CSE 321  Discrete Structures

January 22, 2010
Lecture 08: Inductive Definitions
Recursive Definitions of Sets

• Recursive definition
  – Basis step: 0 ∈ S
  – Recursive step: if x ∈ S, then x + 2 ∈ S

What is the set S?

• Exclusion rule: Every element in S follows from basis steps and a finite number of recursive steps

Terminology: “Recursive definition” = “Inductive Definition”
Recursive Definitions of Sets

• Recursive definition
  – Basis step: \(7 \in S\)
  – Recursive step: if \(x \in S, x \in S\), then \(x - y \in S\)

• Note: here we allow arbitrary integers, positive and negative

What is the set \(S\) ?
Recursive Definitions of Sets

• Recursive definition
  – Basis step: $12 \in S$ and $21 \in S$
  – Recursive step: if $x \in S$, $x \in S$, then $x - y \in S$

What is the set $S$?
Strings

The set $\Sigma^*$ of strings over the alphabet $\Sigma$ is defined as follows:

• Basis: $\lambda \in \Sigma^*$ ($\lambda$ is the empty string)

• Recursive: if $w \in \Sigma^*$, $x \in \Sigma$, then $wx \in \Sigma^*$

Note: we sometimes write $\varepsilon$ for the empty string
Strings

• Example: $\Sigma = \{a, b, c\}$. What is $\Sigma^*$?

• $\Sigma^* = \{ \varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots \}$
Families of strings over $\Sigma = \{a, b\}$

- $L_1$
  - $\lambda \in L_1$
  - $w \in L_1$ then $awb \in L_1$

- What is $L_1$?
Families of strings over $\Sigma = \{a, b\}$

- $L_2$
  - $\lambda \in L_2$
  - $w \in L_2$ then $aw \in L_2$
  - $w \in L_2$ then $wb \in L_2$

- What is $L_2$?
Families of strings over $\Sigma = \{a, b\}$

- Think of $a$ as “(“ and of $b$ as “)”
- Define recursively the set $L_3$ of all well-formed parenthesis

- Strings that should be in $L_3$:
  - aaabbb, abababab, aabbabaaabbb, …
- Strings that should not be in $L_3$:
  - aab (too many a’s), ba (unmatched), abbaab (unmatched)
Recursive Function definitions

The length of a string: \( \textbf{Len} : \Sigma^* \rightarrow \text{Int} \)
\[
\text{Len}(\lambda) = 0; \\
\text{Len}(wx) = 1 + \text{Len}(w); \text{ for } w \in \Sigma^*, x \in \Sigma
\]

The concatenation of two strings: \( \textbf{Concat} : \Sigma^* \times \Sigma^* \rightarrow \Sigma^* \)
\[
\text{Concat}(w, \lambda) = w \text{ for } w \in \Sigma^* \\
\text{Concat}(w_1, w_2x) = \text{Concat}(w_1, w_2)x \text{ for } w_1, w_2 \text{ in } \Sigma^*, x \in \Sigma
\]
Well Formed Formulae

• $\Sigma = \{p, q, r, s, \ldots, T, F, \land, \lor, \rightarrow, \neg, (, )\}$
• Define Well-Formed-Formula for propositional logic
• Basis Step
  – $p, q, r, s, \ldots$ $T, F$ are in WFF
• Recursive Step
  – If $E$ and $F$ are in WFF then $(\neg E), (E \land F), (E \lor F), (E \rightarrow F)$ are in WFF
Well Formed Fomulae

Write recursive definitions on WFF for the following functions:

• Count the number of $\wedge$’s in the formula
• Test if a formula is positive, i.e. every atomic formula occurs under an even number of $\neg$ symbols (recall that $p \rightarrow q = \neg p \lor q$):
  
  $(\neg (\neg p \land \neg q)) \lor (s \land \neg \neg t)$ is positive
  
  $(\neg p \rightarrow s) \land t$ is positive
  
  $p \rightarrow s$ is not positive
Di-Graphs (Directed Graphs)

- Nodes: A, B, …
- Edges: A → B, …

- **Paths** from A to E:
  - A, B, E
  - A, B, D, E
  - A, B, F, A, B, F, A, B, E

- **Cycle**: A, B, F, A
A Directed Acyclic Graph (DAG) is a graph without cycles.

A tree is like this:
What is a “tree”?

• “A tree is a graph such that….”
  – How would you define a tree?
  – Want a tree to have a distinguished node, called the “root”
A Recursive Definition of Trees

• A graph with a single node $r$ is a tree and its root is $r$.

• If $t_1$, $t_2$, ..., $t_n$ are trees with roots $r_1$, $r_2$, ..., $r_n$, then the graph consisting of $t_1$, $t_2$, ..., $t_n$, a new node $r$, and $n$ edges $(r, r_i)$, $i=1, n$, is a tree and its root is $r$. 
Extended Binary Trees

- The empty graph is an extended binary tree.

- A nonempty extended binary tree has a root node $r$, with a left child $t_1$ and a right child $t_2$ s.t. both $t_1$ and $t_2$ are extended binary trees.
Subtle Distinction

In an extended binary tree we distinguish between the left child and the right child:

- Left child only
- Right child only
- Not an “extended” binary tree
Full binary trees

• Now we want to rule out the empty trees and empty subtrees: “full binary tree”

• How do we do this?
Extended Binary Trees

• The graph consisting of a single node is a full binary tree

• A nonempty full binary tree has a root node $r$, with a left child $t_1$ and a right child $t_2$ s.t. both $t_1$ and $t_2$ are full binary trees
Simplifying notation

- $(\cdot, T_1, T_2)$, tree with left subtree $T_1$ and right subtree $T_2$
- $\epsilon$ is the empty tree
- Extended Binary Trees (EBT)
  - $\epsilon \in EBT$
  - if $T_1, T_2 \in EBT$, then $(\cdot, T_1, T_2) \in EBT$
- Full Binary Trees (FBT)
  - $\cdot \in FBT$
  - if $T_1, T_2 \in FBT$, then $(\cdot, T_1, T_2) \in FBT$
Recursive Functions on Trees

- \( N(T) \) - number of vertices of \( T \)
- \( N(\varepsilon) = 0; \ N(\bullet) = 1 \)
- \( N(\bullet, T_1, T_2) = 1 + N(T_1) + N(T_2) \)

- \( Ht(T) \) – height of \( T \)
- \( Ht(\varepsilon) = 0; \ Ht(\bullet) = 1 \)
- \( Ht(\bullet, T_1, T_2) = 1 + \max(Ht(T_1), Ht(T_2)) \)

NOTE: Height definition differs from the text
Base case \( H(\bullet) = 0 \) used in text
More tree definitions: Fully balanced binary trees

- $\varepsilon$ is a FBBT.
- if $T_1$ and $T_2$ are FBBTs, with $Ht(T_1) = Ht(T_2)$, then $(\cdot, T_1, T_2)$ is a FBBT.
And more trees:
Almost balanced trees

- $\varepsilon$ is a ABT.

- If $T_1$ and $T_2$ are ABTs with $Ht(T_1) - 1 \leq Ht(T_2) \leq Ht(T_1) + 1$ then ($\bullet$, $T_1$, $T_2$) is a ABT.