ICS141: Discrete Mathematics for Computer Science I

Dept. Information & Computer Sci., University of Hawaii

Jan Stelovsky

based on slides by Dr. Baek and Dr. Still

Originals by Dr. M. P. Frank and Dr. J.L. Gross

Provided by McGraw-Hill

Lecture 23

Chapter 4. Induction and Recursion

4.5 Program Correctness

Chapter 5. Counting

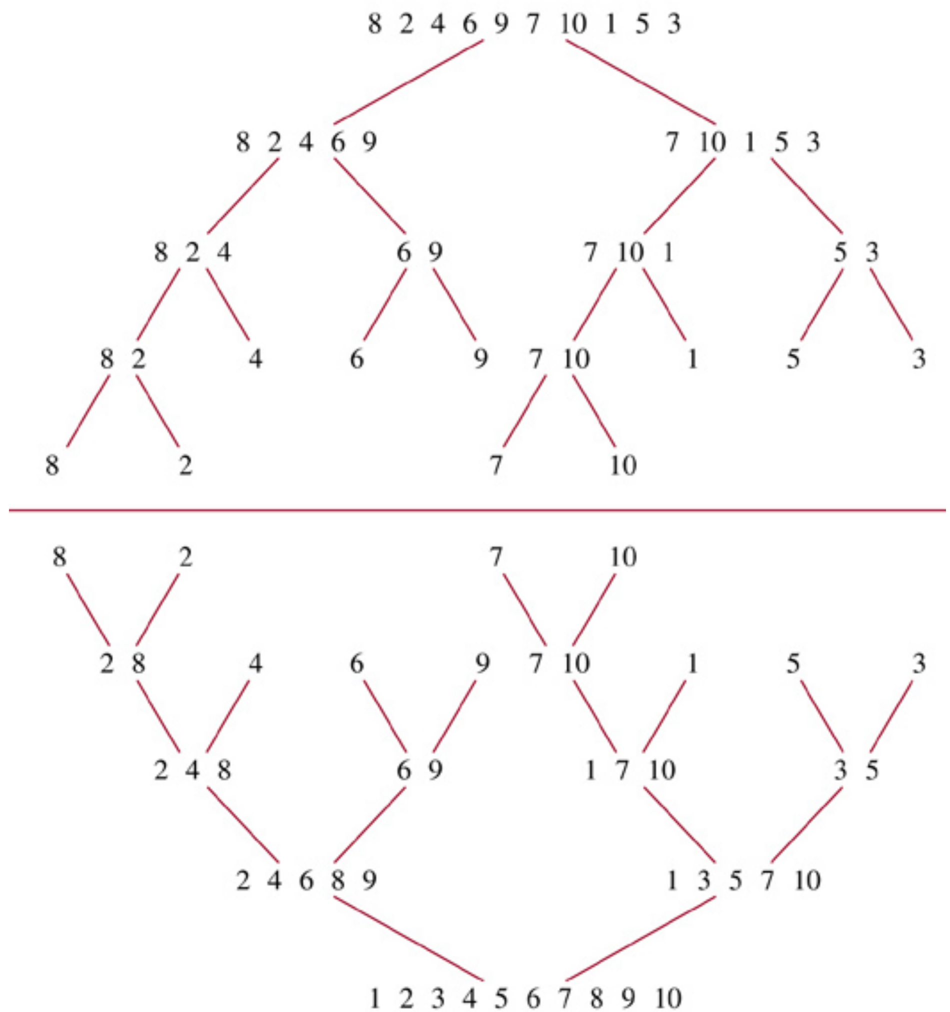
5.1 The Basics of Counting

Quiz

- Develop a recursive procedure for computing the minimum item in a list of integer numbers.

- Give is the recursive definition:
    - f(0) = f(1) = 2
    - f(n+1) = f(n) * f(n-1)
  - Develop a recursive procedure for this definition
  - What is your most time-efficient way to compute f(n)?
  - What are the complexities of the recursive method and of yours?

Recursive Merge Sort Example



Split

Merge

Recursive Merge Sort

University of Hawaii

procedure *mergesort*($L = \ell_1, \ldots, \ell_n$)

if $n > 1$ **then**

$m := \lfloor n/2 \rfloor$ {this is rough ½-way point}

$L_1 := \ell_1, \ldots, \ell_m$

$L_2 := \ell_{m+1}, \ldots, \ell_n$

$L := merge(mergesort(L_1), mergesort(L_2))$

return L

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

Merging Two Sorted Lists

TABLE 1 Merging the Two Sorted Lists 2, 3, 5, 6 and 1, 4.

