Discrete Mathematics (2009 Spring)
Induction and Recursion (Chapter 4, 3 hours)

Chih-Wei Yi

Dept. of Computer Science
National Chiao Tung University

April 17, 2009

§4.1 Mathematical Induction

§4.1 Mathematical Induction

- A powerful, rigorous technique for proving that a predicate $P(n)$ is true for every natural number n, no matter how large.
- Essentially a "domino effect" principle.
- Based on a predicate-logic inference rule:

$$\begin{array}{rcl} \text{BasisStep} & : & P(0) \\ \text{InductiveStep} & : & \forall n \geq 0,\ (P(n) \rightarrow P(n+1)) \\ \hline \text{Conclusion} & : & \therefore \forall n \geq 0,\ P(n) \end{array}$$

- The First Principle of Mathematical Induction.

Outline of an Inductive Proof

- Want to prove $\forall n\ P(n) \ldots$
  1. *Base case* (or basis step): **Prove** $P(0)$.
  2. *Inductive step*: **Prove** $\forall n \geq 0\ P(n) \rightarrow P(n+1)$. E.g. use a direct proof:
     - Let $n \in \mathbb{N}$, assume $P(n)$. (*inductive hypothesis*)
     - Under this assumption, prove $P(n+1)$.
  3. Inductive inference rule then gives $\forall n\ P(n)$.

Example

Prove that the sum of the first n odd positive integers is n^2. That is, prove:

$$\forall n \geq 1 : \underbrace{\sum_{i=1}^{n} (2i-1) = n^2}_{P(n)}$$

Proof.

- **Base Case**: Let $n = 1$. The sum of the first 1 odd positive integer is 1 which equals 1^2.
- **Inductive Step**: Prove $\forall n \geq 1 : P(n) \to P(n+1)$. Let $n \geq 1$, assume $P(n)$, and prove $P(n+1)$.

$$\begin{aligned} \sum_{i=1}^{n+1}(2i-1) &= \left(\sum_{i=1}^{n}(2i-1)\right) + (2(n+1)-1) \\ &= n^2 + 2n + 1 \\ &= (n+1)^2 \end{aligned}$$

- So, **according to the inductive inference rule**, the property is proved.

□

Generalizing Induction

- Can also be used to prove $\forall n \geq c\ P(n)$ for a given constant $c \in \mathbb{Z}$, where maybe $c \neq 0$.
  - In this circumstance, the base case is to prove $P(c)$ rather than $P(0)$, and the inductive step is to prove $\forall n \geq c\ (P(n) \rightarrow P(n+1))$.
- Induction can also be used to prove $\forall n \geq c\ P(a_n)$ for an arbitrary series $\{a_n\}$.
- Can reduce these to the form already shown.

Example

Prove that $\forall n > 0,\ n < 2^n$.

Solution

Let $P(n) = (n < 2^n)$.

1. *Base case:* $P(1) = (1 < 2^1) = (1 < 2) = \mathbf{T}$.
2. *Inductive step: For* $n > 0$*, prove* $P(n) \rightarrow P(n+1)$.
   Assuming $n < 2^n$*, prove* $n + 1 < 2^{n+1}$*. Note*

$$\begin{aligned} n+1 &< 2^n + 1 \text{ (by inductive hypothesis)} \\ &< 2^n + 2^n \text{ (because } 1 < 2 = 2 \cdot 2^0 \leq 2 \cdot 2^{n-1} = 2^n\text{)} \\ &= 2^{n+1} \end{aligned}$$

3. *So* $n + 1 < 2^{n+1}$*, and we're done.*

Harmonic Numbers

Theorem

The harmnoic numbers H_j for $j = 1, 2, 3, \cdots$ are

$$H_j = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{j}.$$

Then, $H_{2^n} \geq 1 + \frac{n}{2}$.

Proof.

BASIS STEP
INDUCTIVE STEP
CONCLUSION □

De Morgan's Law

Theorem

If $A_1, A_2, \cdots, A_n$ are subsets of a universal set U and $n \geq 2$, then we have

$$\overline{\bigcap_{j=1}^{n} A_j} = \bigcup_{j=1}^{n} \overline{A_j} \text{ and } \overline{\bigcup_{j=1}^{n} A_j} = \bigcap_{j=1}^{n} \overline{A_j}.$$

Proof.

BASIS STEP
INDUCTIVE STEP
CONCLUSION $\square$

The Inclusion-Exclusion Principle

Theorem

If $A_1, A_2, \cdots, A_n$ are finite sets and $n \geq 1$, we have

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1} |A_i| - \sum_{1\leq i<j\leq n} |A_i \cap A_j| + \sum_{1\leq i<j<k\leq n} |A_i \cap A_j \cap A_k| - \cdots .$$

Proof.

