MECHTRON 2MD3

Data Structures and Algorithms for Mechatronics

Winter 2022

29 Program Correctness

Department of Computing and Software

Instructor:

Omid Isfahanialamdari

March 31, 2022

Admin

- We will continue our discussion on Heaps and other algorithms after this quick introduction to the program correctness.
  - Next week's tutorial is about this concept

Program Correctness

- A brief introduction to the area of **program verification** which uses:
  - the rules of logic
  - proof techniques
  - the concept of an algorithm

- **Program verification** means to prove the correctness of the program.
- Why is this important? Why can't we merely run testcases?
  - Tests are useful, but they can't prove that your code will work in all possible scenarios.
  - Program testing can be used to show the presence of bugs, but never to show their absence! [E. Dijkstra]
- A program is said to be correct if it produces the correct output for every possible input.

How to Prove

- Instead of running and testing a program, take the help of logic
  - Transform the program specifications into formal specifications consisting of what is known to be true before the intended program runs and what should be true after its successful run
  - Write the code or an algorithm or pseudocode and reason about each statement in the code
  - Argue on what is guaranteed to be true after each statement and what is guaranteed after the last statement is executed.
  - Alternatively, start with the desired result and argue about what should be true before each previous statement

Proposition and Assertion

- **Proposition** is a statement that is either true or false. For example:
  - The grass is green
  - It is raining
  - Ottawa is the capital of Canada
- **Assertion** is a statement that one claims to be true.
  - Depending on the context items 1, 2, 3 above can be assertions.
- Propositional Logic
  - Atomic propositions (**p**, **q**, **r**) plus connectives ($\wedge$, $\vee$, $\neg$, $\Rightarrow$)
  - Connectives create complex propositions.
  - Truth tables used to describe the operations.

Conditionals

- If **p** and **q** are arbitrary propositions, then the conditional of **p** and **q** is written as:

$$p \Rightarrow q$$

(read as: if p then q)

p	q	$p \implies q$
F	F	T
F	T	T
T	F	F
T	T	T

- **p** ⇒ **q** will be true **iff** either **p** is false or **q** is true.

Pre- and Post-Conditions

- A simple formal specification consists of assertions about the state of a program
  - Precondition: Specifies the state before the program executes (initial assertion)
  - Postcondition: Specifies the state after execution of the program (final assertion)

Correctness Proof

- A correctness proof for a program consists of two parts:
  - Establish the **partial correctness** of the program.
    - *If the program terminates, then it halts with the correct answer.*
  - Show that the program **always terminates**.

Proving Output Correct

- We need two propositions to determine what is meant by produce the correct output.
  - Initial Assertion: the properties the input values must have. (p)
  - Final Assertion: the properties the output of the program should have if the program did what was intended. (q)

- A program segment **S** is said to be partially correct with respect to **p** and **q**, (we write it like this: **p** {**S**} **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.
  - **p** {**S**} **q** is called Hoare Triple

$$\begin{array}{l} p\{S_1\}q \\ q\{S_2\}r \\ \hline \therefore\ p\{S_1;\ S_2\}r. \end{array}$$

Example

```
p :   x = 1          // initial assertion

      y = 2          // segment
      z = x + y      //    S

q :   z = 3          // final assertion
```

- Is [p {S} q] true?
  - Suppose that p is true, so that x = 1 as the program begins. Then y is assigned the value 2, and z is assigned the sum of the values of x and y, which is 3. Hence, S is correct with respect to the initial assertion p and the final assertion q. Thus, p{S}q is true.

Composition Rule

- if p is true and S = S1; S2 is executed and terminates, then r is true. This rule of inference, called the **composition rule**

$$\begin{array}{l} p\{S_1\}q \\ q\{S_2\}r \\ \hline \therefore\ p\{S_1; S_2\}r. \end{array}$$

Rules of Inference: Conditional Statements

- IF condition THEN block

$$\textbf{if } \textit{condition} \textbf{ then} \\ S$$

- S is executed when **condition** is true, and it is not executed when condition is false. To verify correctness with respect to p and q, we must show:
  - When p is true and condition is also true, then q is true after S terminates.
  - When p is true and condition is false, q is true (since S does not execute.

