14:30 10/6/2009

Chapter 4

Induction and Recursion

4.1 MATHEMATICAL INDUCTION

From modus ponens:

$$\begin{array}{cl} p & \text{basis assertion} \\ p \rightarrow q & \text{conditional assertion} \\ \hline q & \text{conclusion} \end{array}$$

we can easily derive "double modus ponens":

$$\begin{array}{cl} p_0 & \text{basis assertion} \\ p_0 \rightarrow p_1 & \text{conditional assertion} \\ p_1 \rightarrow p_2 & \text{conditional assertion} \\ \hline p_2 & \text{conclusion} \end{array}$$

We might also derive triple modus ponens, quadruple modus ponens, and so on. Thus, we have no trouble proving assertions about arbitrarily large integers. For instance,

The initial domino falls.
If any of the first 999 dominoes falls,
then so does its successor.

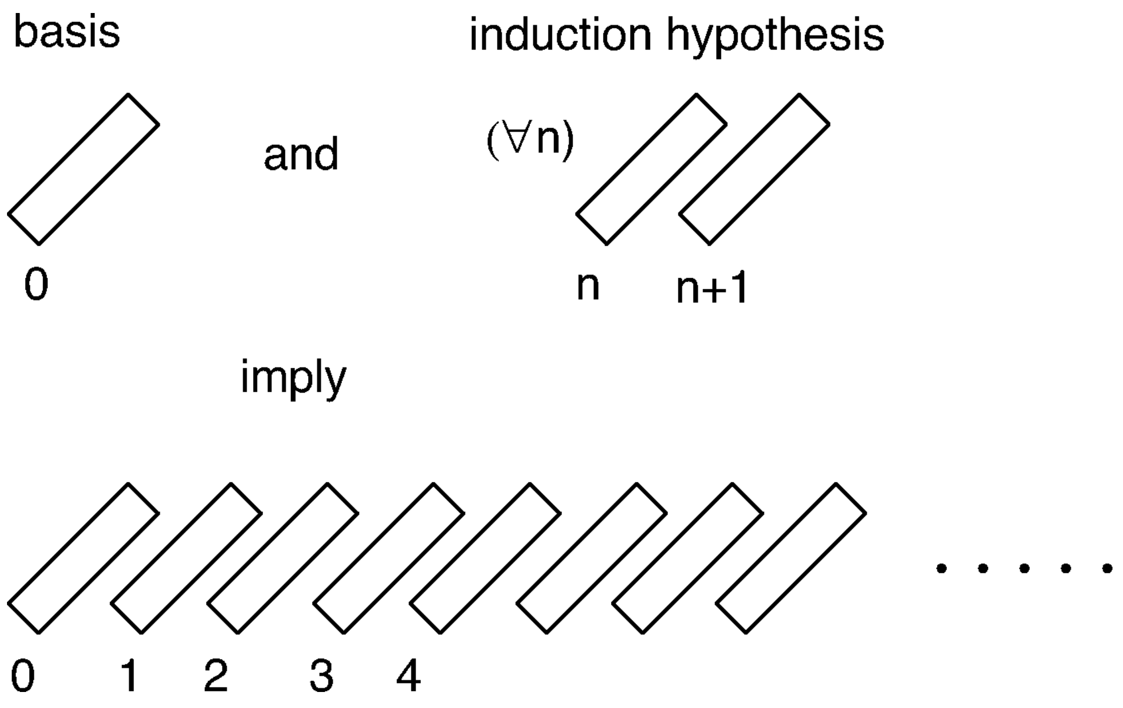
Therefore, the first 1000 dominoes all fall down.

The induction axiom for the integers may be characterized as

THE GREAT LEAP TO INFINITY

Given a countably infinite row of dominoes, suppose that:
(1) The initial domino falls.
(2) If domino n, then so domino $n + 1$.

Conclusion: All the dominoes all fall down.



Example 4.1.1: a proof by induction
Calculate the sum of the first k odd numbers:

$$1 + 3 + 5 + \cdots + (2k - 1)$$

Practical Method for General Problem Solving.
Special Case: Deriving a Formula
Step 1. Calculate the result for some small cases.
Step 2. Guess a formula to match all those cases.
Step 3. Verify your guess in the general case.

