Lecture 22: Recursive Definitions and Structural Induction

Dr. Chengjiang Long
Computer Vision Researcher at Kitware Inc.
Adjunct Professor at SUNY at Albany.
Email: **clong2@albany.edu**

Outline

- Recursive Definitions
- Structural Induction

Outline

- **Recursive Definitions**
- Structural Induction

Example

- The sequence is defined by the following algorithm:

1. At the first step there are two numbers: 1, 1.
2. At the next step, insert between two numbers a new number which is the sum of two neighbors:

1, 2, 1

1, 3, 2, 3, 1

1, 4, 3, 5, 2, 5, 3, 4, 1

…

- The sum of the sequence was defined:
  - recursively $S(0) = 2,\ S(n+1) = 3S(n) - 2$,
  - explicitly $S(n) = 3^n + 1$.

Recursive vs. Explicit

Why Recursive Definition is also called Inductive Definition?

Is the sequence in the previous example defined recursively or explicitly?

Can the sequence be defined explicitly?

Recursive Definition :

- in some cases is the only possible,
- reflects the algorithm,
- easier to read when programed,
- needs stack (risk of stack overflow)

Explicit Definition:

- faster
- more reliable
- hard to read

Recursion is useful to define sequences, functions, sets, and algorithms.

Recursive or Inductive Function Definition

- Basis Step: Specify the value of the function for the base case.
- Recursive Step: Give a rule for finding the value of a function from its values at smaller integers greater than the base case.

Inductive Definitions

- We completely understand the function f(n) = n! right?
  - n! = 1 · 2 · 3 · … · (n-1) · n, n ≥ 1

But equivalently, we could define it like this:

Recursive Case

Base Case

$$n! = \begin{cases} n \cdot (n-1)!, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Inductive (Recursive) Definition

Inductive Definitions

- The 2nd most common example:
- Fibonacci Numbers

Note why you need two base cases.

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

Base Cases

Recursive Case

Is there a non-recursive definition for the Fibonacci Numbers?

$$f(n) = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

(Prove by induction.)
All linear recursions have a closed form.

Recursively Defined Sets: Inductive Definitions

- Examples so far have been inductively defined functions.
- Sets can be defined inductively, too.

Give an inductive definition of $T = \{x: x \text{ is a positive integer divisible by } 3\}$

$3 \in S$ Base Case

$x,y \in S \rightarrow x + y \in S$ Recursive Case

Exclusion Rule: No other numbers are in S.

How can we prove it's correct?

Exclusion rule:
The set contains nothing other than those elements specified in the basic step or generated by the recursive step.

Recursively Defined Sets: Inductive Definitions

We want to show that the definition of S:

rule 1: $3 \in S$

rule 2: $x, y \in S \rightarrow x + y \in S$

Contains the same elements as the set: T={x: x is a positive integer divisible by 3}

To prove S = T, show

$T \subseteq S$

$S \subseteq T$

Perhaps the "trickiest" aspect of this exercise is realizing that there *is* something to prove! ☺

Recursively Defined Sets: Inductive Definitions

First, we prove $T \subseteq S$.

T = {x: x is a positive integer, multiple of 3}

If $x \in T$, then x = 3k for some integer k. We show by induction on |k| that $3k \in S$.

Hypothesis: P(n) = 3 n belongs to S, for all positive integers n.

Inductive Hypothesis: $3k \in S$

Base Case P(1) = $3 \in S$ since $3 \in S$ by rule 1.

Inductive Step: Assume $3k, \in S$, show that $3(k+1), \in S$.

Recursively Defined Sets: Inductive Definitions

- Inductive Step:
- $3k \in S$ by inductive hypothesis.
- $3 \in S$ by rule 1.
- $3k + 3 = 3(k+1) \in S$ by rule 2.

Recursively Defined Sets: Inductive Definitions

Next, we show that $S \subseteq T$.

That is, if an number x is described by S, then it is a positive multiple of 3.

Observe that the exclusion rule, all numbers in S are created by a finite number of applications of rules 1 and 2. We use the number of rule applications as our induction counter.

For example:

$3 \in S$ by 1 application of rule 1.

$9 \in S$ by 3 applications (rule 1 once and rule 2 twice).

Recursively Defined Sets: Inductive Definitions

- Base Case (k=1): If $x \in S$ by 1 rule application, then it must be rule 1 and x = 3, which is clearly a multiple of 3.

