COMP 1002

Logic for Computer Scientists

Lecture 25
Do the following two English sentences have the same parse trees?

– Time flies like an arrow.

– Fruit flies like an apple.
Recursive definitions of sets

• **Basis:** initial elements in the set
  a) Empty string is in S.
  b) Empty string, 0 and 1 are in S.

• **Recursive step:** a rule to make new elements in the set out of existing ones in S
  a) If \( w \in S \), then \( w0 \in S \) and \( w1 \in S \)
  b) If \( w_1 \in S \) and \( w_2 \in S \), then \( w_1 w_2 \in S \)

• **Restriction:** and nothing else is in S.
  – Nothing else is a binary string.

• Both examples define a set of all binary strings \( \{0,1\}^* \)
  – Recursive step b) only works with basis b).
  – Recursive step a) works with both basis a) and basis b).
Structural induction

• Let $S \subseteq U$ be a recursively defined set, and $F(x)$ is a property (of $x \in U$).

• Then
  – if all $x$ in the base of $S$ have the property,
  – and applying the recursion rules preserves the property,
  – then all elements in $S$ have the property.
Multiples of 3

• Let’s define a set $S$ of numbers as follows.
  – Base: $3 \in S$
  – Recursion: if $x, y \in S$, then $x + y \in S$
• Claim: all numbers in $S$ are divisible by 3
  – That is, $\forall x \in S \ \exists z \in \mathbb{N} \ x = 3z$.
• Proof (by structural induction).
  – Base case: $3$ is divisible by 3 ($z=1$).
  – Recursive step:
    • Let $x, y \in S$. Then $\exists z, u \in \mathbb{N} \ x = 3z \land y = 3u$. (inductive hypothesis)
    • Then $x + y = 3z + 3u = 3(z + u)$. (induction step)
    • Therefore, $x + y$ is divisible by 3.
  – As there are no other elements in $S$ except for those constructed from 3 by the recursion rule, all elements in $S$ are divisible by 3.
Trees

- In computer science, a **tree** is an undirected graph without cycles
  - **Undirected**: all edges go both ways, no arrows.
  - **Cycle**: sequence of edges going back to the same point.

- Recursive definition of trees:
  - Base: A single vertex \( v \) is a tree.
  - Recursion:
    - Let \( T \) be a tree, and \( v \) a new vertex.
    - Then a new tree consist of \( T, v \), and an edge (connection) between some vertex of \( T \) and \( v \).
  - Restriction:
    - Anything that cannot be constructed with this rule from this base is not a tree.
Binary trees

• **Rooted trees** are trees with a special vertex designated as a root.
  – Rooted trees are **binary** if every vertex has at most three edges: one going towards the root, and two going away from the root. **Full** if every vertex has either 2 or 0 edges going away from the root.

• **Recursive definition of full binary trees:**
  – Base: A single vertex \( \nu \) is a full binary tree with that vertex as a root.
  – Recursion:
    • Let \( T_1, T_2 \) be full binary trees with roots \( r_1, r_2 \), respectively. Let \( \nu \) be a new vertex.
    • A new full binary tree with root \( \nu \) is formed by connecting \( r_1 \) and \( r_2 \) to \( \nu \).
  – Restriction:
    • Anything that cannot be constructed with this rule from this base is not a full binary tree.
Height of a full binary tree

• The **height** of a rooted tree, $h(T)$, is the maximum number of edges to get from any vertex to the root.
  – Height of a tree with a single vertex is 0.
• Claim: Let $n(T)$ be the number of vertices in a full binary tree $T$. Then $n(T) \leq 2^{h(T)+1} - 1$
• Alternatively, height of a binary tree is at least $\log_2 n(T)$
  – If you have a recursive program that calls itself twice (e.g., within if ... then ... else ...)
  – Then if this code executes $n$ times (maybe on $n$ different cases)
  – Then the program runs in time at least $\log_2 n$, even when cases are checked in parallel.
Height of a full binary tree

• Claim: Let \( n(T) \) be the number of vertices in a full binary tree \( T \). Then \( n(T) \leq 2^{h(T)+1} - 1 \), where \( h(T) \) is the height of \( T \).