Step 1. examine small cases

$$\begin{aligned} \text{(empty sum)} &= 0 \\ 1 &= 1 \\ 1+3 &= 4 \\ 1+3+5 &= 9 \\ 1+3+5+7 &= 16 \end{aligned}$$

Step 2. It sure looks like $1 + 3 + ... + (2k - 1) = k^2$.

Step 3. Try to prove this assertion by induction.

$$(\forall k)\left[\sum_{j=1}^{k}(2j-1) = k^2\right]$$

(see next page for proof)

Basis Step. $\left[\sum_{j=1}^{k}(2j-1) \;=\; k^2\right]$ when $k=0$

Ind Hyp. $\left[\sum_{j=1}^{k}(2j-1) \;=\; k^2\right]$ when $k=n$

Ind Step. Consider the case $k=n+1$.

$$\begin{aligned}\sum_{j=1}^{n+1}(2j-1) &= \sum_{j=1}^{n}(2j-1) + [2(n+1)-1] \\ &= \sum_{j=1}^{n}(2j-1) + 2n+1 \\ &= n^2+2n+1 \text{ by ind. hyp.} \\ &= (n+1)^2 \text{ by factoring } \diamondsuit\end{aligned}$$

Why is induction important to CS majors?

It is the method used to prove that a loop or a recursively defined function correctly calculates the intended result. (just for a start)

Example 4.1.2: another proof by induction
Calculate the sum of the first k numbers:

$$1 + 2 + 3 + \cdots + k$$

Step 1. examine small cases

$$\begin{aligned}
\text{(empty sum)} &= 0 = \frac{0 \cdot 1}{2} \\
1 &= 1 = \frac{1 \cdot 2}{2} \\
1 + 2 &= 3 = \frac{2 \cdot 3}{2} \\
1 + 2 + 3 &= 6 = \frac{3 \cdot 4}{2} \\
1 + 2 + 3 + 4 &= 10 = \frac{4 \cdot 5}{2}
\end{aligned}$$

Step 2. Infer pattern: $\displaystyle\sum_{j=1}^{k} j = \frac{k(k+1)}{2}$.

Step 3. Use induction proof to verify pattern.

See next page.

Proposition 4.1.1. $\displaystyle\sum_{j=1}^{k} j = \frac{k(k+1)}{2}$.

Basis Step. $\displaystyle\sum_{j=1}^{k} j = \frac{k(k+1)}{2} = \frac{0 \cdot 1}{2}$ when $k = 0$.

Ind Hyp. $\displaystyle\sum_{j=1}^{k} j = \frac{k(k+1)}{2}$ when $k = n$.

Ind. Step.

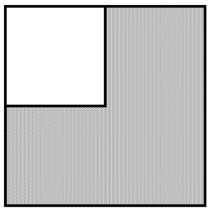
$$\begin{aligned}
\sum_{j=1}^{n+1} j &= \sum_{j=1}^{n} j + (n+1) \\
&= \frac{n(n+1)}{2} + (n+1) \quad \text{by ind hyp} \\
&= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} \quad \text{by arithmetic} \\
&= \frac{n(n+1) + 2(n+1)}{2} \quad \text{by arithmetic} \\
&= \frac{(n+2)(n+1)}{2} \quad \text{distrib in numerator} \\
&= \frac{(n+1)(n+2)}{2} \quad \text{commutativity} \qquad \diamondsuit
\end{aligned}$$

