_0, a_1, a_2, \dots$  is called an **arithmetic sequence** if there is a constant  $d$  such that  $a_k = a_{k-1} + d$ , for all  $k \geq 1$ .
- A sequence  $a_0, a_1, a_2, \dots$  is called a **geometric sequence** if there is a constant  $r$  such that  $a_k = r \cdot a_{k-1}$ , for all  $k \geq 1$ .
- For example, balances in an interest-bearing account in which interest is compounded, will follow a geometric sequence if no deposits or withdrawals are made.
- Functions defining arithmetic or geometric sequences can be described **explicitly**.
  - We have  $a_n = a_0 + d \cdot n$ , for all  $n \geq 0$ , for any arithmetic sequence.
  - We have  $a_n = a_0 \cdot r^n$ , for all  $n \geq 0$ , for any geometric sequence.
- The latter identities may be viewed as **solutions** of the respective recurrence relations above.
- Methods for solving other kinds of recurrence relations are of relevance to the analysis of algorithms, but will not be discussed in this course.

Proving that a solution is correct typically requires **mathematical induction** arguments, a topic we will discuss later on.

# Summary

- Recursion is a general method for the definition of functions (and also a powerful technique for designing algorithms).
- Recursive definitions generally specify only partial functions.

Recall that a partial function satisfies the uniqueness, but not the completeness property required of total functions.

- If a recursive definition employs only values of the defined function for smaller arguments, it defines a total function.
- The evaluation of recursively defined function for specific arguments is based on calculation by substitution and simplification.
- These two concepts,
  - definition by recursion and
  - evaluation by substitution and simplification,

form the foundation of *functional programming languages* such as ML.