CIS 313 Intermediate Data Structures – Fall 2016 September 30th, 2016

Note on Loop Invariants

Lecturer: Xiaodi Wu *Scribe: Xiaodi Wu*

- **Reading:**[1] Chapter 1.3.3 and this note.
- This note is supplementary to the slides. Please read them both.

1 Loop Invariants

State of computation : (**boolean predicate**) a "snap-shot" of the computation; inter-relations of values of variables. Relations denoted by P, Q and corresponding states by $\langle P\rangle, \langle Q\rangle$.

Program Statement : (**predicate transformers**) a program statement S causes state $\langle P\rangle$ to state $\langle Q\rangle$. (denoted $\langle P\rangle\, S\, \langle Q\rangle$).

Examples:

- State: $\langle x = 0; y = 1\rangle$, $\langle x = y^2\rangle$, ...
- Program Statement: $x \leftarrow x + 1$, $x \leftarrow x \times y$, ...

Abstraction of a loop

$\langle P\rangle$, the state before the loop
while Condition (C) **do** Body (B)
end while
$\langle Q\rangle$, the state after the loop

Definition 1 *A* **loop invariant** I *is a boolean predicate that does not change during the execution of the loop. Moreover, we have*

- $P \to I$ *before the loop.*
- $\langle I$ *and* $C\rangle\, B\, \langle I\rangle$ *in the loop.* [2]
- $(I$ *and* $\neg C) \to Q$ *after the loop.*

We will demonstrate how to use loop invariants as a tool to prove the correctness of the computation.

1.1 Example 1

The following code calculates the summation of numbers from 0 to n:

$s \leftarrow 0, k \leftarrow 0$
while $k < n + 1$ **do**
  $s \leftarrow s + k, k \leftarrow k + 1$
end while

[1] Contents will appear in homework and exams.
[2] "and" denotes logical operation "AND", which is also denoted by && in Java, sometimes denoted by $\wedge$. Stick to one notation in your homework.

Thus, it is expected that $s = \sum_{k=0}^{n} = \frac{n(n+1)}{2}$ after the loop. Let us prove the statement with loop invariants.

Finding P and Q is not hard. P is the state of computation before the loop and one just needs to copy from that. Q is the statement you want to prove, i.e., $s = \sum_{k=0}^{n} = \frac{n(n+1)}{2}$ in this example and P is $s = 0, k = 0$. The hard part is to find the loop invariant I based on P, Q and the code. For this code, we will choose the following loop invariant:

$$I = (0 \leq k \leq n+1) \wedge (s = k(k-1)/2).$$

Let us see how we prove the loop invariant and use the loop invariant to prove the desired statement. It suffices to prove all the three items in Definition 1.

(i) P is $s = 0, k = 0$ which implies I. It suffices to substitute $s = 0, k = 0$ into I and directly verifies that.

(ii) $\langle I \text{ and } C \rangle\, B\, \langle I \rangle$: let s, k denote the values of variables s, k at the beginning of the loop and $\bar{s}, \bar{k}$ their values at the end of the loop. When C holds, namely, $k < n+1$, then $0 \leq \bar{k} = k+1 \leq n+1$. Because I holds, namely, $s = k(k-1)/2$, and $\bar{s} = s + k, \bar{k} = k + 1$, then we have

$$\bar{s} = s + k = k(k-1)/2 + k = k(k+1)/2 = \bar{k}(\bar{k}-1)/2.$$

Thus, we prove the loop invariant I holds.

(iii) When C does not hold, namely, $k = n+1$ when the loop terminates, we have $s = n(n+1)/2$ by plugging $k = (n+1)$ into the loop invariant I.

1.2 Example 2

The following code returns the index i such that $A[i] = x$ for given x and returns -1 if x is not stored in $A[0...n-1]$, where $A[0..(i-1)]$ refers to the values stored in array A from indices 0 through $i-1$, inclusive, for $i = 0, 1, \cdots$ and $A[0..-1]$ denotes an empty set.

```
i ← 0
while i < n do
    if x = A[i] then
        return i
    else
        i ← i + 1
    end if
end while
return -1
```

We want to prove that if the code returns -1, then x is not in A, with the help of loop invariants. Similar to Example 1, we can define P, Q, i.e., $P : i = 0$, $Q : x \notin A[0..n-1]$. The loop invariant I is defined to be $(0 \leq i \leq n) \wedge (x \notin A[0 \cdots (i-1)])$, i.e., "$0 \leq i \leq n$ and x is not stored in $A[0..(i-1)]$".

(i) P, $i = 0$. Clearly, $0 \leq 0 \leq n$. Because $i = 0$, $A[0..i-1] = \emptyset$ and the second half of the invariant is trivially true.

(ii) Let i refer to the value of variable i at the beginning of the loop and $\bar{i}$ to its value at the end of the loop. From the execution of the loop body, $\bar{i} = i + 1$. Since $0 \leq i$ (from I) and $i < n$ (from C), $0 \leq i + 1 \leq n$, so $0 \leq \bar{i} \leq n$. Furthermore, from I we know that x is not contained in $A[0..(i-1)]$. From the loop body, if the loop continues then x is not in $A[i]$ either. Therefore, x is not contained in $A[0..i]$. From the definition of $\bar{i}$, we can conclude that x is not contained in $A[0..(\bar{i}-1)]$ and the invariant remains true with the new value of i.