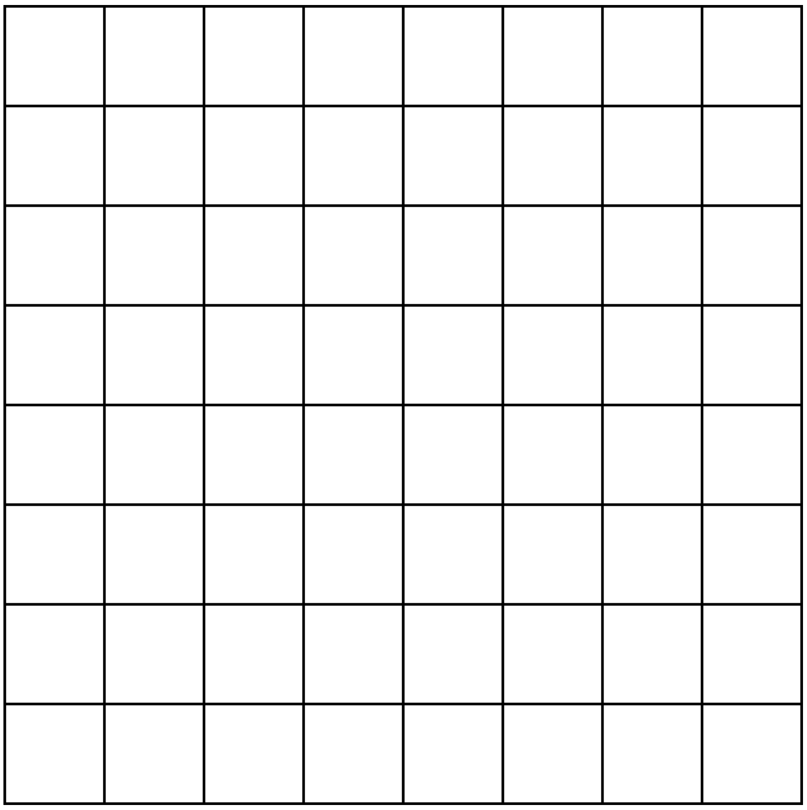
NONALGEBRAIC APPLICATIONS of INDUCTION

Consider tiling a 2^k-by-2^k chessboard.

(k = 3 in the figure below)

with L-shaped tiles, so that one corner-square is left uncovered.





Basis Step. You can do this when k = 0.

Ind Hyp. Assume you can do this for k = n.

Ind. Step. Prove you can do it for k = n+1.

4.2 ALTERNATIVE INDUCTION

In a proof by induction, verifying the inductive premise means you show that the antecedent of the quantified statement implies the conclusion.

DEF: In a proof by mathematical induction, the ***inductive hypothesis*** is the antecedent of the inductive premise.

Standard 0-based inductive rule of inference:

$$\begin{array}{cl} 0 \in S & \text{basis premise} \\ (\forall n)\,[n \geq 0 \wedge n \in S \Rightarrow n+1 \in S] & \text{ind prem} \\ \hline (\forall n)[n \geq 0 \Rightarrow n \in S] & \text{conclusion} \end{array}$$

Alternative Form 1. Using an integer other than zero as a basis.

$$\begin{array}{cl} b \in S & \text{basis premise} \\ (\forall n)\,[n \geq b \wedge n \in S \Rightarrow n+1 \in S] & \text{ind prem} \\ \hline (\forall n)[n \geq b \Rightarrow n \in S] & \text{conclusion} \end{array}$$

Example 4.2.1: using 5 as the basis

$$n^2 > 2n+1 \text{ for all } n \geq 5$$

Basis Step. $5^2 > 2 \cdot 5 + 1$

Ind Hyp. Assume $k^2 > 2k+1$ for $k \geq 5$.

Ind. Step.

$$\begin{aligned}
(k+1)^2 &= k^2 + 2k + 1 \quad \text{by arithmetic} \\
&> (2k+1) + 2k + 1 \quad \text{by ind hyp} \\
&= 4k + 2 \quad \text{by arithmetic} \\
&= 2(k+1) + 2k \quad \text{by arithmetic} \\
&\geq 2(k+1) + 10 \quad \text{since } k \geq 5 \\
&\geq 2(k+1) + 1 \quad \text{since } 10 \geq 1 \qquad \diamondsuit
\end{aligned}$$

Example 4.2.2: $2^n > n^2$ for all $n \geq 5$.

Basis Step. $2^5 > 5^2$

Ind Hyp. Assume $2^k > k^2$ for $k \geq 5$

Ind. Step.

$$\begin{aligned} 2^{k+1} &= 2 \cdot 2^k && \text{arithmetic} \\ &= 2^k + 2^k && \text{arithmetic} \\ &> k^2 + k^2 && \text{ind. hyp.} \\ &> k^2 + (2k+1) && \text{by Example 4.2.1} \\ &= (k+1)^2 && \text{arithmetic} \end{aligned}$$

$\diamondsuit$

Example 4.2.3: Prove that any postage of 8¢ or more can be created from nothing but 3¢ and 5¢ stamps.

Basis Step. $8 = 1 \cdot 3¢ + 1 \cdot 5¢$

Ind Hyp. Assume n¢ possible from 3's and 5's.

Ind. Step. Try to make $(n+1)$¢ postage.

