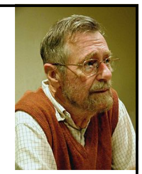
Correctness of Algorithm

CS 231

Dianna Xu

1

What does it mean for a program to be correct?



- Syntax errors
- Implementation errors
- Logical errors (algorithmic errors)
  – This part can be proved mathematically
  – "We now take the position that it is not only the programmer's task to produce a correct program, but also to demonstrate its correctness in a convincing manner" – Dijkstra, 1967

2

Predicates

- An algorithm is designed to produce a certain final state (post-condition) from a certain initial state (pre-condition).
- Proof of correctness: show that if the pre-condition is true for a collection of values, then the post-condition is also true.

3

Example

- Algorithm to compute a product of two nonnegative integers
  – Pre-condition: input variables x and y are non-negative integers
  – Post-condition: output variable $p = xy$

4

Correctness of a loop

- Method to prove the correctness of a loop
- Given a **while** loop, entry restricted by a condition G (guard).

```
Pre-condition for the loop
while (G)
  body
end while
Post-condition for the loop
```

5

Loop Invariant Theorem

- Given a predicate I(n), a loop is correct if:
  – Basis: I(0) is true before the first iteration of the loop
  – Inductive: For all integers $k \geq 0$, $G \wedge I(k)$ before any iteration $\rightarrow I(k+1)$ after the iteration
  – Eventual Guard Falsity: After a finite number of iterations, G becomes false
  – Correctness of post-condition: If I(N) is true when N is the least number of iterations after which G is false, the values of the algorithm variables will be as specified in the post-condition.

6

Loop to compute a product

Pre-condition: x and y are nonnegative integers, $i = 0$ and *product* = 0

```
while (i≠x)
    product := product + y
    i := i+1
end while
```

Post-condition: *product* = xy

Loop invariant: I(n): $i = n \wedge$ product = ny

7

Proof

- Base: I(0): i=0 and product = 0*y
- Inductive: G ∧ I(k) before iteration → I(k+1) after iteration
  - inductive hypothesis:
  - $i=k \wedge$ *product* = ky
  - inductive step:
  - *product* = *product* + y = $ky + y = (k+1)y$
  - $i = i+1 = k+1$

8

Proof

- Falsity of Guard: after x iterations, $i=x$
- Correctness of Post-condition:
  - N=x
  - i=N ∧ *product* = Ny
  - $i=x \wedge$ *product* = xy

9

Loop Invariant

- A statement of conditions that must be true on entry into a loop and are guaranteed to remain true after every iteration of the loop
- Inductive invariant
- Finding the right one is often the hardest part of proving the correctness of a loop
- Loop invariant and negated guard implies post-condition – must be strong enough

10

Loop

Pre-condition: $x = 0$, $i = 2$

```
while (i<=10)
    x := x + i*i
    i := i+1
end while
```

Post-condition: x = sum of squares of 2-10

Loop invariant: I(n): $i = n \wedge x = \sum_{i=2}^{n} i^2$

11

- Thinking about loops in terms of invariants help you avoid errors and bad practices:
  - off by one errors
  - wrong/missing code in the loop body
  - declarations of variables outside the loop that are only used inside the loop body

12

Finding the Max Element

```
Pre-condition: a1, a2…an ϵ Z, max:= a1
for (i:= 2 to n)
  if (max < ai) then max:= ai
  next i
Post-condition:
max = the largest value in {a}
```

Pre-condition: $a_1, a_2 \ldots a_n \in \mathcal{Z}$, $max := a_1$

for ($i := 2$ **to** n)

  if ($max < a_i$) **then** $max := a_i$

  next i

Post-condition:

max = the largest value in {a}

13