CSE 15
Discrete Mathematics

Lecture 19 – Mathematical Induction (3)

Announcement

- HW #8
  - Due **5pm** 11/21 (Wed).
- Reading assignment
  - Ch.6.1 – 6.3 of textbook

Well-Ordering Property

- *Well-ordering property*: Every nonempty set of nonnegative integers has a least element.
- The well-ordering property can be generalized.
  - **Definition:** A set is *well ordered if every subset has a least element.*
    - **N** is well ordered under ≤ (≤ means standard ordering).
    - The set of finite strings over an alphabet using lexicographic ordering is well ordered.

Recursive Definitions and Structural Induction (Ch. 5.3)

- Recursively Defined Functions
- Recursively Defined Sets and Structures
- Structural Induction
- Generalized Induction

Recursively Defined Functions

Definition: A *recursive* or *inductive definition* of a function consists of two steps:

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

- A function $f(n)$ is the same as a sequence $a_0, a_1, \ldots$ where $f(i) = a_i$.

Recursively Defined Functions

Example: Suppose f is defined by:

$f(0) = 3,$

$f(n + 1) = 2f(n) + 3$

Find $f(1)$, $f(2)$, $f(3)$, $f(4)$

Solution:

- $f(1) = 2f(0) + 3 = 2 \cdot 3 + 3 = 9$
- $f(2) = 2f(1) + 3 = 2 \cdot 9 + 3 = 21$
- $f(3) = 2f(2) + 3 = 2 \cdot 21 + 3 = 45$
- $f(4) = 2f(3) + 3 = 2 \cdot 45 + 3 = 93$

Recursively Defined Functions

Example: Give a recursive definition of the factorial function $n!$:

Solution:

$f(0) = 1$

$f(n + 1) = f(n) \cdot (n + 1)$

Recursively Defined Functions

Example: Give a recursive definition of:

$$\sum_{k=0}^{n} a_k.$$

Solution: The first part of the definition is

$$\sum_{k=0}^{0} a_k = a_0.$$

The second part is

$$\sum_{k=0}^{n+1} a_k = \left(\sum_{k=0}^{n} a_k\right) + a_{n+1}.$$

Recursively Defined Sets and Structures

- ***Recursive definitions*** of sets have two parts:
  - The *basis step* specifies an initial collection of elements.
  - The *recursive step* gives the rules for forming new elements in the set from those already known to be in the set.

Recursively Defined Sets and Structures

Example: Subset of Integers *S.*

BASIS STEP: $3 \in S$.

RECURSIVE STEP: If $x \in S$ and $y \in S$, then $x + y$ is in S.

- Initially 3 is in *S*, then $3 + 3 = 6$, then $3 + 6 = 9$, etc.

Example: The natural numbers **N**.

BASIS STEP: $0 \in \mathbf{N}$.

RECURSIVE STEP: If n is in **N**, then $n + 1$ is in **N**.

- Initially 0 is in *S*, then $0 + 1 = 1$, then $1 + 1 = 2$, etc.

Strings

Definition: The set Σ^* of *strings* over the alphabet Σ:

BASIS STEP: $\lambda \in \Sigma^*$ (λ is the empty string)

RECURSIVE STEP: If w is in Σ^* and x is in Σ, then $wx \in \Sigma^*$.

Example: If $\Sigma = \{0,1\}$, the strings in Σ^* are the set of all bit strings, λ,0,1, 00,01,10,11, etc.

Example: If $\Sigma = \{a,b\}$, show that aab is in Σ^*.

- Since $\lambda \in \Sigma^*$ and $a \in \Sigma$, $a \in \Sigma^*$.
- Since $a \in \Sigma^*$ and $a \in \Sigma$, $aa \in \Sigma^*$.
- Since $aa \in \Sigma^*$ and $b \in \Sigma$, $aab \in \Sigma^*$.

String Concatenation

Definition: Two strings can be combined via the operation of *concatenation*.

Let Σ be a set of symbols and Σ* be the set of strings formed from the symbols in Σ.

We can define the concatenation of two strings, denoted by ·, recursively as follows:

BASIS STEP: If $w \in \Sigma^*$, then $w \cdot \lambda = w$.

RECURSIVE STEP: If $w_1 \in \Sigma^*$ and $w_2 \in \Sigma^*$ and $x \in \Sigma$, then

$$w_1 \cdot (w_2 x) = (w_1 \cdot w_2)x.$$

- Often $w_1 \cdot w_2$ is written as $w_1 w_2$.
- If $w_1 = abra$ and $w_2 = cadabra$, the concatenation

$$w_1 w_2 = abracadabra.$$

Length of a String

Example: Give a recursive definition of $l(w)$, the length of the string w.