Suppose that $n = r \cdot 3¢ + s \cdot 5¢$

Case 1: $s \geq 1$. Then $n + 1 = \ldots$

Case 2: $s = 0$. Then $n + 1 = \ldots$

Alternative Form 2. Inductive hypothesis is that the first n dominoes all fall down.

$$\begin{array}{cr} b \in S & \text{basis premise} \\ (\forall n : n \geq b)\,[(\forall k \leq n)\,[k \in S] \Rightarrow n+1 \in S] & \text{ind p} \\ \hline (\forall n : n \geq b)\,[n \in S] & \text{conclusion} \end{array}$$

Example 4.2.4: Prove that every integer $n > 0$ is the product of finitely many primes.

Basis Step. 1 is the empty product.

Ind Hyp. Assume that $1, \ldots, n$ are each a product of finitely many primes.

Ind Step.
(1) Either $n + 1$ is prime, or $\exists b, c \in \mathcal{Z}$ such that $n + 1 = bc$. (law of excl middle, def of prime)
(2) But b and c are the products of finitely many primes. (by Ind Hyp)
(3) Thus, so is bc. ♢

Mind-Benders re Induction

1. 2/3 ancestry
2. All solid billiard balls are the same color.
3. Everyone is essentially bald.

4.3 RECURSIVE DEFINITIONS

Functions can be defined recursively. The simplest form of recursive definition of a function f on the natural numbers specifies a basis rule

(B) the value $f(0)$

and a recursion rule

(R) how to obtain $f(n)$ from $f(n-1)$, $\forall n \geq 1$

Example 4.3.1: n-factorial $n!$

(B) $$0! = 1$$

(R) $$(n+1)! = (n+1) \cdot n!$$

However, recursive definitions often take somewhat more general forms.

Example 4.3.2: mergesort ($A[1 \ldots 2^n]$: real)

```
if n = 0
    return($A$)
otherwise
    return(merge (m'sort(1st half), m'sort(2nd half)))
```

Since a sequence is a special kind of function, some sequences can be specified recursively.

Example 4.3.3: Hanoi sequence $0, 1, 3, 7, 15, \ldots$

$$\begin{aligned} h_0 &= 0 \\ h_n &= 2h_{n-1} + 1 \text{ for } n \geq 1 \end{aligned}$$

Example 4.3.4: Fibo seq $1, 1, 2, 3, 5, 8, 13, \ldots$

$$\begin{aligned} f_0 &= 1 \\ f_1 &= 1 \\ f_n &= f_{n-1} + f_{n-2} \text{ for } n \geq 2 \end{aligned}$$

Example 4.3.5: partial sums of sequences

$$\sum_{j=0}^{n} a_j = \begin{cases} a_0 & \text{if } n = 0 \\ \sum_{j=0}^{n-1} a_j + a_n & \text{otherwise} \end{cases}$$

Example 4.3.6: Catalan seq $1, 1, 2, 5, 14, 42, \ldots$

$$\begin{aligned} c_0 &= 1 \\ c_n &= c_0 c_{n-1} + c_1 c_{n-2} + \cdots + c_{n-1} c_0 \text{ for } n \geq 1 \end{aligned}$$

RECURSIVE DEFINITION of SETS

DEF: A ***recursive definition of a set*** S comprises the following:

(B) a ***basis clause***,
which specifies a set of ***primitive elements***;

(R) a ***recursive clause***
which specifies how elements of the set may be constructed from elements already known to be in set S; there may be several recursive subclauses;

(E) an ***implicit exclusion clause***,
which provides that anything not in the set as a result of the basis clause or the recursive clause is not in set S.

Backus Normal Form (BNF) is an example of a context-free grammar that is used to give recursive definitions of sets. In W3261, you will learn that context-free languages are recognizable by push-down automata.

Example 4.3.7: a rec. def. set of integers

(B) $7, 10 \in S$

(R) if $r \in S$ then $r + 7, r + 10 \in S$

This reminds us of the postage stamp problem.

Claim $(\forall n \geq 54)\,[n \in S]$

Basis: $54 \;=\; 2 \cdot 7 + 4 \cdot 10$

Ind Hyp: Assume $n \;=\; r \cdot 7 + s \cdot 10$ with $n \geq 54$.

Ind Step: Two cases.

