Induction and recursion

Chapter 5

With Question/Answer Animations

Copyright © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.

Chapter Summary

- Mathematical Induction
- Strong Induction
- Well-Ordering
- Recursive Definitions
- Structural Induction
- Recursive Algorithms
- **Program Correctness**

Section 5.5:
Program Correctness

Program Correctness (4.5)

- Introduction

  Question: How can we be sure that a program always produces the correct answer?

  - The syntax is correct (all bugs removed!)
  - Testing a program with a randomly selected sample of input data is not sufficient
  - Correctness of a program should be proven!
  - Theoretically, it is never possible to mechanize the proof of correctness of complex programs
  - We will cover some of the concepts and methods that prove that "simple" programs are correct

Program verification

- To prove program correct, we need two parts:
  1. For every possible input, the correct answer is obtained if the program terminates
  2. The program always terminates

Definition 1:

A program, or program segment, S is said to be partially correct with respect to the initial assertion p and the final assertion q if

whenever p is true for the input values of S and S terminates, then q is true for the output values of S.

The notation p{S}q indicates that the program, or program segment, S is partially correct with respect to the initial assertion p and the final assertion q.

Some notes and Example

This definition of partial correctness

- has nothing to do with whether a program terminates or not
- is due to Tony Hoare

Example: Show that the program segment

```
y := 2
z := x + y
```

is correct with respect to

- initial assertion p: x = 1;
- final assertion q: z = 3.

Solution: p is true $\Rightarrow$ x = 1 $\Rightarrow$ y := 2 $\Rightarrow$ z := 3 $\Rightarrow$ partially correct w.r.t. p and q

Rules of inference

Goal: Split the program into a series of subprograms and show that each subprogram is correct. This is done through a rule of inference.

- Let us start by taking the program S and splitting it into 2 subprograms S_1 and S_2 ($S = S_1; S_2$)
- Assume that we have S_1 correct w.r.t. p and q (initial and final assertions)
- Assume that we have S_2 correct w.r.t. q and r (initial and final assertions)

Rules of inference (cont.)

- It follows that

  "if p is true $\wedge$ (S_1 executed and terminates) then q is true"

  "if q is true $\wedge$ (S_2 executed and terminates) then r is true"

  "thus, if p = true and S = S_1; S_2 is executed and terminates then r = true"

  This rule of inference is known as the composition rule.

- It is written as:

$$\frac{p\ \{S_1\}\ q \qquad q\ \{S_2\}\ r}{\therefore p\ \{S1; S2)\ r}$$

Conditional Statements

Assume that a program segment has the following form:

"if *condition* then S" where S is a block of statement

Goal: Verify that this piece of code is correct

Strategy:

a) We must show that when p is true and *condition* is also true, then q is true after S terminates

b) We also must show that when p is true and *condition* is false, then q is true

Conditional Statements (cont.)

- We summarize as:

$$\frac{(p \wedge \textit{condition})\ \{S\}\ q \qquad (p \wedge \neg \textit{condition}) \Rightarrow q}{\therefore p\ \{\textbf{if}\ \textit{condition}\ \textbf{then}\ S\}\ q}$$

Example: Verify that the following program segment is correct w.r.t. the initial assertion T and the final assertion $y \geq x$

```
if x > y then y:= x
```

Solution:

a) If T = true and x > y is true then the final assertion $y \geq x$ is true

b) If T = true and x > y is false then $x \leq y$ is true $\Rightarrow$ final assertion is true again

If then else

"if *condition* then S_1 else S_2"

if *condition* is true then S_1 executes; otherwise S_2 executes

This piece of code is correct if:

a) If p = true $\wedge$ *condition* = true $\Rightarrow$ q = true after S_1 terminates

b) If p = true $\wedge$ *condition* = false $\Rightarrow$ q = true after S_2 terminates

$$\frac{(p \wedge condition)\ \{S_1\}q \qquad (p \wedge \neg condition)\ \{S_2\}q}{\therefore p\ \{\textbf{if}\ condition\ \textbf{then}\ S_1\ \textbf{else}\ S_2)\ q}$$

Example

Check that the following program segment

```
if x < 0 then
        abs := -x
else
        abs := x
```

is correct w.r.t. the initial assertion T and the final assertion abs = |x|.

Solution:

a) If T = true and (x<o) = true ⇒ abs := -x; compatible with definition of abs

b) If T = true and (x<o)= false ⇒ (x ≥ o) = true ⇒ abs := x; also compatible with abs definition.

Loop invariants

How to prove codes that contain the while loop:

"while *condition* S"

An assertion that remains true each time S is a loop invariant, i.e.,

p is a loop invariant if:

$$(p \wedge \textit{condition})\ \{S\}\ p \text{ is true}$$

If p is a loop invariant, then if p is true before the program segment is executed, p and $\neg$*condition* are true after termination, if it occurs. We can write the rule of inference as:

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

Loop example: factorial

Determine the loop invariant that verifies that the following program segment terminates with factorial = n! when n ≥ 0.

```
i := 1
factorial := 1
While i < n
        begin
                i := i + 1
                factorial := factorial * i
        end
```

Loop example: factorial (cont.)

Solution:

Choose p = (factorial = i! $\wedge$ (i $\leq$ n))

a) Prove that p is in fact a loop invariant

b) If p is true before execution, p and $\neg$condition are true after termination

c) Prove that the **while** loop terminates

Loop example: factorial (cont.)

a) P is a loop invariant:

Suppose p is true at the beginning of the execution of the **while** loop and the **while** *condition* holds;

$\Leftrightarrow$ factorial = i! $\wedge$ i < n

$i_{new} = i + 1$

$factorial_{new} = factorial * (i + 1) = (i + 1)! = i_{new}!$

Since i < n $\Rightarrow$ $i_{new} = i + 1 \leq n$

$\Rightarrow$ p true at the end of the execution of the loop

$\Rightarrow$ p is a loop invariant

Loop example: factorial (cont.)

b) Before entering the loop, $i = 1 \leq n$ and
factorial :=1 = 1! = i! $\Rightarrow (i \leq n) \wedge (\text{factorial} = i!) = \text{true}$
$\Rightarrow p = \text{true}$
Since p is a loop invariant
$\Rightarrow$ through the inference rule,
if the **while** loop terminates
$\Rightarrow p = \text{true}$ and $i < n$ false $\Rightarrow i = n$ and factorial = i! = n!

c) **While** loop terminates:

Because p is a loop invariant, the rule of inference implies that if the **while** loop terminates, it terminates with p true and with $i < n$ false.

In this case, at the end, factorial = i! and i<=n are true,
but i<n is false, in other words,
i=n and factorial= i!=n!, as desired.