Inductive Hypothesis: Assume any number described by k or fewer applications of the rules in S is a multiple of 3

Inductive Step: Prove that any number described by (k+1) applications of the rules is also a multiple of 3, assuming IH.

Suppose the (k+1)st rule is applied (rule 2), and it results in value x = a + b. Then a and b are multiples of 3 by inductive hypothesis, and thus x is a multiple of 3.

Aside --- Message here: in a proof, follow a well-defined sequence of steps. This avoids subtle mistakes.

Outline

- Recursive Definitions
- **Structural Induction**

Recursive Definition of Structures

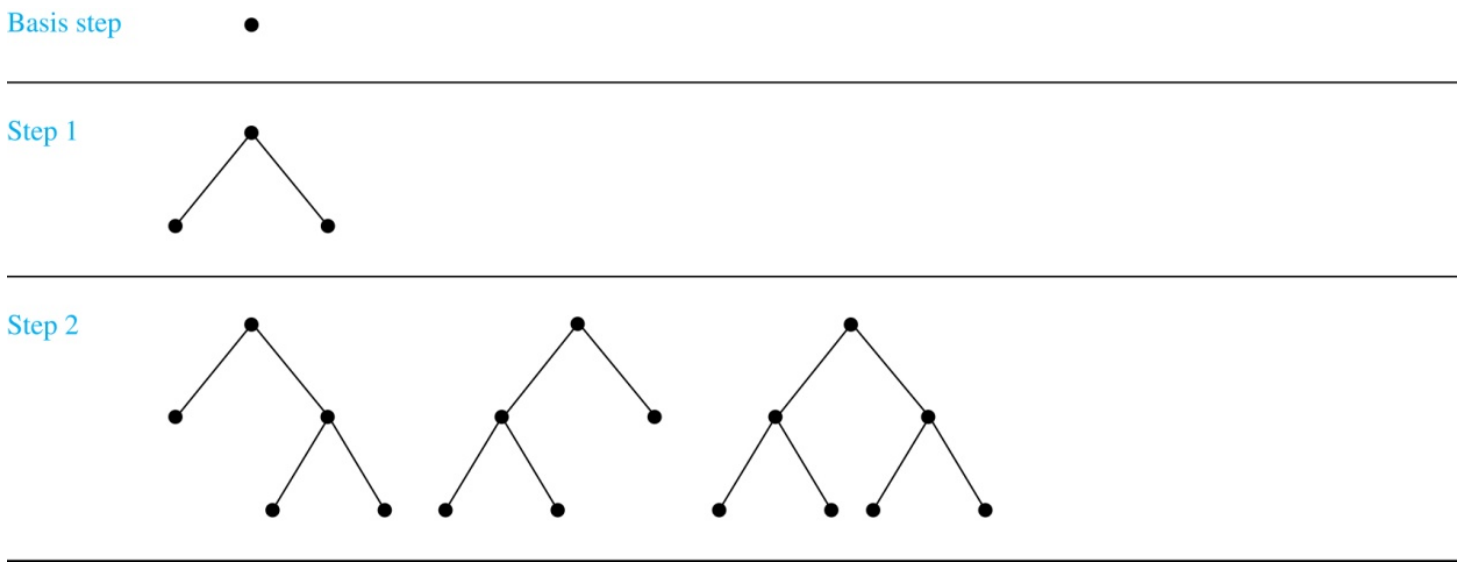
- In the previous lecture we showed that recursive definitions are applicable for functions, sets and other **structures**.

1. Define the “smallest” or “simplest” object (or objects) in the set of structures.
2. Define the ways in which “larger” or “more complex” objects in the set can be constructed out of “smaller” or “simpler” objects in the set.