Case 1: $r \geq 7$.
Then $n + 1 \;=\; (r - 7) \cdot 7 + (s + 5) \cdot 10$.

Case 2: $r < 7 \;\Rightarrow\; r \cdot 7 \leq 42 \;\Rightarrow\; s \geq 2$.
Then $n + 1 \;=\; (r + 3) \cdot 7 \;+\; (s - 2) \cdot 10$.

In computer science, we often use recursive definitions of sets of strings.

RECURSIVE DEFINITION of STRINGS

NOTATION: The set of all strings in the alphabet Σ is generally denoted Σ^*.

Example 4.3.8: $\{0,1\}^*$ denotes the set of all binary strings.

DEF: ***string in an alphabet*** Σ

(B) (empty string) λ is a string;

(R) If s is a string and $b \in \Sigma$, then sb is a string.

Example 4.3.9: ***BNF*** for strings

$\langle\text{string}\rangle ::= \lambda \mid \langle\text{string}\rangle\langle\text{character}\rangle$

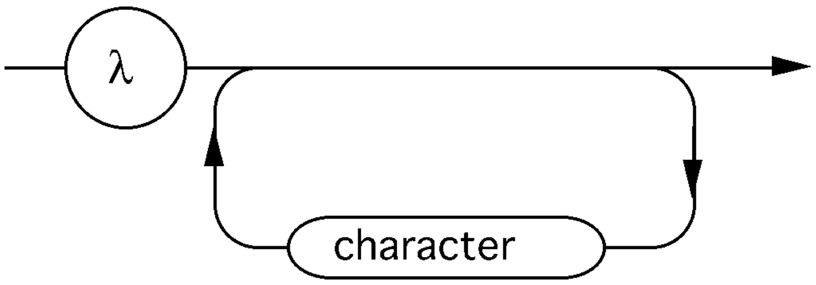


Fig 4.3.1 ***Railroad Normal Form*** for strings.

RECURSIVE DEFINITION of IDENTIFIERS

DEF: An ***identifier*** is either

(B) a letter, or

(R) an identifier followed by a digit or a letter.

(This definition of *identifier* is close to true for some early programming languages.)

Example 4.3.10: BNF for identifiers

$$\langle\text{lowercase\_letter}\rangle ::= a \mid b \mid \cdots \mid z$$
$$\langle\text{uppercase\_letter}\rangle ::= A \mid B \mid \cdots \mid Z$$
$$\langle\text{letter}\rangle ::= \langle\text{lowercase\_letter}\rangle \mid \langle\text{uppercase\_letter}\rangle$$
$$\langle\text{digit}\rangle ::= 0 \mid 1 \mid \cdots \mid 9$$
$$\langle\text{identifier}\rangle ::= \langle\text{letter}\rangle \mid \langle\text{identifier}\rangle\langle\text{letter}\rangle \mid \langle\text{identifier}\rangle\langle\text{digit}\rangle$$

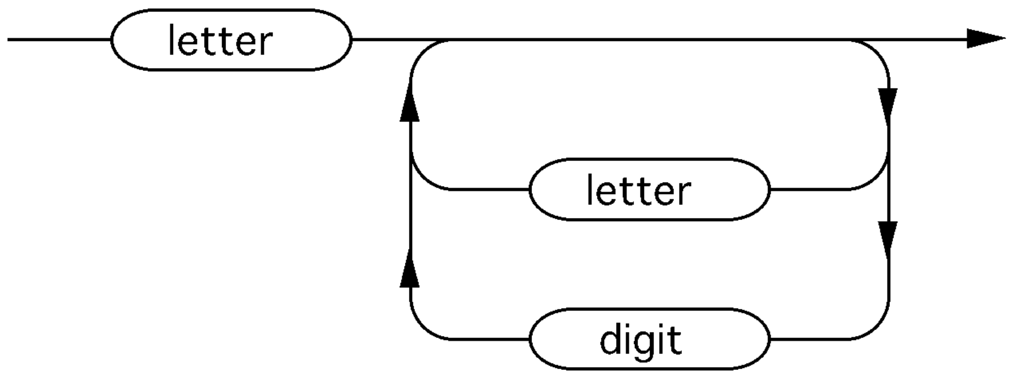


Fig 4.3.2 Railroad Form for identifiers.