BASIS STEP
INDUCTIVE STEP
CONCLUSION □

§4.2 Strong Induction and Well-Ordering

§4.2: Strong Induction Second Principle of Induction

- Characterized by another inference rule:

$$\begin{array}{c} P(0) \\ \forall n \geq 0 : \overbrace{(\forall 0 \leq k \leq n \; P(k))}^{P \text{ is true in all previous cases}} \rightarrow P(n+1) \\ \hline \therefore \quad \forall n \geq 0 : P(n) \end{array}$$

- Difference with the 1st principle is that the inductive step uses the fact that $P(k)$ is true for all smaller $k < n+1$, not just for $k = n$.

Example of Second Principle

Example

Show that every $n > 1$ can be written as a product $p_1 p_2 \ldots p_s$ of some series of s prime numbers.

Solution

Let $P(n) =$ "n has that property".

1. *Base case: For $n = 2$, let $s = 1$, $p_1 = 2$.*
2. *Inductive step: Let $n \geq 2$. Assume $\forall 2 \leq k \leq n$: $P(k)$. Consider $n + 1$. If prime, let $s = 1$, $p_1 = n + 1$. Else $n + 1 = ab$, where $1 < a \leq n$ and $1 < b \leq n$. Then $a = p_1 p_2 \ldots p_t$ and $b = q_1 q_2 \ldots q_u$. Then $n + 1 = p_1 p_2 \ldots p_t q_1 q_2 \ldots q_u$, a product of $s = t + u$ primes.*
3. *So, $P(n)$ is true for all $n > 1$.*

Another 2nd Principle Example

Example

Prove that every amount of postage of 12 cents or more can be formed using just 4-cent and 5-cent stamps.

Solution

Let $P(n) =$ *"n can be formed using 4-cent and 5-cent stamps."*

1. *Base case:* $12 = 3(4)$, $13 = 2(4) + 1(5)$, $14 = 1(4) + 2(5)$, $15 = 3(5)$, *so* $\forall 12 \leq n \leq 15$, $P(n)$.
2. *Inductive step: Let* $n \geq 15$. *Assume* $\forall 12 \leq k \leq n \; P(k)$. *Note* $n + 1 = (n - 3) + 4$ *and* $12 \leq n - 3 \leq n$. *Since* $P(n-3)$, *add a 4-cent stamp to get postage for* $n + 1$.
3. *So,* $P(n)$ *is true for all* $n \geq 12$.

Examples

1. Consider a game in which two players take turns removing any positive number of matches they want from one of two piles of matches. The player who removes the last match wins the game. Show that if the two piles contain the same number of matches initially, the second player can always guarantee a win.
2. A simple polygon with n sides, where n is an integer with $n \geq 3$, can be triangulated into $n - 2$ triangles.

Lemma (For the 2nd example)

Every simple polygon has an interior diagonal.

Validity of Induction

- Proof that $\forall k \geq 0 \; P(k)$ is a valid consequent:
  - Given any $k \geq 0$,
  - $\forall n \geq 0 \; (P(n) \rightarrow P(n+1))$ (antecedent 2) trivially implies $\forall n \geq 0 \; (n < k) \rightarrow (P(n) \rightarrow P(n+1))$, or $(P(0) \rightarrow P(1)) \wedge (P(1) \rightarrow P(2)) \wedge \ldots \wedge (P(k-1) \rightarrow P(k))$.
- Repeatedly applying the hypothetical syllogism rule to adjacent implications $k-1$ times then gives $P(0) \rightarrow P(k)$.
- $P(0)$ (antecedent $\#1$) and modus ponens give $P(k)$. Thus $\forall k \geq 0 \; P(k)$.

§4.3 Recursive Definitions and Structural Induction

Recursive Definitions

- In induction, we *prove* all members of an infinite set have some property P by proving the truth for larger members in terms of that of smaller members.
- In *recursive definitions*, we similarly define a function, a predicate or a set over an infinite number of elements by defining the function or predicate value or set-membership of larger elements in terms of that of smaller ones.

Recursion

- Recursion is a general term for the practice of defining an object in terms of *itself* (or of part of itself).
- An inductive proof establishes the truth of $P(n+1)$ recursively in terms of $P(n)$.
- There are also recursive *algorithms*, *definitions*, *functions*, *sequences*, and *sets*.