(iii) Since $0 \leq i \leq n$ (from I) and $i \not< n$ (from C), $i = n$. Substituting $i = n$ back into the loop invariant, we can conclude that x is not stored in $A[0..(n-1)]$, which is the entire array.

1.3 Example 3

Here is a more complicated example, where we can demonstrate a bit of the power of loop invariants.

$x \leftarrow a; y \leftarrow b; z \leftarrow 1$
while $y > 0$ **do**
  while even(y) **do**
    $x \leftarrow x^2; y \leftarrow y$ div 2
  end while
  $y \leftarrow y - 1; z \leftarrow z * x$
end while
return z.

1.3.1 Inner Loop

Let us look at the inner loop first.

$\langle P_1 \rangle$
while even(y) $[C_1]$ **do**
  $x \leftarrow x^2; y \leftarrow y$ div 2 $[B_1]$
end while
$\langle Q_1 \rangle$

One observes that x^y does not change during the loop. Here is the list of definitions of P_1, Q_1 and loop invariant I_1.

- $P_1 = (y > 0) \wedge (x^y = d)$ for some unknown but fixed d.
- $I_1 = (y > 0) \wedge (x^y = d)$.
- $Q_1 = (y > 0) \wedge (x^y = d) \wedge \text{odd}(y)$ for the same d.

It remains to formally prove the loop invariant.

(i) $P_1 \rightarrow I_1$ by definition.

(ii) $\langle I_1$ and $C_1 \rangle\, B_1\, \langle I_1 \rangle$: Let x, y refer to the value of variable x, y at the beginning of the loop and $\bar{x}, \bar{y}$ to its values at the end of the loop. When C_1 holds, $y = 2k$ for some natural number k. The loop body B_1 updates x to $\bar{x} = x^2$ and y to $\bar{y} = y$ div $2 = k$. It is easy to see that $\bar{y} > 0$ and

$$x^y = x^{2k} = (x^2)^k = \bar{x}^{\bar{y}}.$$

Hence, I_1 remains true.

(iii) (I_1 and $\neg C_1$) $\rightarrow Q_1$ by definition.

1.3.2 Outer Loop (Optional)

Here we demonstrate how to make use of the loop invariants of the inner loop to argue about the outer loop. Substitute the analysis of the inner loop.

$\langle P_2 \rangle$ $x \leftarrow a; y \leftarrow b; z \leftarrow 1$
while $y > 0$ $[C_2]$ **do**
  $\langle P_1 \rangle$: $(y > 0) \wedge (x^y = d)$.
  I_1 : $(y > 0) \wedge (x^y = d)$.
  $\langle Q_1 \rangle$: $(y > 0) \wedge (x^y = d) \wedge \text{odd}(y)$
  (the above d will change in each execution of the outer loop)
  $y \leftarrow y - 1; z \leftarrow z \times x$ $[B_2]$
end while
return z.
$\langle Q_2 \rangle$

- $I_2 : (y \geq 0)$ and $(z \times x^y = a^b)$.
- $P_2 \rightarrow z \times x^y = a^b$.
- $Q_2 : y = 0; z \times x^y = a^b \rightarrow z = a^b$.

In the outer loop, the loop condition is C_2 and the transition from P_1 to Q_1 plus statement B_2 is the body of the loop.

(i) $P_2 : x = a; y = b; z = 1$ implies I_2 by substituting the values of x, y, z.

(ii) Let x, y, z refer to the value of variable x, y, z at the beginning of the loop, x_0, y_0, z_0 to their values at Q_1 before B_2 is executed. $\bar{x}, \bar{y}, \bar{z}$ to their values at the end of the loop.

At state $\langle P_1 \rangle$, we know $z \times x^y = a^b$. Because of the loop invariant of the inner loop, when the computation reaches state $\langle Q_1 \rangle$, y_0 is positive and odd and $x_0^{y_0} = x^y$. Because there is no change of z during the inner loop, so $z_0 = z$. Thus, $z_0 \times x_0^{y_0} = zx^y = a^b$ at $\langle Q_1 \rangle$.

It then suffices to see what happens with B_2, which updates y_0 to $\bar{y} = y_0 - 1$, z_0 to $\bar{z} = z_0 \times x_0$, and $\bar{x} = x_0$ remains unchanged. Thus,

$$z \times x^y = z_0 \times x_0^{y_0} = z_0 \times x_0 \times x_0^{y_0 - 1} = \bar{z}\bar{x}^{\bar{y}}.$$

Because y_0 is positive, thus $\bar{y} \geq 0$. Thus, we have I_2.

(iii) When C_2 does not hold, which implies $y = 0$ and the loop terminates, we have $x^y = 1$ and $z \times x^y = a^b = z$.

Remarks

There are a few remarks about loop invariants.

- First, we didn't teach how to design loop invariants but rather to provide a framework that verify loop invariants and then the correctness of the computation.
- Thus, the right order of using this method is first to analyze the code and get some hypothesis about what the computation is doing. Then one comes up with loop invariants and verifies the correctness using the loop invariant framework.
- There could be many loop invariants for the same code. One needs to first identify the desired after-loop statement and then to pick the appropriate loop invariant.