ARITHMETIC EXPRESSIONS

DEF: ***arithmetic expressions***

(B) A numeral is an arithmetic expression.

(R) If e_1 and e_2 are arithmetic expressions, then so are all of the following:

$$e_1 + e_2 \quad e_1 - e_2 \quad e_1 * e_2$$
$$e_1/e_2 \quad e_1 * *e_2 \quad (e_1)$$

Example 4.3.11: Backus Normal Form

$$\begin{aligned}\langle\text{expression}\rangle \ ::= \ & \langle\text{numeral}\rangle \\ & \mid \langle\text{expression}\rangle + \langle\text{expression}\rangle \\ & \mid \langle\text{expression}\rangle - \langle\text{expression}\rangle \\ & \mid \langle\text{expression}\rangle * \langle\text{expression}\rangle \\ & \mid \langle\text{expression}\rangle/\langle\text{expression}\rangle \\ & \mid \langle\text{expression}\rangle * *\langle\text{expression}\rangle \\ & \mid (\langle\text{expression}\rangle)\end{aligned}$$

SUBCLASSES of STRINGS

Example 4.3.12: binary strings of even length

(B) $\lambda \in S$

(R) If $b \in S$, then $b00, b01, b10, b11 \in S$.

Example 4.3.13: binary strings of even length that start with 1

(B) $10, 11 \in S$

(R) If $b \in S$, then $b00, b01, b10, b11 \in S$.

DEF: A ***strict palindrome*** is a character string that is identical to its reverse.

NB. In *natural language palidromes*, punctuation and blanks are ignored, as is the distinction between upper and lower case letters.

Example 4.3.14: Able was I ere I saw Elba.

Example 4.3.15: A palindromic couplet.

Madam, I'm Adam.

Eve.

Example 4.3.16: set of binary palindromes

(B) $\lambda, 0, 1 \in S$

(R) If $x \in S$ then $0x0, 1x1 \in S$.

LOGICAL PROPOSITIONS

DEF: ***propositional forms***

(B) p, q, r, s, t, u, v, w are propositional forms

(R) If x and y are propositional forms, then so are

$$\begin{array}{ccc} \neg x & x \wedge y & x \vee y \\ x \to y & x \longrightarrow y & (x) \end{array}$$

Propositional forms under basis clause (B) are called ***atomic***.

Remark: Recursive definition of a set facilitates induction proofs of properties of its elements.

Proposition 4.3.1. *Every proposition has an even number of parentheses.*

Pf: by induction on the length of the derivation of a proposition.

Basis Step. Every atomic propositions has evenly many parentheses.

Ind Step. Assume that propositions x and y have evenly many parentheses. Then so do propositions

$$\begin{array}{ccc} \neg x & x \wedge y & x \vee y \\ x \to y & x \longrightarrow y & (x) \end{array}$$

$\diamondsuit$

CIRCULAR DEFINITIONS

DEF: A would-be recursive definition is ***circular*** if the sequence of iterated applications it generates fails to terminate in applications to elements of the basis set.

Example 4.3.17: a circular definition from Index and Glossary of Knuth, Vol 1.

Circular definition, 260
  see Definition, circular

Definition, circular,
  see Circular definition

4.4 RECURSIVE ALGORITHMS

REVIEW : An ***algorithm*** is a computational representation of a function.

Remark: Although it is often easier to write a correct recursive algorithm for a function, iterative implementations typically run faster, because they avoid calling the stack.

RECURSIVELY DEFINED ARITHMETIC

Example 4.4.1: recursive addition of natural numbers: *succ* = successor, *pred* = predecessor.

```
Algo 4.4.1: recursive addition

recursive function: sum$(m, n)$
Input: integers $m \geq 0, n \geq 0$
Output: $m + n$

If $n = 0$ then return $(m)$
  else return $(\text{sum}(succ(m), pred(n)))$
```

Example 4.4.2: iterative addition of natural numbers

Algo 4.4.2: iterative addition

```
function: sum($m, n$)
Input: integers $m \geq 0, n \geq 0$
Output: $m + n$
While $n > 0$ do
    $m := succ(m)$;
    $n := pred(n)$;
    endwhile
Return ($m$)
```