Recursively Defined Functions

- Simplest case: One way to define a function $f : \mathbb{N} \to S$ (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 $n \in \mathbb{N}$ recursively by:
  - Let $f(0) = 3$.
  - For all $n \in \mathbb{N}$, 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 definition of the factorial function $F(n) :\equiv n! :\equiv 2 \cdot 3 \cdot \ldots \cdot n$.
  - Base case: $F(0) :\equiv 1$
  - Recursive part: $F(n) :\equiv n \cdot F(n-1)$.
    - $F(1) = 1$
    - $F(2) = 2$
    - $F(3) = 6$

The Fibonacci Series

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

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

$$\begin{tikzcd}
0 \arrow[dr] & & & \\
1 \arrow[r] \arrow[dr] & 1 \arrow[d] \arrow[dr] & & \\
& 2 \arrow[r] \arrow[dr] & 3 \arrow[d] \arrow[dr] & \\
& & 5 \arrow[r] \arrow[dr] & 8 \arrow[d] \\
& & & 13
\end{tikzcd}$$

Inductive Proof about Fib. Series

Theorem

For all $n \in \mathbb{N}$, $f_n < 2^n$.

Proof.

Prove the theorem by induction.

- *Base cases:* $\left.\begin{array}{l} f_0 = 0 < 2^0 = 1 \\ f_1 = 1 < 2^1 = 2 \end{array}\right\}$ *the base cases of recursive def'n*
- *Inductive step: Use the 2nd principle of induction (strong induction). Assume $\forall k < n$, $f_k < 2^k$. Then*

$$\begin{aligned} f_n &= f_{n-1} + f_{n-2} \\ &< 2^{n-1} + 2^{n-2} < 2^{n-1} + 2^{n-1} = 2^n. \end{aligned}$$

- *So, $f_n < 2^n$ is proved.*

$\square$

Exercise

Problem

Let f_n denote the Fibonacci numbers. Show that whenever $n \geq 3$, $f_n > \alpha^{n-2}$, where $\alpha = \frac{(1+\sqrt{5})}{2}$.

Problem (**Lamé's Theorem**)

Let a and b be positive integers with $a \geq b$. Then the number of divisions used by the Euclidean algorithm to find $\gcd(a, b)$ is less than or equal to five times the number of decimal digits in b.

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 $\sum$, the set $\sum^*$ of all strings over $\sum$ can be recursively defined as

$$\begin{cases} \varepsilon \in \Sigma^* \ (\varepsilon :\equiv \text{``''}, \text{ the empty string}) \\ (\lambda \in \Sigma^*) \wedge (x \in \Sigma) \to \lambda x \in \Sigma^* \end{cases}$$

Problem

Prove that this definition is equivalent to our old one

$$\Sigma^* = \bigcup_{n \in \mathbb{N}} \Sigma^n .$$

Tree Structures — Rooted Trees

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

Examples of Rooted Trees

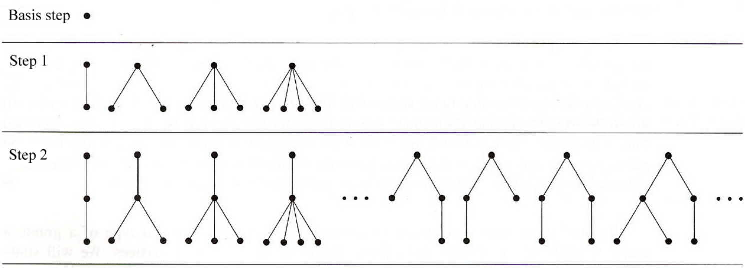

FIGURE 2 **Building Up Rooted Trees.**

Tree Structures — Binary Trees

- The set of extended binary trees can be defined recursively by these steps:
  - BASIS STEP: A empty set is an extended binary tree.
  - RECURSIVE STEP: If T_1 and T_2 are disjoint extended binary trees, there is an extended binary tree, denoted by $T_1 \cdot T_2$, containing 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 when these trees are nonempty.

Example of Extended Binary Trees

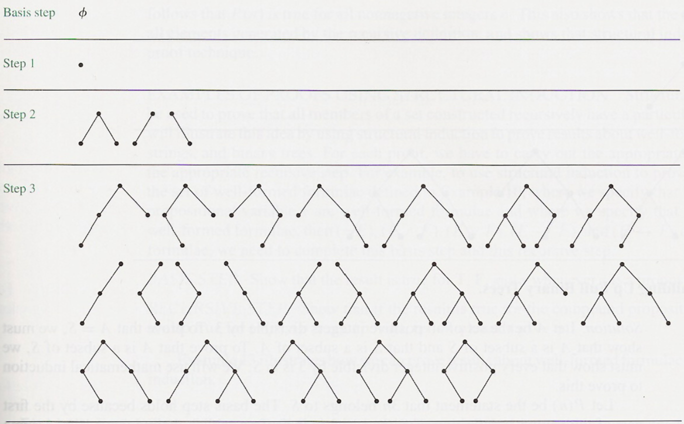

FIGURE 3 Building Up Extended Binary Trees.

§4.4 Recursive Algorithms

§4.4 Recursive Algorithms

- Recursive definitions can be used to describe algorithms as well as functions and sets.

