Today's topics

- Recursion
  - Recursively defined functions
  - Recursively defined sets
  - Structural Induction
- Reading: Sections 3.4
- Upcoming
  - Counting

§3.4: Recursive Definitions

- In induction, we *prove* all members of an infinite set satisfy some predicate P by:
  - proving the truth of the predicate for larger members in terms of that of smaller members.
- In *recursive definitions*, we similarly *define* a function, a predicate, a set, or a more complex structure over an infinite domain (universe of discourse) by:
  - defining the function, predicate value, set membership, or structure of larger elements in terms of those of smaller ones.
- In *structural induction*, we inductively prove properties of recursively-defined objects in a way that parallels the objects' own recursive definitions.

Recursion

- *Recursion* is the general term for the practice of defining an object in terms of *itself*
  - or of part of itself
  - This may seem circular, but it isn't necessarily.
- An inductive proof establishes the truth of $P(n+1)$ *recursively* in terms of $P(n)$.
- There are also recursive *algorithms*, *definitions*, *functions*, *sequences*, *sets*, and other structures.

Recursively Defined Functions

- Simplest case: One way to define a function $f:\mathbf{N}\rightarrow S$ (for any set S) or series $a_n=f(n)$ is to:
  - Define $f(0)$.
  - For $n>0$, define $f(n)$ in terms of $f(0),\ldots,f(n-1)$.
- *E.g.*: Define the series $a_n :\equiv 2^n$ recursively:
  - Let $a_0 :\equiv 1$.
  - For $n>0$, let $a_n :\equiv 2a_{n-1}$.

Another Example

- Suppose we define $f(n)$ for all $n \in \mathbf{N}$ recursively by:
  - Let $f(0)=3$
  - For all $n \in \mathbf{N}$, let $f(n+1)=2f(n)+3$
- What are the values of the following?
  - $f(1)=9$ $f(2)=21$ $f(3)=45$ $f(4)=93$

Recursive definition of Factorial

- Give an inductive (recursive) definition of the factorial function,

  $F(n) :\equiv n! :\equiv \prod_{1 \le i \le n} i = 1 \cdot 2 \cdot \ldots \cdot n.$
  - Base case: $F(0) :\equiv 1$
  - Recursive part: $F(n) :\equiv n \cdot F(n-1)$.
    - $F(1)=1$
    - $F(2)=2$
    - $F(3)=6$

More Easy Examples

- Write down recursive definitions for:

  $i+n$ (i integer, n natural) using only $s(i) = i+1$.

  $a \cdot n$ (a real, n natural) using only addition

  a^n (a real, n natural) using only multiplication

  $\sum_{0 \le i \le n} a_i$ (for an arbitrary series of numbers $\{a_i\}$)

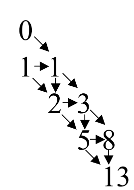
  $\prod_{0 \le i \le n} a_i$ (for an arbitrary series of numbers $\{a_i\}$)

  $\bigcap_{0 \le i \le n} S_i$ (for an arbitrary series of sets $\{S_i\}$)

The Fibonacci Series

- The *Fibonacci series* $f_{n \ge 0}$ is a famous series defined by:

  $$f_0 :\equiv 0, \quad f_1 :\equiv 1, \quad f_{n \ge 2} :\equiv f_{n-1} + f_{n-2}$$



Leonardo Fibonacci
1170-1250

Inductive Proof about Fib. series

- **Theorem:** $f_n < 2^n$. $\longleftarrow$ Implicitly for all $n\in\mathbf{N}$
- **Proof:** By induction.

Base cases: $f_0 = 0 < 2^0 = 1$, $f_1 = 1 < 2^1 = 2$ — Note use of base cases of recursive def'n.

Inductive step: Use 2nd principle of induction (strong induction). Assume $\forall k<n,\ f_k < 2^k$. Then $f_n = f_{n-1} + f_{n-2}$ is $< 2^{n-1} + 2^{n-2} < 2^{n-1} + 2^{n-1} = 2^n$. ■

