

# Recursion and Structural Induction

---

Andreas Klappenecker

# Inductively Defined Sets

# Motivating Example

---

Consider the set

$$A = \{3, 5, 7, \dots\}$$

There is a certain ambiguity about this "definition" of the set  $A$ .

Likely,  $A$  is the set of odd integers  $\geq 3$ .

[However,  $A$  could be the set of odd primes... ]

# Motivating Example

---

In Computer Science, we prefer to avoid such ambiguities. You will often encounter sets that are **inductively defined**.

We can specify the set as follows:

$3 \in A$  and if  $n$  is in  $A$ , then  $n+2$  is in  $A$ .

In this definition, there is

- (a) an initial element in  $A$ , namely  $3$ .
- (b) you construct additional elements by adding  $2$  to an element in  $A$ ,
- (c) nothing else belongs to  $A$ .

We will call this an **inductive definition** of  $A$ .

# Inductively Defined Sets

---

An **inductive definition** of a set  $S$  has the following form:

- (a) **Basis:** Specify one or more "initial" elements of  $S$ .
- (b) **Induction:** Give one or more rules for constructing "new" elements of  $S$  from "old" elements of  $S$ .
- (c) **Closure:** The set  $S$  consists of exactly the elements that can be obtained by starting with the initial elements of  $S$  and applying the rules for constructing new elements of  $S$ .

The closure condition is **usually omitted**, since it is **always** assumed in inductive definitions.

# Example 1: Natural Numbers

---

Let  $S$  be the set defined as follows:

Basis:  $0 \in S$

Induction: If  $n \in S$ , then  $n+1 \in S$

Then  $S$  is the set of natural numbers (with 0).

Closure? Implied!

## Example 2

---

Let  $S$  be the set defined as follows:

Basis:  $0 \in S$

Induction: If  $n \in S$ , then  $2n+1 \in S$ .

Can you describe the set  $S$ ?

$$S = \{0, 1, 3, 7, 15, 31, \dots\} = \{2^n - 1 \mid n \text{ a nonnegative integer}\}$$

$$\text{since } 2^0 - 1 = 0 \text{ and } 2^{n+1} - 1 = 2(2^n - 1) + 1$$

## Example 3: Well-Formed Formulas

---

We can define the set of well-formed formulas consisting of variables, numerals, and operators from the set  $\{+, -, *, /\}$  as follows:

Basis:  $x$  is a well-formed formula if  $x$  is a numeral or a variable.

Induction: If  $F$  and  $G$  are well-formed formulas, then  $(F+G)$ ,  $(F-G)$ ,  $(F^*G)$ , and  $(F/G)$  are well-formed formulas.

Examples:  $42$ ,  $x$ ,  $(x+42)$ ,  $(x/y)$ ,  $(3/0)$ ,  $(x^*(y+z))$

# Example 4: Lists

---

We can define the set L of finite lists of integers as follows.

Basis: The empty list () is contained in L

Induction: If i is an integer, and l is a list in L, then (cons i l) is in L.

[Note: This is the Lisp style of lists, where (cons i l) appends the data item i at the front of the list l]

Example: (cons 1 (cons 2 (cons 3 () ))) is the list (1 2 3) in Lisp.

# Example 5: Binary Trees

---

We can define the set  $B$  of binary trees over an alphabet  $A$  as follows:

Basis:  $\langle \rangle \in B$ .

Induction: If  $L, R \in B$  and  $x \in A$ , then  $\langle L, x, R \rangle \in B$ .

Example:  $\langle \langle \rangle, 1, \langle \rangle \rangle$  // tree with one node (1)

Example:  $\langle \langle \langle \rangle, 1, \langle \rangle \rangle, r, \langle \langle \rangle, 2, \langle \rangle \rangle \rangle$

// tree with root  $r$  and two children (1 and 2).

# Applications of Inductively Defined Sets

---

In Computer Science, we typically use inductively defined sets (a.k.a. recursively defined sets) when defining:

- programming languages (via grammars)
- logic (via well-formed logical formulas)
- data structures (binary trees, rooted trees, lists).
- fractals

We also use them in connection with functional programming languages.

Extremely popular in Computer Science!

# Recursively Defined Functions

---

# Recursively Defined Functions

---

Suppose we have a function with the set of nonnegative integers as its domain.

We can specify the function as follows:

**Basis step:** Specify the value of the function at 0

**Inductive step:** Give a rule for finding its value at an integer from its values at smaller integers.

This is called a **recursive** or **inductive definition**.

# Example: Factorial Function

---

We can define the factorial function  $n!$  as follows:

Base step:  $0! = 1$

Inductive step:  $n! = n(n-1)!$

# Example: Fibonacci Numbers

---

The Fibonacci numbers  $f_n$  are defined as follows:

Base step:  $f_0=0$  and  $f_1=1$

Inductive step:  $f_n = f_{n-1} + f_{n-2}$  for  $n \geq 2$

We can use the recursive definition of the Fibonacci numbers to prove many properties of these numbers. The recursive structure actually helps to formulate the proofs.