First List	*Second List*	*Merged List*	*Comparison*
2 3 5 6	1 4		1 < 2
2 3 5 6	4	1	2 < 4
3 5 6	4	1 2	3 < 4
5 6	4	1 2 3	4 < 5
5 6		1 2 3 4	
		1 2 3 4 5 6	

Recursive Merge Method

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

{Given two sorted lists $A = (a_1,\ldots, a_{|A|})$,
$B = (b_1,\ldots, b_{|B|})$, return a sorted list of all.}

if A = empty **return** B {If A is empty, it's B.}

if B = empty **return** A {If B is empty, it's A.}

if $a_1 < b_1$ **then**

$L := (a_1, merge((a_2,\ldots, a_{|A|}), B))$

else

$L := (b_1, merge(A, (b_2,\ldots, b_{|B|})))$

return L

Merge Routine

University of Hawaii

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

L = empty list

i:=0, j:=0, k:=0

while $i < |A| \wedge j < |B|$ {$|A|$ 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

Takes $\Theta(|A|+|B|)$ time

Program Correctness

University of Hawaii

- We want to be able to ***prove*** that a given program meets the intended specifications.
  - This can often be done manually, or even by *automated program verification* tools.
- A program is ***correct*** if it produces the correct output for every possible input.
- A program is ***partially correct*** if it produces the correct output for every input for which the program eventually halts.

Initial & Final Assertions

- A program's I/O specification can be given using ***initial*** and ***final*** assertions.
  - **The *initial assertion p*** is the condition that the program's input (its initial state) is guaranteed to satisfy (by its user).
  - **The *final assertion q*** is the condition that the output produced by the program (in its final state) is required to satisfy.
- ***Hoare triple*** notation:
  - The notation ***p*{*S*}*q*** means that, for all inputs I such that $p(I)$ is true, if program S (given input I) halts and produces output $O = S(I)$, then $q(O)$ is true.
    - That is, S is partially correct with respect to specification p, q.

A Trivial Example

- Let S be the program fragment "$y := 2; z := x + y$"
- Let p be the initial assertion "$x = 1$".
  - The variable x will hold 1 in all initial states.
- Let q be the final assertion "$z = 3$".
  - The variable z must hold 3 in all final states.
- Prove $p\{S\}q$.
  - **Proof:** If $x = 1$ in the program's input state, then after running $y := 2$ and $z := x + y$, z will be $1 + 2 = 3$.

Hoare Triple Inference Rules

- Deduction rules for Hoare Triple statements.
- A simple example: the ***composition rule:***

$$\frac{\begin{array}{c} p\{S_1\}q \\ q\{S_2\}r \end{array}}{\therefore\ p\{S_1;\ S_2\}r}$$

- **It says:** If program S_1 given condition p produces condition q, and S_2 given q produces r, then the program "S_1 followed by S_2", if given p, yields r.

Inference Rule for *if* Statements

University of Hawaii

- Program segment that is the conditional statement

  if *condition* **then**

  $\quad S$

- Rule of inference

$$\frac{\begin{array}{l}(p \wedge \mathit{condition})\{S\}q \\ (p \wedge \neg \mathit{condition}) \rightarrow q\end{array}}{\therefore\ p\{\textbf{if } \mathit{condition} \textbf{ then } S\}q}$$

Initial assertion

Final assertion

- Example: Show that **T** {**if** $x > y$ **then** $y := x$} $y \geq x$.
  - **Proof:** When the initial assertion is true and if $x > y$, then the **if** body is executed, which sets $y = x$, and so afterwards $y \geq x$ is true.
    Otherwise, $x \leq y$ and so $y \geq x$. In either case $y \geq x$ is true.
    So the fragment meets the specification.

if-then-else Rule

- Program segment that is the conditional statement

  if *condition* **then**

  S_1

  else

  S_2

- Rule of inference

$$\frac{\begin{array}{l}(p \wedge \mathit{condition})\{S_1\}q \\ (p \wedge \neg\mathit{condition})\{S_2\}q\end{array}}{\therefore\ p\{\textbf{if }\mathit{condition}\textbf{ then }S_1\textbf{ else }S_2\}q}$$

- Example: Show that

  T {**if** $x < 0$ **then** $abs := -x$ **else** $abs := x$} $abs = |x|$

  - If $x < 0$ then after the **if** body, $abs = -x = |x|$.
    If $\neg(x < 0)$, *i.e.*, $x \geq 0$, then after the **else** body, $abs = x = |x|$.
    So the rule applies and the program segment is correct.