• Proof (by structural induction)
  – Base case: a tree with a single vertex has \( n(T) = 1 \) and \( h(T) = 0 \).
    • So \( 2^{h(T)+1} - 1 = 1 \geq 1 \)
  – Recursion: Suppose \( T \) was built by attaching \( T_1, T_2 \) to a new root vertex \( v \).
    • Number of vertices in \( T \) is \( n(T) = n(T_1) + n(T_2) + 1 \)
    • Every vertex in \( T_1 \) or \( T_2 \) now has one extra step to get to the new root in \( T \).
      – So \( h(T) = 1 + \max(h(T_1), h(T_2)) \)
    • By the induction hypothesis, \( n(T_1) \leq 2^{h(T_1)+1} - 1 \) and \( n(T_2) \leq 2^{h(T_2)+1} - 1 \)
    • \( n(T) = n(T_1) + n(T_2) + 1 \)
      \[ \leq 1 + (2^{h(T_1)+1} - 1) + (2^{h(T_2)+1} - 1) \]
      \[ \leq 2 \cdot \max(2^{h(T_1)+1}, 2^{h(T_2)+1}) - 1 \]
      \[ \leq 2 \cdot 2^{\max(h(T_1), h(T_2)) + 1} - 1 \]
      \[ = 2 \cdot 2^{h(T)} - 1 = 2^{h(T)+1} - 1 \]
  – Therefore, the number of vertices of any binary tree \( T \) is \( \leq 2^{h(T)+1} - 1 \)
Function growth.

• What does it mean to “grow” at a certain speed? How to compare growth rate of two functions?
  – Is \( f(n) = 100n \) larger than \( g(n) = n^2 \)?
    • For small \( n \), yes. For \( n > 100 \), not so much...
  – As usually program take longer on larger inputs, performance on larger inputs matters more.
  – Constant factors don’t matter that much.

• So to compare two functions, check which becomes larger as \( n \) increases (to infinity).
  – Ignoring constant factors, as they don’t contribute to the rate of growth.
Function growth.

• How to estimate the rate of growth?
  – Plotting a graph?

• Not quite conclusive:
  – How do you know what they will do past the graphed part?
O-notation.

- We say that $f(n)$ grows at most as fast as $g(n)$ if
  - There is a value $n_0$ such that after $n_0$, $f(n)$ is always at most as large as $g(n)$
    - More precisely, compare absolute values: $|g(n)|$ vs. $|f(n)|$
  - Moreover, ignore constant factors:
    - So if two functions only differ by a constant factor, consider them having the same growth rate.

- Denote set of all functions growing at most as fast as $g(n)$ by $O(g(n))$
  - Big-Oh of $g(n)$.
  - $g(n)$ is an asymptotic upper bound for $f(n)$.
  - When both $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$, write $f(n) \in \Theta(g(n))$
    - $f(n)$ is in big-Theta of $g(n)$.

- More generally, for real-valued functions $f(x)$ and $g(x)$,

\[
\text{if f(x) \in O(g(x)) then there exists } x_0 \geq 0 \text{ and } c > 0 \text{ such that for all } x \geq x_0 \text{,}
\]

\[
|f(x)| \leq c \cdot |g(x)|
\]

- That is, from some point $x_0$ on, each $|f(x)|$ is less than $|g(x)|$ (up to a constant factor).
- Usually in time complexity have functions $\mathbb{N} \rightarrow \mathbb{R}^\geq 0$, so use $n$ for $x$ and ignore $| |$. 
O-notation.

\[ f(n) \in O(g(n)) \text{ iff } \exists n_0 \in \mathbb{N} \exists c \in \mathbb{R}^+ \forall n \geq n_0 \ f(n) \leq c \cdot g(n) \]

- \( f(n) = n^2, g(n) = 2^n. \)
  - Take \( c = 1, n_0 = 4. \)
  - For every \( n \geq n_0, f(n) \leq g(n) \)
    - Proof by induction.
  - So \( n^2 \in O(2^n) \)

- \( f(n) = n^2, g(n) = 10n. \)
  - Take arbitrary \( c \) and look at \( n^2 \leq c \cdot 10n. \)
  - No matter what \( c \) is, when \( n > c \cdot 10, n^2 \geq c \cdot 10n \)
  - So \( n^2 \not\in O(10n). \)

- \( f(n) = n^2 + 100n, g(n) = 10n^2. \)
  - Here, \( f(n) \in O(g(n)) \) and also \( g(n) \in O(f(n)) \)
    - So \( f(n) \in \Theta(g(n)) \)
    - \( f(n) \in O(g(n)) \): \( c = 20 \) and/or \( n_0 = 100 \) work.
    - \( g(n) \in O(f(n)) \): Take \( c = 10, n_0 = 1 \).
  - Can ignore not only constants, but also all except the leading term in the expression.
Tower of Hanoi game

- Rules of the game:
  - Start with all disks on the first peg.
  - At any step, can move a disk to another peg, as long as it is not placed on top of a smaller disk.
  - Goal: move the whole tower onto the second peg.