A lower bound on Fibonacci series

- **Theorem.** For all integers $n \geq 3$, $f_n > \alpha^{n-2}$, where $\alpha = (1+5^{1/2})/2 \approx 1.61803$.
- **Proof.** (Using strong induction.)
  - Let $P(n) = (f_n > \alpha^{n-2})$.
  - **Base cases:** For n=3, note that $\alpha < 2 = f_3$. For n=4, $\alpha^2 = (1+2\cdot 5^{1/2}+5)/4 = (3+5^{1/2})/2 \approx 2.61803 < 3 = f_4$.
  - **Inductive step:** For $k\geq 4$, assume $P(j)$ for $3\leq j\leq k$, prove $P(k+1)$. Note $\alpha^2 = \alpha+1$. Thus, $\alpha^{k-1} = (\alpha+1)\alpha^{k-3} = \alpha^{k-2} + \alpha^{k-3}$. By inductive hypothesis, $f_{k-1} > \alpha^{k-3}$ and $f_k > \alpha^{k-2}$. So, $f_{k+1} = f_k + f_{k-1} > \alpha^{k-2} + \alpha^{k-3} = \alpha^{k-1}$. Thus $P(k+1)$. ■

Lamé's Theorem

- **Theorem:** $\forall a,b\in\mathbf{N}$, $a\geq b>0$, the number of steps in Euclid's algorithm to find gcd(a,b) is $\leq 5k$, where $k = \lfloor\log_{10} b\rfloor+1$ is the number of decimal digits in b.
  - Thus, Euclid's algorithm is linear-time in the number of digits in b.
- **Proof:**
  - Uses the Fibonacci sequence!
  - See next 2 slides.

Gabriel Lamé
1795-1870

Proof of Lamé's Theorem

- Consider the sequence of division-algorithm equations used in Euclid's alg.:

$r_0 = r_1q_1 + r_2$ with $0 \leq r_2 < r_1$
$r_1 = r_2q_2 + r_3$ with $0 \leq r_3 < r_2$
…
$r_{n-2} = r_{n-1}q_{n-1} + r_n$ with $0 \leq r_n < r_{n-1}$
$r_{n-1} = r_nq_n + r_{n+1}$ with $r_{n+1} = 0$ (terminate)

Where $a = r_0$, $b = r_1$, and gcd(a,b)=r_n.

- The number of divisions (iterations) is n.

Continued on next slide…

Lamé Proof, continued

- Since $r_0 \geq r_1 > r_2 > \ldots > r_n$, each quotient $q_i \equiv \lfloor r_{i-1}/r_i \rfloor \geq 1$.
- Since $r_{n-1} = r_n q_n$ and $r_{n-1} > r_n$, $q_n \geq 2$.
- So we have the following relations between r and f:

  $r_n \geq 1 = f_2$
  $r_{n-1} \geq 2r_n \geq 2f_2 = f_3$
  $r_{n-2} \geq r_{n-1} + r_n \geq f_2 + f_3 = f_4$
  …
  $r_2 \geq r_3 + r_4 \geq f_{n-1} + f_{n-2} = f_n$
  $b = r_1 \geq r_2 + r_3 \geq f_n + f_{n-1} = f_{n+1}$.
- Thus, if n>2 divisions are used, then $b \geq f_{n+1} > \alpha^{n-1}$.
  - Thus, $\log_{10} b > \log_{10}(\alpha^{n-1}) = (n-1)\log_{10}\alpha \approx (n-1)0.208 > (n-1)/5$.
  - If b has k decimal digits, then $\log_{10} b < k$, so $n-1 < 5k$, so $n \leq 5k$.

Recursively Defined Sets

- An infinite set S may be defined recursively, by giving:
  - A small finite set of *base* elements of S.
  - A rule for constructing new elements of S from previously-established elements.
  - Implicitly, S has no other elements but these.
- **Example:** Let $3 \in S$, and let $x+y \in S$ if $x,y \in S$. What is S?

The Set of All Strings

- Given an alphabet Σ, the set Σ^* of all strings over Σ can be recursively defined by:

  $\varepsilon \in \Sigma^*$ ($\varepsilon :\equiv$ "", the empty string) Book uses λ

  $w \in \Sigma^* \wedge x \in \Sigma \rightarrow wx \in \Sigma^*$
- **Exercise:** Prove that this definition is equivalent to our old one: $$\Sigma^* :\equiv \bigcup_{n \in \mathbf{N}} \Sigma^n$$