Loop Invariants

- For a while loop "**while** *condition* S", we say that p is a ***loop invariant*** of this loop if $(p \wedge condition)\{S\}p$.
  - If p (and the continuation condition *condition*) is true before executing the body, then p remains true afterwards.
    - And so p stays true through *all* subsequent iterations.
- This leads to the inference rule:

p is a loop invariant

$$\frac{(p \wedge condition)\{S\}p}{\therefore\ p\{\textbf{while}\ condition\ S\}(\neg condition \wedge p)}$$

Loop Invariant Example

University of Hawaii

S

> $i := 1$
> $fact := 1$
> **while** $i < n$
> $\quad i := i + 1;$
> $\quad fact := fact \cdot i$
> **end while**

- Prove that the following Hoare triple holds when n is a positive integer: **T** $\{S\}$ $(fact = n!)$

Read textbook for the detailed proof including the proof for loop invariant.

- **Proof.** Note that p: "$fact = i! \land i \le n$" is a loop invariant, and is true before the loop. Thus, after the loop we have $(\neg condition \land p) \Leftrightarrow \neg(i < n) \land (fact = i! \land i \le n) \Leftrightarrow i = n \land fact = i! \Leftrightarrow fact = n!$. ■

Big Example

- S **=** S_1**;** S_2**;** S_3**;** S_4 (compute the product of two integers m, n)

procedure *multiply*(m, n: integers) — p: $m, n \in \mathbf{Z}$

S_1 **if** $n < 0$ **then** $a := -n$ **else** $a := n$ — q: $p \wedge (a = |n|)$

S_2 $k := 0$; $x := 0$ — r: $q \wedge (k = 0) \wedge (x = 0)$

Loop invariant $x = mk \wedge k \le a$

S_3 **while** $k < a$ { — Maintains loop invariant:

$x = x + m$; $k = k + 1$ — $x = mk \wedge k \le a$

} — $x = mk \wedge k = a \;\therefore x = ma = m|n|$ s

$\therefore (n < 0 \wedge x = -mn) \vee (n \ge 0 \wedge x = mn)$

S_4 **if** $n < 0$ **then** $prod := -x$ **else** $prod := x$ — $prod = mn$ t

Chapter 5: Counting

- Combinatorics
  - The study of the number of ways to put things together into various combinations.
  - *E.g.* In a contest entered by 100 people,
    - how many different top-10 outcomes could occur?
  - *E.g.* If a password is 6~8 letters and/or digits,
    - how many passwords can there be?

Sum and Product Rules

- Let m be the number of ways to do task 1 and n the number of ways to do task 2,
  - with each number independent of how the other task is done,
  - and also assume that no way to do task 1 simultaneously also accomplishes task 2.
- Then, we have the following rules:
  - The ***sum rule***: The task "do either task 1 or task 2, but not both" can be done in $m + n$ ways.
  - The ***product rule***: The task "do both task 1 and task 2" can be done in mn ways.

The Sum Rule

- If a task can be done in one of n_1 ways, or in one of n_2 ways, …, or in one of n_m ways, where none of the set of n_i ways of doing the task is the same as any of the set of n_j ways, for all pairs i and j with $1 \leq i < j \leq m$.
- Then the number of ways to do the task is

$$n_1 + n_2 + \cdots + n_m.$$

The Sum Rule: Example 1

- A student can choose a computer project from one of three lists A, B, and C:
  - List A: 23 possible projects
  - List B: 15 possible projects
  - List C: 19 possible projects
  - No project is on more than one list
- How many possible projects are there to choose from?

$$23 + 15 + 19 = 57$$

The Sum Rule: Example 2

- What is the value of k after the following code has been executed?

$k := 0$

for $i_1 := 1$ **to** n_1

$k := k + 1$

for $i_2 := 1$ **to** n_2

$k := k + 1$

…

for $i_m := 1$ **to** n_m

$k := k + 1$

$$n_1 + n_2 + \cdots + n_m$$

The Product Rule

University of Hawaii

- Suppose that a procedure can be broken down into a sequence of m successive tasks.
  If the task T_1 can be done in n_1 ways;
  the task T_2 can then be done in n_2 ways; ...;
  and the task T_m can be done in n_m ways, then
  there are $n_1 \cdot n_2 \cdots n_m$ ways to do the procedure.

The Product Rule: Example

University of Hawaii

- Show that a set $\{x_1,\ldots, x_n\}$ containing n elements has 2^n subsets.
  - A subset can be constructed in n successive steps:
    - Pick or do not pick x_1, pick or do not pick x_2, …, pick or do not pick x_n .
  - Each step can be done in two ways.
  - Thus the number of possible subsets is $\underbrace{2\cdot2\cdots2}_{n \text{ factors}} = 2^n.$