Example of Structural Recursion

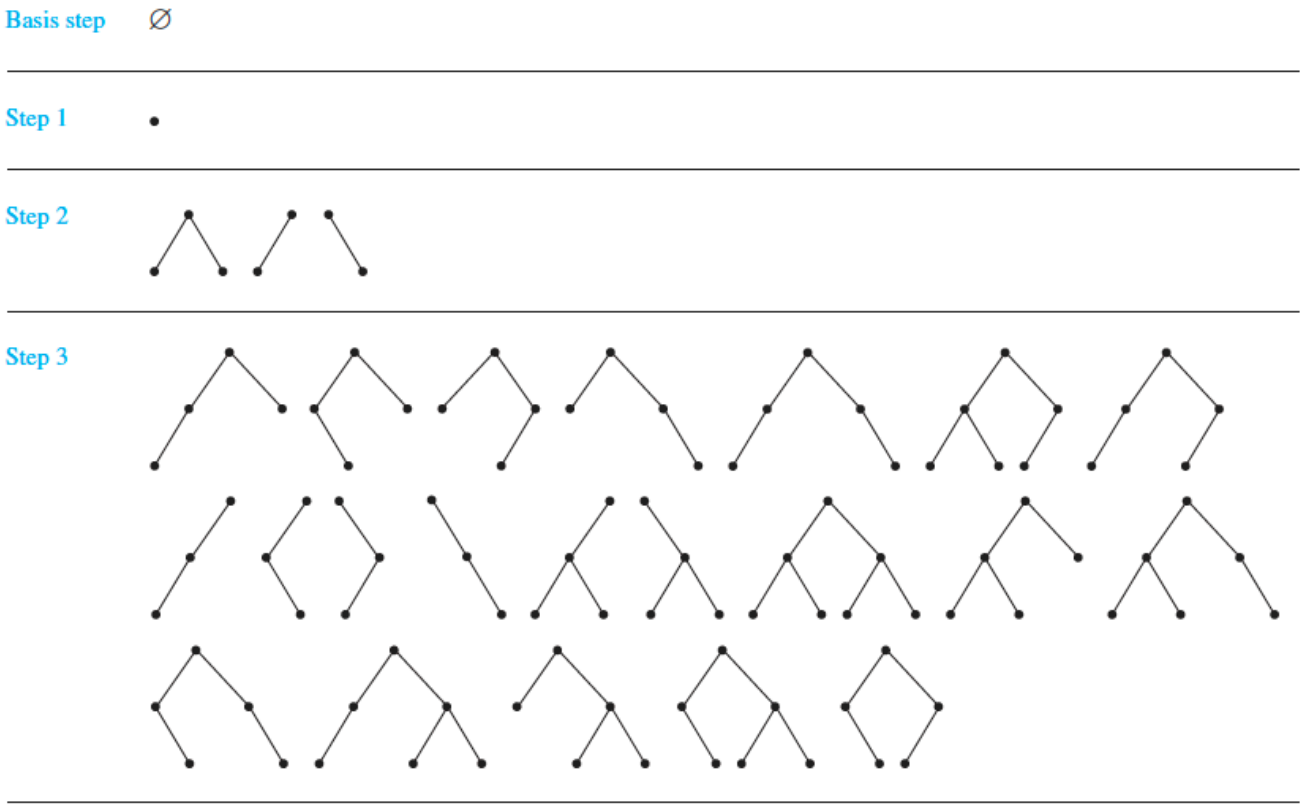
Full binary trees can be defined recursively as follows (*from the textbook*):

1. BASIS STEP: There is a full binary tree consisting of only a single node r.
2. RECURSIVE STEP: If T_1 and T_2 are disjoint full binary trees, there is a full binary tree, denoted by $T_1 \cdot T_2$, 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.



Extended Binary Tree

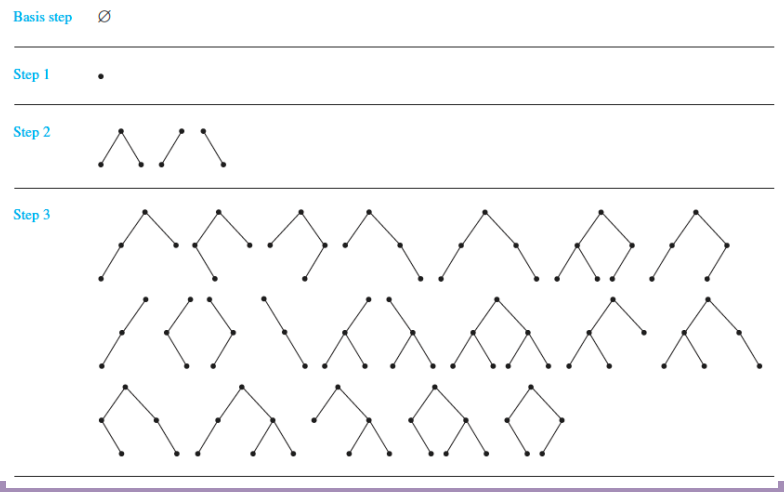
- Find the difference between full and extended binary trees.