Other Easy String Examples

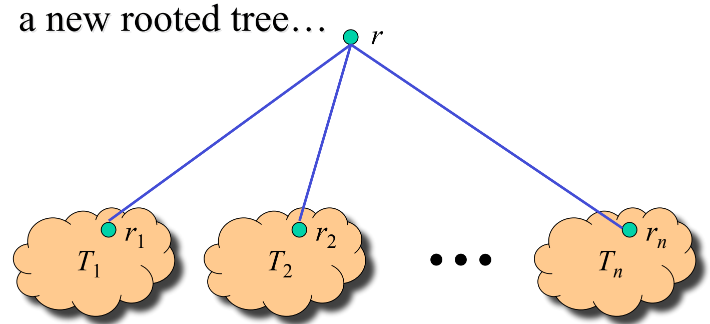
- Give recursive definitions for:
  - The concatenation of strings $w_1 \cdot w_2$.
  - The length $\ell(w)$ of a string w.
  - Well-formed formulae of propositional logic involving **T**, **F**, propositional variables, and operators in $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$.
  - Well-formed arithmetic formulae involving variables, numerals, and ops in $\{+, -, *, \uparrow\}$.

Rooted Trees

- Trees will be covered in CompSci 130.
  - Briefly, a tree is a graph in which there is exactly one undirected path between each pair of nodes.
  - An undirected graph can be represented as a set of unordered pairs (called *arcs*) of objects called *nodes*.
- Definition of the set of rooted trees:
  - Any single node r is a rooted tree.
  - If $T_1, \ldots, T_n$ are disjoint rooted trees with respective roots $r_1, \ldots, r_n$, and r is a node not in any of the T_i's, then another rooted tree is $\{\{r,r_1\},\ldots,\{r,r_n\}\}\cup T_1\cup\ldots\cup T_n$.

Illustrating Rooted Tree Def'n.

- How rooted trees can be combined to form a new rooted tree…



Draw some examples…

Extended Binary Trees

- A special case of rooted trees.
- Recursive definition of EBTs:
  - The empty set $\varnothing$ is an extended binary tree.
  - If T_1,T_2 are disjoint EBTs, then $e_1\cup e_2\cup T_1\cup T_2$ is an EBT, where $e_1 = \varnothing$ if $T_1 = \varnothing$, and $e_1 = \{(r,r_1)\}$ if $T_1 \neq \varnothing$ and has root r_1, and similarly for e_2.

Draw some examples…

Full Binary Trees

- A special case of extended binary trees.
- Recursive definition of FBTs:
  - A single node r is a full binary tree.
    - Note this is different from the EBT base case.
  - If T_1,T_2 are disjoint FBTs, then $e_1\cup e_2\cup T_1\cup T_2$, where $e_1 = \varnothing$ if $T_1 = \varnothing$, and $e_1 = \{(r,r_1)\}$ if $T_1 \neq \varnothing$ and has root r_1, and similarly for e_2.
    - Note this is the same as the EBT recursive case!
      - Can simplify it to "If T_1,T_2 are disjoint FBTs with roots r_1 and r_2, then $\{(r, r_1),(r,r_2)\}\cup T_1\cup T_2$ is an FBT."

Draw some examples…

Structural Induction

- Proving something about a recursively defined object using an inductive proof whose structure mirrors the object's definition.
- **Example problem:** Let $3 \in S$, and let $x+y \in S$ if $x,y \in S$. Show that $S = \{n \in \mathbf{Z}^+ \mid (3|n)\}$ (the set of positive multiples of 3).

Example continued

- Let $3 \in S$, and let $x+y \in S$ if $x,y \in S$. Let $A = \{n \in \mathbf{Z}^+ \mid (3|n)\}$.
- **Theorem:** $A=S$. **Proof:** We show that $A \subseteq S$ and $S \subseteq A$.
  - To show $A \subseteq S$, show $[n \in \mathbf{Z}^+ \wedge (3|n)] \rightarrow n \in S$.
    - **Inductive proof.** Let $P(n) :\equiv n \in S$. Induction over positive multiples of 3. Base case: $n=3$, thus $3 \in S$ by def'n. of S. Inductive step: Given $P(n)$, prove $P(n+3)$. By inductive hyp., $n \in S$, and $3 \in S$, so by def'n of S, $n+3 \in S$.
  - To show $S \subseteq A$: let $n \in S$, show $n \in A$.
    - **Structural inductive proof.** Let $P(n) :\equiv n \in A$. Two cases: $n=3$ (base case), which is in A, or $n=x+y$ (recursive step). We know x and y are positive, since neither rule generates negative numbers. So, $x<n$ and $y<n$, and so we know x and y are in A, by strong inductive hypothesis. Since $3|x$ and $3|y$, we have $3|(x+y)$, thus $x+y \in A$.