# Recursively Defined Functions

---

One can define recursively defined functions for domains other than the nonnegative integers.

In general, a function  $f$  is called **recursively defined** if and only if at least one value  $f(x)$  is defined in terms of another value  $f(y)$ , where  $x$  and  $y$  are distinct elements.

[However, we will typically consider recursively defined functions that have more structure than this definition suggests.]

# Example: Ackermann Function

$$A(m, n) = \begin{cases} n + 1 & \text{if } m = 0, \\ A(m - 1, 1) & \text{if } m > 0 \text{ and } n = 0, \\ A(m - 1, A(m, n - 1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

The Ackermann function has particular significance in computability theory. The values of  $A(m,n)$  grow very, very quickly.

# Example: Ackermann Function

---

# In Ruby, the Ackermann function can be defined as follows:

```
def A(m,n)
    return n+1 if m==0
    return A(m-1,1) if n==0
    return A(m-1,A(m,n-1))
end
```

# Now try calculating  $A(0,0)$ ,  $A(1,1)$ ,  $A(2,2)$ ,  $A(3,3)$ ,  $A(4,4)$

# Example: Ackermann Function

$$A(m, n) = \begin{cases} n + 1 & \text{if } m = 0, \\ A(m - 1, 1) & \text{if } m > 0 \text{ and } n = 0, \\ A(m - 1, A(m, n - 1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

Why does the recursion terminate?

Lexicographically order the pairs  $(m, n)$ .

So  $(m, n) < (m', n')$  iff  $m < m'$  or  $(m = m' \text{ and } n < n')$ .

Note that arguments used in the RHS are lexicographically smaller than the arguments in the LHS. Thus, eventually we need to end up in case  $m=0$ , though it might take an extremely long time.

# Structural Induction

---

# Structural Induction

---

Structural induction asserts a property about elements of an inductively defined set. The proof method directly exploits the inductive definition of the set.

The method is more powerful than strong induction in the sense that one can prove statements that are difficult (or impossible) to prove with strong induction. Typically, though, it is simply used because it is more convenient than (strong) induction.

# Structural Induction

---

In **structural induction**, the proof of the assertion that every element of an inductively defined set  $S$  has a certain property  $P$  proceeds by showing that

Basis: Every element in the basis of the definition of  $S$  satisfies the property  $P$ .

Induction: Assuming that every argument of a constructor has property  $P$ , show that the constructed element has the property  $P$ .

# Example: Binary Trees

---

Recall that the set  $B$  of binary trees over an alphabet  $A$  is defined as follows:

Basis:  $\langle \rangle \in B$ .

Induction: If  $L, R \in B$  and  $x \in A$ , then  $\langle L, x, R \rangle \in B$ .

We can now prove that every binary tree has a property  $P$  by arguing that

Basis:  $P(\langle \rangle)$  is true.

Induction: For all binary trees  $L$  and  $R$  and  $x$  in  $A$ , if  $P(L)$  and  $P(R)$ , then  $P(\langle L, x, R \rangle)$ .

# Example: Binary Trees

Let  $f: B \rightarrow \mathbb{N}$  be the function defined by

$$f(\langle \rangle) = 0$$
$$f(\langle L, x, R \rangle) = \begin{cases} 1 & \text{if } L = R = \langle \rangle \\ f(L) + f(R) & \text{otherwise} \end{cases}$$

**Theorem:** Let  $T$  in  $B$  be a binary tree. Then  $f(T)$  yields the number of leaves of  $T$ .

# Example: Binary Trees

---

**Theorem:** Let  $T$  in  $B$  be a binary tree. Then  $f(B)$  yields the number of leaves of  $T$ .

**Proof:** By structural induction on  $T$ .

**Basis:** The empty tree has no leaves, so  $f(\langle \rangle) = 0$  is correct.

**Induction:** Let  $L, R$  be trees in  $B$ ,  $x$  in  $A$ .

Suppose that  $f(L)$  and  $f(R)$  denotes the number of leaves of  $L$  and  $R$ , respectively.

If  $L=R=\langle \rangle$ , then  $\langle L, x, R \rangle = \langle \langle \rangle, x, \langle \rangle \rangle$  has one leaf, namely  $x$ , so  $f(\langle \langle \rangle, x, \langle \rangle \rangle) = 1$  is correct.

# Example: Binary Trees

---

If  $L$  and  $R$  are not both empty, then the number of leaves of the tree  $\langle L, x, R \rangle$  is equal to the number of leaves of  $L$  plus the number of leaves of  $R$ . Hence, by induction hypothesis, we get

$$f(\langle L, x, R \rangle) = f(R) + f(L)$$

as claimed.

This completes the proof.

# Example: Full Binary Trees

---

The set of **full binary trees** can be defined as follows:

Basis: There is a full binary tree consisting of a single vertex  $r$

Induction: If  $T_1$  and  $T_2$  are disjoint full binary trees and  $r$  in  $A$  is a node, then  $\langle T_1, r, T_2 \rangle$  is a full binary tree with root  $r$  and left subtree  $T_1$  and right subtree  $T_2$ .

