CSC236 Intro. to the Theory of Computation
Lecture 8: correctness proof of iterative programs

Amir H. Chinaei, Fall 2016
Office Hours: W 2-4 BA4222
ahchinaei@cs.toronto.edu
http://www.cs.toronto.edu/~ahchinaei/

Course page:
http://www.cof.toronto.edu/~cs236h/fall/index.html
Section page:
http://www.cof.toronto.edu/~cs236h/fall/amir_lectures.html

review

• Last lecture
  - correctness proof of recursive programs
    - based on the program specification
      - i.e., pre- & post- conditions
    - normally using induction
      - proof is parallel to the code

• this week
  - correctness proof of iterative programs
    - based on the program specification
      - i.e., pre- & post- conditions

Example 76: $x^n$

```python
def power(x, n):
    r = 1
    c = 0
    while c < n:
        r = r * x
        c = c + 1
    return r
```

How to start the proof, formally?
- recall, we need to show:
  - preconditions $\Rightarrow$ postconditions

Example 76: correctness of $x^n$:

- We want to show: preconditions $\Rightarrow$ postconditions
  $P(n)$: if $n \in \mathbb{N}, x \in \mathbb{R}$, then $\text{power}(x, n)$ terminates and returns $x^n$.

- Partial correctness:
  $P'(n)$: if $n \in \mathbb{N}, x \in \mathbb{R}$, and $\text{power}(x, n)$ terminates, then it returns $x^n$.

- To prove partial correctness of iterative algorithms, we use
  - loop invariant: an assertion (predicate) that must be true before and after each iteration of the loop

Example 76: pre- post- conditions

- $\text{power}(x, n)$:
  - preconditions:
    - $0 \leq n \in \mathbb{N}$
    - $x \in \mathbb{R}$
  - postconditions:
    - $\text{power}(x, n)$ terminates and returns $x^n$

Example 76: loop invariant

- We want to show: preconditions $\Rightarrow$ postconditions
  $\text{power}(x, n)$:
  $r = 1$
  $c = 0$
  while $c < n$:
      $r = r \times x$
      $c = c + 1$
  return $r$

- Loop invariant:
  $L_I(c, r): 0 \leq c \leq n$ and $r = x^c$
LI(c, r): 0 ≤ c ≤ n and r = x^c

**Recipe**

- show the LI holds before the loop starts
  - show LI(c₀, r₀) holds

- show the LI holds at each iteration of the loop
  - show LI(cᵢ, rᵢ) → LI(cᵢ₊₁, rᵢ₊₁)

- show the LI holds after the loop exits
  - by showing LI(c₀, r₀) holds,
    - conclude the postcondition is met.

---

**Example 76: Partial Correctness**

```python
def power(x, n):
    r = 1
    c = 0
    while c < n:
        r = r * x
        c = c + 1
    return r
```

---

**Example 76: Loop Variant**

```python
def power(x, n):
    r = 1
    c = 0
    while c < n:
        r = r * x
        c = c + 1
    return r
```

---

**Termination of power**

- To prove **termination** of iterative algorithms, we use
  - **loop variant**: a number, \( k_c \), associated with each iteration \( c \) of the loop, and we show
    - \( k_c \in \mathbb{N} \)
    - \( k_c \) is decreasing

---

**Example 76: Loop Variant**

```python
def power(x, n):
    r = 1
    c = 0
    while c < n:
        r = r * x
        c = c + 1
    return r
```

- **loop variant**:
  - \( k_c = n - c \)
Example 76: terminations

\[
k_c = n - c
\]

```python
def power(x, n):
    r = 1
    c = 0
    while c < n:
        r = r * x
        c = c + 1
    return r
```

notes

Example 77: iterative binSearch

```python
def iterativeBinSearch(x, A):
    # Precondition: A is a sorted array, and
    # A is not empty.
    # Postcondition: Return an integer p such that 0 <= p <= length(A)
    # and A[p] = x, if such a p exists; otherwise return -1.
    b = 0
    e = len(A) - 1
    while b != e:
        m = (b + e) // 2  # midpoint
        if x <= A[m]:
            e = m
        else:
            b = m + 1
    if A[b] == x:
        return b
    else:
        return -1
```

Example 77: pre-post-conditions

- **iteBinSearch(x, A):**
  - **preconditions:**
    - A is not empty
    - elements of A are sorted non-decreasingly
    - elements of A and x are comparable
  - **postconditions:**
    - iteBinSearch(x, A) terminates and returns p such that 0 ≤ p ≤ Length(A) − 1 and x = A[p] if such a p exists;
    - otherwise it terminates and returns −1.

Example 77: correctness of iteBinSearch:

- **loop invariant:**

```python
def iterativeBinSearch(x, A):
    b = 0
    e = len(A) - 1
    while b != e:
        m = (b + e) // 2  # midpoint
        if x <= A[m]:
            e = m
        else:
            b = m + 1
    if A[b] == x:
        return b
    else:
        return -1
```

Example 77: loop invariant

- **loop invariant:**

```
```

```
```
recipe

- show the LI holds before the loop starts
  - show \( LI(b_0, e_0) \) holds

- show the LI holds at each iteration of the loop
  - show \( LI(b_k, e_k) \Rightarrow LI(b_{k+1}, e_{k+1}) \)

- show the LI holds after the loop exits
  - and from there, conclude the postcondition is met.

We use induction to show the LI holds for all \( i \)'s.

Example 77: partial correctness

\( P(i) \): if the loop iterates at least \( i \) times, \( LI(b_i, e_i) \) holds.

```
1 def iteBinSearch(x, A):
2     b = 0
3     e = len(A)-1
4     while b != e:
5         m = (b + e) // 2
6         if x <= A[m]:
7             e = m
8         else:
9             b = m + 1
10        if A[b] == x:
11            return b
12        else:
13            return -1
```

LI(b_i, e_i): \( 0 \leq b_i \leq e_i \leq n - 1 \) and if \( x \) is in \( A \), it's in \( A[b_i..e_i] \)

Example 77: loop variant

```
1 def iteBinSearch(x, A):
2     b = 0
3     e = len(A)-1
4     while b != e:
5         m = (b + e) // 2
6         if x <= A[e]:
7             e = m
8         else:
9             b = m + 1
10        if A[b] == x:
11            return b
12        else:
13            return -1
```

an important task remains:

- Termination of iteBinSearch
  - To prove termination of iterative algorithms, we use
    - loop variant: a number, \( k_i \), associated with each iteration \( i \) of the loop, and we show
      - \( k_i \in \mathbb{N} \)
      - \( k_i \) is decreasing
Example 77: terminations

```python
def iteBinSearch(x, A):
    b = 0
    e = len(A) - 1
    while b != e:
        m = (b + e) // 2
        if x <= A[m]:
            e = m
        else:
            b = m + 1
    if A[b] == x:
        return b
    else:
        return -1
```