This leads to the following rule of inference:

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

Rules of Inference: Conditional Statements

Verify if this program

$$\begin{aligned}&\textbf{if } x > y \textbf{ then}\\&\quad y := x\end{aligned}$$

is correct with respect to the initial assertion **T** and the final assertion y ≥ x.

Solution: When the initial assertion is true and x > y, the assignment y := x is carried out. Hence, the final assertion, which asserts that y ≥ x, is true in this case. Moreover, when the initial assertion is true and x > y is false, so that x ≤ y, the final assertion is again true. Hence, using the rule of inference for program segments of this type, this program is correct with respect to the given initial and final assertions.

Rules of Inference: Conditional Statements

if *condition* **then**
    S_1
else
    S_2

- If condition is true, then S1 executes; if condition is false, then S2 executes. To verify correctness with respect to p and q, we must show:
  - When p is true and condition is also true, then q is true after S1 terminates.
  - When p is true and condition is false, q is true after S2 terminates. This leads to the following rule of inference:

Rules of Inference: Conditional Statements

Verify that:

if $x < 0$ **then**
    $abs := -x$
else
    $abs := x$

is correct with respect to the initial assertion **T** and the final assertion

abs = |x|.

Solution:

Two things must be demonstrated. First, it must be shown that if the initial assertion is true and $x < 0$, then *abs* = |x|. This is correct, because when $x < 0$ the assignment statement *abs* := −x sets *abs* = −x, which is |x| by definition when $x < 0$. Second, it must be shown that if the initial assertion is true and $x < 0$ is false, so that $x \geq 0$, then *abs* = |x|. This is also correct, because in this case the program uses the assignment statement *abs* := x, and x is |x| by definition when $x \geq 0$, so *abs* := x. Hence, using the rule of inference for program segments of this type, this segment is correct with respect to the given initial and final assertions.

Loop Invariant

$$\textbf{while } \textit{condition} \\ S$$

- Where S is repeatedly executed until condition becomes false.
- Loop Invariant: an assertion that remains true each time block is executed. (a property that remains true during every traversal of a loop)
  - I.e., p is a loop invariant if (p ∧ condition){block}p is true
  - Let p be a loop invariant.
  - If p is true before Segment S is executed, then p and ¬condition are true after the loop terminates (if it does).

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

Loop Invariant - Example

- We wish to verify the following code segment terminates with factorial = n! when n is a positive integer.

  - Our loop invariant p is: **factorial = i! and i ≤ n**

```
i = 1
factorial = 1
while i < n  {
   i = i + 1
   factorial = factorial * i
}
```

Loop Invariant - Example

- **[Base Case]** p is true before we enter the loop since factorial = 1 = 1!, and $1 \leq n$.
- **[Inductive Hypothesis]** Assume for some arbitrary $k \geq 1$ that p is true. Thus $i < k$ (so we enter the loop again), and factorial= (i-1)!.
- **[Inductive Step]** Show p is still true after execution of the loop. Thus $i \leq k$ and factorial = i!.

Loop Invariant - Example

- First, i is incremented by 1
- Thus $i \leq k$ since we assumed $i < k$, and i and $k \geq 1$.
- Also, factorial, which was $(i - 1)!$ by IH, is set to $(i - 1)! * i = i!$
- Hence, p remains true.
- Since p remains true, p is a loop invariant and thus the assertion:
  - $[p \wedge (i < n)]\{S\}p$ is true
- It follows that the assertion:
  - $p\{\text{while } i < n \ S\}[(i \geq n) \wedge p]$ is also true.

Loop Invariant - Example

- Furthermore, the loop terminates after $n - 1$ iterations with $i = n$, since:
  - i is assigned the value 1 at the beginning of the program,
  - 1 is added to i during each iteration of the loop, and
  - the loop terminates when $i \geq n$
- Thus, at termination, factorial = $n!$.

- Note that: We split larger segments of code into component parts, and use the rule of composition to build the correctness proof.
  - $(p = p1)\{S1\}q1, q1\{S2\}q2, \ldots, qn-1\{Sn\}(qn = q) \rightarrow p\{S1; S2; \ldots; Sn\}q$

Questions?