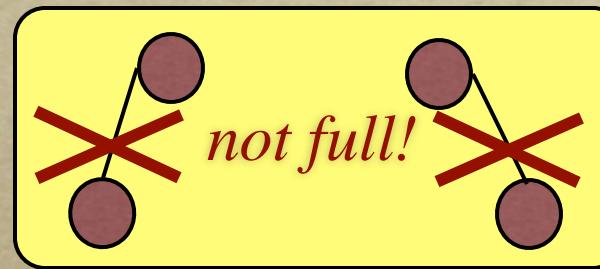
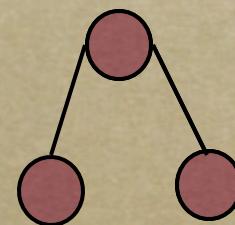
The difference between binary trees and full binary trees is in the **basis step**. A binary tree is full if and only if each node is either a leaf or has precisely two children.

# Small Full Binary Trees

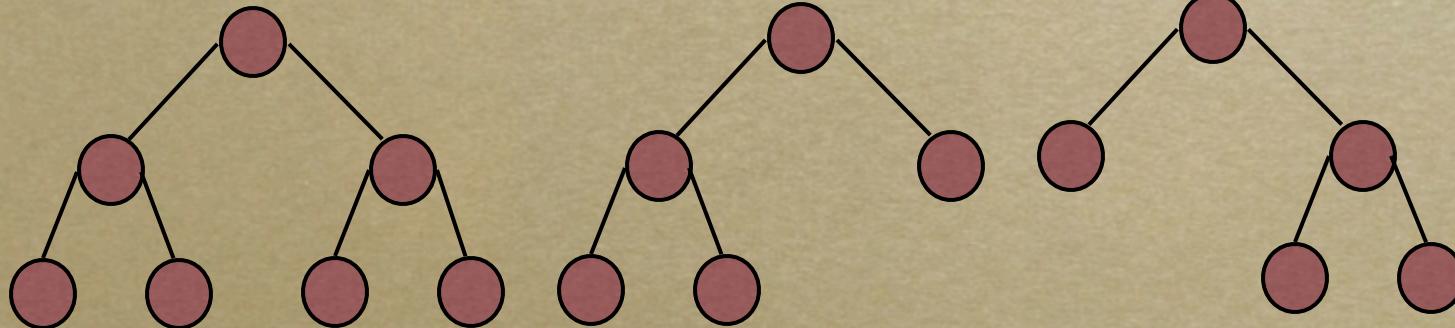
Level 0:



Level 1:



Level 2:



# Height of a Full Binary Tree

---

Let  $T$  be a full binary tree over an alphabet  $A$ . We define the **height**  $h(T)$  of a full binary tree as follows:

Basis: For  $r$  in  $A$ , we define  $h(r) = 0$ ; that is, the height of a full binary tree with just a single node is 0.

Induction: If  $L$  and  $R$  are full binary trees and  $r$  in  $A$ , then the tree  $\langle L, r, R \rangle$  has height

$$h(\langle L, r, R \rangle) = 1 + \max(h(L), h(R)).$$

# Number of Nodes of a Full Binary Tree

---

Let  $n(T)$  denote the number of nodes of a full binary tree over an alphabet  $A$ . Then

Basis: For  $r$  in  $A$ , we have  $n(r)=1$ .

Induction: If  $L$  and  $R$  are full binary trees and  $r$  in  $A$ , then the number of nodes of  $\langle L, r, R \rangle$  is given by

$$n(\langle L, r, R \rangle) = 1 + n(L) + n(R).$$

# Example of Structural Induction

**Theorem:** Let  $T$  be a full binary tree over an alphabet  $A$ . Then we have  $n(T) \leq 2^{h(T)+1}-1$ .

**Proof:** By structural induction.

**Basis step:** For  $r$  in  $A$ , we have  $n(r)=1$  and  $h(r)=0$ , therefore, we have  $n(r) = 1 \leq 2^{(0+1)-1} = 2^{h(r)+1}-1$ , as claimed.

**Inductive step:** Suppose that  $L$  and  $R$  are full binary trees that satisfy  $n(L) \leq 2^{h(L)+1}-1$  and  $n(R) \leq 2^{h(R)+1}-1$ . Then the tree  $T=\langle L, r, R \rangle$  satisfies:

$$n(T) = 1 + n(L) + n(R) \leq 1 + 2^{h(L)+1}-1 + 2^{h(R)+1}-1$$

$$\leq 2 \max(2^{h(L)+1}, 2^{h(R)+1})-1, \text{ since } a+b \leq 2\max(a,b)$$

$$\leq 2 \cdot 2^{\max(h(L), h(R))+1} - 1 = 2 \cdot 2^{h(T)} - 1 = 2^{h(T)+1}-1$$