Definition of Extended Binary Trees

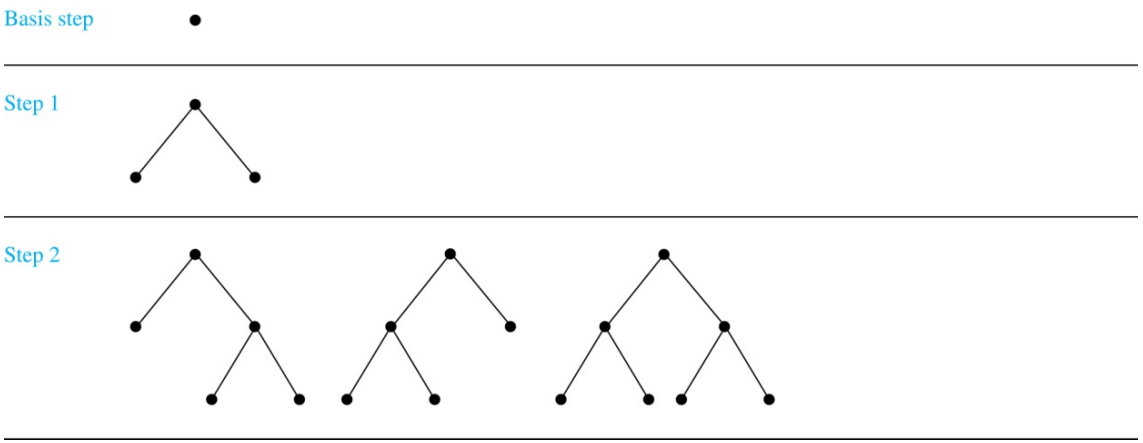
1. BASIS STEP: $\Lambda \cup V$ is in T. The set V represents the set of single nodes and Λ represents the empty binary tree in which **nodes**(Λ) = **edges**(Λ) = $\emptyset$ and **root**(Λ) is undefined. We use **nodes**(), **edges**(), and **root**() to specify sets of nodes, edges and roots of a tree.
2. RECURSIVE STEP: Let T_1 and T_2 be elements of T such that **nodes**(T_1) $\cap$ **nodes**(T_2) = $\emptyset$, and
   r $\notin$ **nodes**(T_1) $\cup$ **nodes**(T_2). Then the ordered triple $t = (r, T_1, T_2)$ is also in T. Furthermore, we define
   root(T) = r
   nodes(T) = $\{r\}$ $\cup$ **nodes**(T_1) $\cup$ **nodes**(T_2)
   edges(T) = $E \cup$ **edges**(T_1) $\cup$ **edges**(T_2)
   where E is the set that contains (r, **root**(T_1)) if $T_1 \neq \Lambda$, (r, **root**(T_2)) if $T_2 \neq \Lambda$, and nothing else.

How to change this recursive definition so that it becomes the definition of full binary trees?



Definition of Full Binary Trees

1. BASIS STEP: V is in T. The set V represents the set of single nodes.
2. RECURSIVE STEP: Let T_1 and T_2 be elements of T such that **nodes**$(T_1) \cap$ **nodes**$(T_2) = \emptyset$, and r $\notin$ **nodes**$(T_1) \cup$ **nodes**(T_2). Then the ordered triple $t = (r, T_1, T_2)$ is also in T. Furthermore, we define
root$(T) = r$
nodes$(T) =$ {r} $\cup$ **nodes**$(T_1) \cup$ **nodes**(T_2)
edges$(T) = E \cup$ **edges**$(T_1) \cup$ **edges**(T_2)
where E is the set that contains (r, **root**(T_1)), (r, **root**(T_2)).



Importance of "Nothing Else"

- **Example:** The set $S = \{n \in \mathbf{N} : n \geq 2\}$ is recursively defined:

1. BASIS STEP: $2 \in S$
2. RECURSIVE STEP: if $n \in S$, then $n + 1 \in S$
3. ***S* contains nothing else.**

- Why is this definition ambiguous without statement 3?
- Otherwise **N** and **Z** can also satisfy properties because it is structural recursion that defines a set. Sets are not ordered and basis step means entrance point, but does not mean the first point.

Mathematical and Structural Recursions

- As you can see from the previous example ***mathematical recursion*** over the natural numbers is an instance of the more general concept of *structural recursion* over values of an ***recursively-defined sets***.
- The natural numbers themselves is a recursively defined set. Because **N** may be defined as:

1. 0 is an element of **N**.
2. If *m* is an element of **N**, then so is *m+1*.
3. Nothing else is an element of **N**.