Solution: The length of a string can be recursively defined by:

$l(\lambda) = 0$;

$l(wx) = l(w) + 1$ if $w \in \Sigma^*$ and $x \in \Sigma$.

Well-Formed Formulae in Propositional Logic

Definition: The set of *well-formed formulae* in propositional logic involving **T**, **F**, propositional variables, and operators from the set $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$.

BASIS STEP: **T**,**F**, and *s*, where *s* is a propositional variable, are well-formed formulae.

RECURSIVE STEP: If *E* and *F* are well formed formulae, then $(\neg E)$, $(E \wedge F)$, $(E \vee F)$, $(E \rightarrow F)$, $(E \leftrightarrow F)$, are well-formed formulae.

Examples: $((p \vee q) \rightarrow (q \wedge \mathbf{F}))$ is a well-formed formula.

$pq \wedge$ is not a well formed formula.

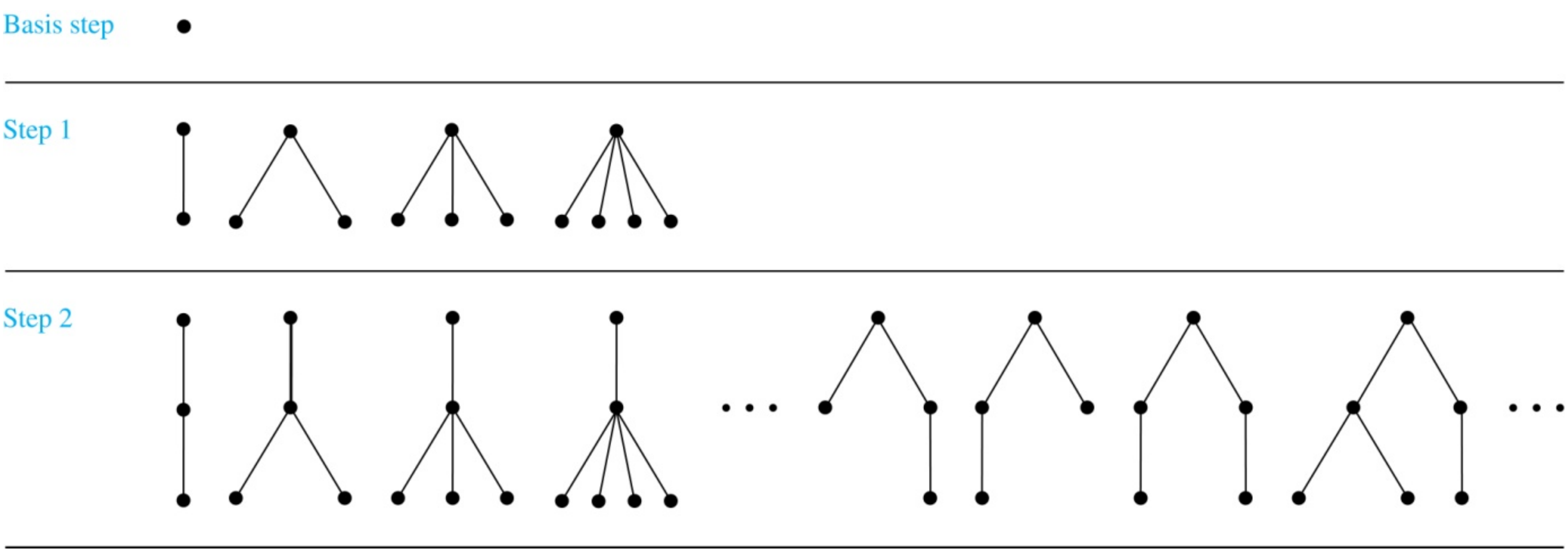
Rooted Trees

Definition: The set of *rooted trees,* where a rooted tree consists of a set of vertices containing a distinguished vertex called the *root*, and edges connecting these vertices, can be defined recursively by these steps:

BASIS STEP: A single vertex r is a rooted tree.

RECURSIVE STEP: Suppose that T_1, T_2, …,T_n are disjoint rooted trees with roots r_1, r_2,…,r_n, respectively. Then the graph formed by starting with a root r, which is not in any of the rooted trees T_1, T_2, …,T_n, and adding an edge from r to each of the vertices r_1, r_2,…,r_n, is also a rooted tree.

Building Up Rooted Trees



- Next we look at a special type of tree, the full binary tree.

Full Binary Trees

Definition: The set of *full binary trees* can be defined recursively by these steps.

BASIS STEP: There is a full binary tree consisting of only a single vertex *r*.

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.

Building Up Full Binary Trees