Example 4.4.3: proper subtraction of natural numbers: *succ* = successor, *pred* = predecessor.

```
Algo 4.4.3: proper subtraction

recursive function: diff$(m, n)$
Input: integers $0 \le n \le m$
Output: $m - n$
If $n = 0$ then return $(m)$
  else return $\big(\text{diff}(pred(m), pred(n))\big)$
```

Example 4.4.4: natural multiplication:

```
Algo 4.4.4: natural multiplication

recursive function: prod$(m, n)$
Input: integers $0 \le n \le m$
Output: $m \times n$
If $n = 0$ then return $(0)$
  else return $\big(\text{prod}(m, pred(n)) + m\big)$
```

Example 4.4.5: factorial function:

Algo 4.4.5: factorial

```
recursive function: factorial($n$)
Input: integer $n \geq 0$
Output: $n!$
If $n = 0$ then return $(1)$
  else return $\big(\text{prod}(n, \text{factorial}(n-1))\big)$
```

NOTATION: Hereafter, we mostly use

$$\text{infix notations}: + \quad - \quad * \quad !$$

to mean the functions

sum, diff, prod, and factorial

respectively.

RECURSIVELY DEFINED RELATIONS

DEF: The ***(Iverson) truth function*** **true** assigns to an assertion the boolean value TRUE if true and FALSE otherwise.

Example 4.4.6: order relation:

```
Algo 4.4.6: order relation

recursive function: ge($m, n$)
Input: integers $m, n \geq 0$
Output: true $(m \geq n)$
If $n = 0$ then return (TRUE)
  elseif $m = 0$ then return (FALSE)
  else return $\big(\text{ge}(m-1, n-1)\big)$
```

Time-Complexity: $\Theta(\min(m, n))$.

OTHER RECURSIVELY DEFINED FUNCTIONS

REVIEW EUCLIDEAN ALGORITHM:

Algo 4.4.7: Euclidean algorithm

```
recursive function: gcd($m, n$)
Input: integers $m > 0, n \geq 0$
Output: gcd $(m, n)$
If $n = 0$ then return $(m)$
  else return $\big(\gcd(n, m \bmod n)\big)$
```

Time-Complexity: $\mathcal{O}(\ln n)$.

Example 4.4.7: Iterative calc of gcd (289, 255)

$$\begin{array}{ccc} m_1 = 289 & n_1 = 255 & r_1 = 34 \\ m_2 = 255 & n_2 = 34 & r_2 = 17 \\ m_3 = 34 & n_3 = 17 & r_3 = 0 \end{array}$$

DEF: The execution of a function exhibits ***exponential recursive descent*** if a call at one level can generate multiple calls at the next level.

Example 4.4.8: Fibonacci function

$$f_0 = 1 \qquad f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2} \text{ for } n \geq 2$$
$$\Rightarrow 1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13$$
$$21 \quad 34 \quad 55 \quad 89 \quad 144 \quad \ldots$$

Algo 4.4.8: Fibonacci function

```
iterative speedup function: fibo($n$)
Input: integer $n \geq 0$
Output: fibo ($n$)
If $n = 0 \vee n = 1$ then return (1)
else $f_{n-2} := 1; f_{n-1} := 1;$
  for $j := 2$ to $n$ step 1
    $f_n := f_{n-1} + f_{n-2};$
    $f_{n-2} := f_{n-1};$
    $f_{n-1} := f_n;$ endfor
return ($f_n$)
```

Time-Complexity: $\Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$.

RECURSIVE STRING OPERATIONS

$$\begin{aligned}
\Sigma &= \text{set}; \quad c \in \Sigma(\text{object}); \quad s \in \Sigma^*(\text{string}) \\
\Sigma^0 &= \Lambda = \{\lambda\} \quad \text{strings of length } 0 \\
\Sigma^n &= \Sigma^{n-1} \times \Sigma \quad \text{strings of length } n \\
\Sigma^+ &= \Sigma^1 \cup \Sigma^2 \cup \cdots \text{ all finite non-empty strings} \\
\Sigma^* &= \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots \text{ all finite strings}
\end{aligned}$$

These three ***primitive string functions*** are all defined and implemented nonrecursively for arbitrary sequences, not just strings of characters.