Mathematical and Structural Inductions

- We used mathematical induction to prove properties of functions mapped onto the set of natural numbers. And we saw that mathematical induction repeats recursive steps.
- To prove that a property *P(n)* holds of every *n* in **N**, it suffices to demonstrate the following facts:
  - Show that *P(0)* holds.
  - Assuming that *P(m)* holds, show that *P(m+1)* holds.
- Structural recursion is more universal notion and the approach that is used to prove properties of structures is called ***structural induction***.
- The pattern of reasoning using structural induction follows the recursive definition of structures.

Structural Induction and Binary Trees

- **Prove**: If T is a full binary tree, then $n(T) \leq 2^{h(T)+1} - 1$ where $n(T)$ is size of tree, $h(T)$ is height of a tree.

Solution: Use structural induction.

- BASIS STEP: The result holds for a full binary tree consisting only of a root, $n(T) = 1$ and $h(T) = 0$. Hence, $n(T) = 1 \leq 2^{0+1} - 1 = 1$.
- INDUCTION STEP: Assume $n(T_1) \leq 2^{h(T1)+1} - 1$ and also $n(T_2) \leq 2^{h(T2)+1} - 1$ whenever T_1 and T_2 are full binary trees.

- $n(T) = 1 + n(T_1) + n(T_2)$ (*by recursive formula of n(T), see textbook*)
- $\leq 1 + (2^{h(T1)+1} - 1) + (2^{h(T2)+1} - 1)$ (*by inductive hypothesis*)
- $\leq 2 \cdot \max(2^{h(T1)+1}, 2^{h(T2)+1}) - 1$
- $= 2 \cdot 2^{\max(h(T1),h(T2))+1} - 1$ ($\max(2^x, 2^y) = 2^{\max(x,y)}$)
- $= 2 \cdot 2^{h(T)} - 1$ (*by recursive definition of h(T), see textbook*)
- $= 2^{h(T)+1} - 1$

Principle of Structural Induction

- Let *R* be a recursive definition.
- Let *P* be a statement (property) about the elements defined by *R*.

- If the following hypotheses hold:
  - *P* is **True** for every element $b_1,\ldots,b_m$ in the base case of the definition *R*.
  - For every element *E* constructed by the recursive definition from some elements $e_1,\ldots,e_n$: *P* is **True** for $e_1,\ldots,e_n \Rightarrow$ *P* is true for *E*.
- Then we can conclude that:
- *P* is **True** for every element *E* defined by the recursive definition *R*.

Example: Set of Strings

A. Recursive definition of a set of valid strings :

- BASE CASE: $b \in S$, where b is the empty string, S is a set of valid strings.
- RECURSIVE STEP: asa, $s \in S$, where a is any letter from the alphabet A ($a \in A$).

B. Explicit formula for a valid string:

- $a^n b a^n$, where $n \geq 0$ is a number of letters.

- Prove that
  - Recursive Definition ⇔ Explicit Definition

Proof A: Recursive → Explicit

- Property $P(s)$: "Every element s constructed recursively is of the form $a^n b a^n$."

- By **Structural Induction**.
  - Base Case: $b = a^0 b a^0$.
  - Induction Step: Suppose $s = a^n b a^n$.
  - Structural Induction is to prove: if $P(b) \vee (\forall s\, P(s) \rightarrow P(asa))$, then $\forall s\, P(s)$.

  By recursive step next element $asa = a(a^n b a^n)a = a^{n+1} b a^{n+1}$

Proof B: Explicit → Recursive

- Predicate $P(n)$: "Every element of the form $a^n b a^n$ can be constructed recursively."
- By **Mathematical Induction**.
  - Base Case: $n = 0 \rightarrow a^0 b a^0 = b$.
  - Induction Hypothesis: Every element of the form $a^n b a^n$ can be constructed recursively.
  - Prove by mathematical induction must show: Every element of the form $a^{n+1} b a^{n+1}$ can be constructed recursively.

$$a^{n+1} b a^{n+1} = a(a^n b a^n)a = asa.$$

By the inductive hypothesis: $a^n b a^n$ satisfies the recursive definition; hence by the recursive step, $a^n b a^n$, so does $a^{n+1} b a^{n+1}$.

Observations on Structural Induction

- **Proofs by Structural Induction**
- Extends inductive proofs to discrete data structures: strings, lists, trees, etc.
- For every recursive definition there is a corresponding structural induction rule.
- The base case and the recursive step mirror the recursive definition.
  - Prove Base Case
  - Prove Recursive Step

Next class

- Topic: Recursive Algorithms
- Pre-class reading: Chap 5.4