Example (A procedure to compute a^n)

procedure *power* $(a \neq 0 : \text{real}, n \in \mathbb{N})$
    if $n = 0$ **then return** 1
    else return $a \cdot power\,(a, n-1)$

Efficiency of Recursive Algorithms

- The time complexity of a recursive algorithm may depend critically on the number of recursive calls it makes.
- E.g., *modular exponentiation* to a power n can take $\log(n)$ time if done right, but linear time if done slightly differently.

$$\boxed{\text{Compute } b^n \bmod m, \text{ where } m \geq 2,\ n \geq 0, \text{ and } 1 \leq b < m.}$$

Modular Exponentiation Alg. #1

- Use the fact that $b^n = b \cdot b^{n-1}$ and that $x \cdot y \bmod m = x \cdot (y \bmod m) \bmod m$. (Prove!!!)

Example (Returns $b^n \bmod m$ by using $b^n = b \cdot b^{n-1}$.)

procedure *mpower* $(b \geq 1, n \geq 0, m > b \in \mathbb{N})$
  if $n = 0$ **then return** 1 **else**
  return $(b \cdot mpower(b, n-1, m)) \bmod m$

- Note this algorithm takes $\Theta(n)$ steps!

Modular Exponentiation Alg. #2

- Use the fact that $b^{2k} = b^{k \cdot 2} = (b^k)^2$.

Example (Returns $b^n \bmod m$ by using $b^{2k} = (b^k)^2$.)

procedure *mpower* (b, n, m)
  if $n = 0$ **then return** 1
  else if $2 \mid n$ **then**
    return $mpower\,(b, n/2, m)^2 \bmod m$
  else return $(mpower\,(b, n-1, m) \cdot b) \bmod m$

- What is its time complexity?

Modular Exponentiation Alg. #2

- Use the fact that $b^{2k} = b^{k \cdot 2} = (b^k)^2$.

Example (Returns $b^n \bmod m$ by using $b^{2k} = (b^k)^2$.)

procedure *mpower* (b, n, m)
  if $n = 0$ **then return** 1
  else if $2 \mid n$ **then**
    return *mpower* $(b, n/2, m)^2 \bmod m$
  else return $(\mathit{mpower}\,(b, n-1, m) \cdot b) \bmod m$

- What is its time complexity?
  - $\Theta(\log n)$ steps

A Slight Variation

Example

procedure *mpower*(b, n, m)
  if $n = 0$ **then return** 1
  else if $2 | n$ **then**
    return $\left(\begin{array}{c} mpower\left(b, n/2, m\right) \cdot \\ mpower\left(b, n/2, m\right) \end{array} \right) \bmod m$
  else return $\left(mpower\left(b, n-1, m\right) \cdot b\right) \bmod m$

- Nearly identical but takes $\Theta\left(n\right)$ time instead!

The number of recursive calls made is critical.

Recursive Euclid's Algorithm

Example (Recursive Euclid's Algorithm)

procedure gcd $(a, b \in N)$
    if $a = 0$ **then return** b
    else return gcd $(b \bmod a, a)$

- Note recursive algorithms are often simpler to code than iterative ones. . .
- However, they can consume more stack space, if your compiler is not smart enough.

Merge Sort

Example (Merge Sort)

procedure $sort\left(L = \ell_1, \ldots, \ell_n\right)$
  if $n > 1$ **then**
    $m = \lfloor n/2 \rfloor$ // this is rough $\frac{1}{2}$-way point
    $L = merge\left(sort\left(\ell_1, \ldots, \ell_m\right), sort(\ell_{m+1}, \ldots, \ell_n)\right)$
  return L

- The merge takes $\Theta\left(n\right)$ steps, and merge-sort takes $\Theta\left(n \log n\right)$.

Example (Merge routine)

procedure *merge* $(A, B$: sorted lists$)$

$L =$ empty list
$i = 0; j = 0; k = 0;$
while $i < |A| \vee j < |B|$ // $|A|$ is length of A
if $i == |A|$ **then** $L_k = B_j; j = j + 1$
else if $j == |B|$ **then** $L_k = A_i;\ i = i + 1$
else if $A_i < B_j$ **then** $L_k = A_i;\ i = i + 1$
else $L_k = B_j; j = j + 1$
$k = k + 1$
return L

Ackermann's Function

Problem

Find the value $A(1,0)$, $A(0,1)$, $A(1,1)$ *,and* $A(2,2)$ *according to the following recursive definition*

$$A(m,n) = \begin{cases} 2n & \text{if } m = 0 \\ 0 & \text{if } m \geq 1 \text{ and } n = 0 \\ 2 & \text{if } m \geq 1 \text{ and } n = 1 \\ A(m-1, A(m, n-1)) & \text{if } m \geq 1 \text{ and } n \geq 2 \end{cases}.$$