DEF: ***appending a character*** to a string

append : $\Sigma^* \times \Sigma \to \Sigma^*$ non-recursive

$$(a_1 a_2 \cdots a_n, c) \mapsto a_1 a_2 \cdots a_n c$$

DEF: ***first character*** of a non-empty string

first : $\Sigma^+ \to \Sigma$ non-recursive

$$a_1 a_2 \cdots a_n \mapsto a_1$$

DEF: ***trailer*** of a non-empty string

trailer : $\Sigma^n \to \Sigma^{n-1}$

$$a_1 a_2 \cdots a_n \mapsto a_2 \cdots a_n$$

These four ***secondary string functions*** are all defined and implemented recursively.

DEF: ***length*** of a string $\Sigma^* \to \mathbb{N}$

$$\text{length}(s) \;=\; \begin{cases} 0 & \text{if } s = \lambda \\ 1 + \text{length}(\text{trailer}(s)) & \text{if } s \neq \lambda \end{cases}$$

DEF: ***concatenate*** two strings $\Sigma^* \times \Sigma^* \to \Sigma^*$

$$(s \circ t) \;=\; \begin{cases} s & \text{if } t = \lambda \\ \text{append}(s, \text{first}(t)) \circ \text{trailer}(t) & \text{if } t \neq \lambda \end{cases}$$

NOTATION: It is customary to **overload** the concatenation operator $\circ$ so that it also appends.

DEF: ***reversing*** a string $\Sigma^* \to \Sigma^*$

$$s^{-1} \;=\; \begin{cases} s & \text{if } s = \lambda \\ \text{trailer}(s)^{-1} \circ \text{first}(s) & \text{if } s \neq \lambda \end{cases}$$

DEF: ***last character*** of string $\neq \lambda$ $\Sigma^+ \to \Sigma$

$$\text{last}(s) \;=\; \text{first}(s^{-1})$$

RECURSIVE ARRAY OPERATIONS

Algo 4.4.9: location

```
**recursive function:** location$(x, A[\,])$
*Input:* target value $x$, sorted array $A[\,]$
*Output:* 0 if $x \notin A$; $\min\{j \mid x = A[j]\}$ if $x \in A$

**If length** $(A) = 1$ **then**
    **return** (**true** $(x = A[1])$)
**elseif** $x \leq$ midval$(A)$
    **return** location$(x, \text{fronthalf}(A))$
**else**
    **return** location$(x, \text{backhalf}(A))$

**function:** midindex$(A)$
*Input:* array $A[\,]$
*Output:* middle location of array $A$
midindex$(A) = \lfloor \textbf{length}(A)/2 \rfloor$

**function:** midval$(A)$
*Input:* array $A[\,]$
*Output:* value at middle location of array $A$
midval$(A) = A[\text{midindex}(A)]$
```

continued on next page

Algo 4.4.9: location, continuation

```
**function:** fronthalf$(A)$
*Input:* array $A[\,]$
*Output:* front half-array of array $A$
fronthalf$(A) = A[1 \ldots \text{midindex}(A)]$

**function:** backhalf$(A)$
*Input:* array $A[\,]$
*Output:* back half-array of array $A$
fronthalf$(A) = A[\text{midindex}(A) + 1 \ldots \text{length}(A)]$
```

Time-Complexity: $\Theta(\log n)$.

Algo 4.4.10: verify ascending order

```
**recursive function:** ascending$(A[\,])$
*Input:* array $A[\,]$
*Output:* TRUE if ascending; FALSE if not
**if length** $(A[\,]) \leq 1$ **then**
    **return** $(TRUE)$
**else**
    **return** $(a_1 \leq a_2 \wedge \text{ascending}(\text{trailer}(A[\,])))$
```

Time-Complexity: $\Theta(n)$.

Algo 4.4.11: merge sequences

```
recursive function: merge($s, t$)
Input: ascending sequences $s, t$
Output: merged ascending sequence
If length $(s) = 0$ then
    return $t$
elseif $s_1 \le t_1$
    return $\big(\text{first}(s) \circ \text{merge}(\text{trailer}(s), t)\big)$
else
    return $\big(\text{first}(t) \circ \text{merge}(s, \text{trailer}(t))\big)$
```

Algo 4.4.12: mergesort

```
recursive function: msort($A$)
Input: array $A[\,]$
Output: ascending array
If length $(A[\,]) \le 1$ then
    return $(A[\,])$
else
    return
    $(\text{merge}\,(\text{msort}\,(\text{fronthalf}(A), \text{msort}\,(\text{backhalf}(A)))$
```