- Question: how many steps are needed to move the tower of 8 disks? How about n disks?
Tower of Hanoi game

• Rules of the game:
  – Start with all disks on the first peg.
  – At any step, can move a disk to another peg, as long as it is not placed on top of a smaller disk.
  – Goal: move the whole tower onto the second peg.
• Question: how many steps are needed to move the tower of 8 disks? How about n disks?

• Let us call the number of moves needed to transfer n disks $H(n)$.
  – Names of pegs do not matter: from any peg $i$ to any peg $j \neq i$ would take the same number of steps.
• Basis: only one disk can be transferred in one step.
  – So $H(1) = 1$
• Recursive step:
  – Suppose we have $n-1$ disks. To transfer them all to peg 2, need $H(n-1)$ number of steps.
  – To transfer the remaining disk to peg 3, 1 step.
  – To transfer $n-1$ disks from peg 2 to peg 3 need $H(n-1)$ steps again.
  – So $H(n) = 2H(n-1)+1$ (recurrence).
• Closed form: $H(n) = 2^n - 1$. 
Recurrence relations

• **Recurrence:** an equation that defines an $n^{th}$ element in a sequence in terms of one or more of previous terms.
  - Think of $F(n) = s_n$ for some sequence $\{s_n\}$
    - $H(n) = 2H(n - 1) + 1$
    - $F(n) = F(n - 1) + F(n - 2)$

• **A closed form** of a recurrence relation is an expression that defines an $n^{th}$ element in a sequence in terms of $n$ directly.
  - Often use recurrence relations and their closed forms to describe performance of (especially recursive) algorithms.
Closed forms of some sequences

• Arithmetic progression:
  – Sequence: \( c, c + d, c + 2d, c + 3d, \ldots, c + nd, \ldots \)
  – **Closed form**: \( s_n = c + nd \)
    • Closed forms are very useful for analysis of recursive programs, etc.

• Geometric progression:
  – Sequence: \( c, cr, cr^2, cr^3, \ldots, cr^n, \ldots \)
  – **Closed form**: \( s_n = c \cdot r^n \)

• Fibonacci sequence: \( F(n)=F(n-1)+F(n-2) \)
  – Sequence: 1, 1, 2, 3, 5, 8, 13, ...
  – **Closed form**: \( F_n = \frac{\phi^n-(1-\phi)^n}{\sqrt{5}} \)
    • Where \( \phi \) ("phi") is the “golden ratio”: a ratio such that \( \frac{a+b}{a} = \frac{a}{b} \)
    • \( \phi = \frac{1+\sqrt{5}}{2} \)
Solving recurrences

• Solving a recurrence: finding a closed form.
  – Solving the recurrence $H(n)=2H(n-1)+1$
    • $H(n) = 2 \cdot H(n - 1) + 1$
      
      $= 2(2H(n - 2) + 1) + 1 = 2^2 H(n - 2) + 2 + 1$
      $= 2^3 H(n - 3) + 2^2 + 2 + 1$
      $= 2^4 H(n - 4) + 2^3 + 2^2 + 2 + 1 \ldots$

  – Closed form: $H(n) = \sum_{i=0}^{n-1} 2^i = 2^n - 1$
    • Proof by induction (coming in the next lecture).
    • Or by noticing that a binary number 111...1 plus 1 gives a binary number 10000...0

  – So adding one more disk doubles the number of steps.
    • We say that the function defined by $H(n)$ grows exponentially

• Solving recurrences in general might be tricky.
  – When the recurrence is of the form $T(n)=a \cdot T(n/b)+f(n)$, there is a general method to estimate the growth rate of a function defined by the recurrence
    • Called the Master Theorem for recurrences.
Puzzle: chocolate squares

• Suppose you have a piece of chocolate like this:

• How many squares are in it? – of all sizes, from single to the whole thing