Section 2: Reasoning About Loops

Fall 2017

Loop Invariants

{{ Inv: I }}

```
while (cond)
    S
```

- A loop invariant is a statement that always holds at the top of a loop
  - It holds when we first get to the loop
  - It holds each time we execute `S` and come back to the top
- Loop invariants are necessary for checking the validity of a while loop

While Loop Rule

{{ P }} `while (cond) S` {{ Q }}

Triple is valid if and only if there is a loop invariant I such that:

{{ P }}
{{ Inv: I }}
`while (cond)`
`  S`
{{ Q }}

- I holds initially
- I holds each time we execute `S`
- Q holds when I holds and `cond` is false

Example 1 - Assertions

```
{{ }}
int v = 1;
{{ v = 1 }}
int i = A.length;
{{ v = 1 and i = A.length }}
{{ Inv: v = A[i] * A[i+1] * … * A[A.length-1] }}
while (i != 0) {
    {{ v = A[i] * A[i+1] * … * A[A.length-1] and i != 0}}
    i = i - 1;
    {{ v = A[i+1] * A[i+2] * … * A[A.length-1] and i != -1 }}
    v = A[i] * v;
    {{ v = A[i] * A[i+1] * A[i+2] * … * A[A.length-1] and i != -1 }}
}
{{ v = A[0] * A[1] * … * A[A.length-1] }}
```

1. Does the invariant hold initially?

   (v = 1 and i = A.length) → (v = A[i] * A[i+1] * … * A[A.length-1])
   (Empty product is 1)

2. Is the invariant preserved through the loop body?

   (v = A[i] * A[i+1] * A[i+2] * … * A[A.length-1] and i != -1)
   → (v = A[i] * A[i+1] * … * A[A.length -1])

3. Does the postcondition hold on termination?

   (v = A[i] * A[i+1] * … * A[A.length-1] and i = 0)
   → (v = A[0] * A[1] * A[2] * … * A[A.length -1])

Example 2 - Find where proof fails

```
{{n = A.length}}
public void replaceZeroes(int[] A, int n) {
int i = n - 1;
{{ i = n - 1 and n = A.length }}
{{ Inv: A[n - 1] != 0, … , A[i] != 0 }}
while (i > 0) {
     {{ A[n - 1] != 0, … , A[i] != 0 and i > 0 }}
     i--;
     {{ A[n - 1] != 0, … , A[i + 1] != 0 }}
     if (A[i] == 0) {
          {{ A[n - 1] != 0, … , A[i + 1] != 0 }}
          A[i] = 1;
          {{ A[n - 1] != 0, … , A[i] != 0 }}
     }
     {{ A[n - 1] != 0, … , A[i] != 0 }}
}
{{ A[n - 1] != 0, … , A[0] != 0 }}
```

1. Does the invariant hold initially?
2. Does the invariant hold after the loop body is executed?
3. Does the invariant imply the post-condition upon termination of the loop?

Since `i` is initially `n - 1` and the invariant states that the array `A` must have non-zero values from `n - 1` to `i`, there is no assertion prior to the loop which guarantees the index `n - 1` will be non-zero. In fact, at no point in this program does the index `n - 1` ever become updated.

Example 3 - Given invariant, fill in code

Fill in code to return the count of even numbers in array `A`.

```
{{ }}
int count = 0, i = 0;
{{ Inv: count stores the number of even numbers in A[0],...,A[i-1] }}
while ( i != A.length ) {
    {{ count stores the number of even numbers in A[0],...,A[i-1 }}
    if (A[i] % 2 == 0) {
        {{ count stores the number of even numbers in A[0],...,A[i-1] and A[i] is even }}
        count++;
        {{ count stores the number of even numbers in A[0],...,A[i-1],A[i] }}
    }
    {{ count stores the number of even numbers in A[0],...,A[i]}}
    i++;
    {{ count stores the number of even numbers in A[0],...,A[i-1] }}
}
{{ count stores the number of even numbers in A[0],...,A[A.length-1] }}
```

Example 4 - Fill in invariant and code

Fill in implementation for method, `copyArray` - all loops must provide loop invariant.

```
{{ n < src.length and n < dst.length }}
void copyArray(int[] src, int[] dst, int n) {
    i = 0;
    {{ i = 0 }}
    {{ Inv: dst[0] = src[0], … , dst[i-1] = src[i-1] }}
    while (i < n) {
        {{ dst[0] = src[0], … , dst[i-1] = src[i-1] and i < n }}
        dst[i] = src[i];
        {{ dst[0] = src[0], … , dst[i] = src[i] }}
        i++;
        {{ dst[0] = src[0], … , dst[i-1] = src[i-1] }}
    }
    {{ dst[0] = src[0], … , dst[n-1] = src[n-1] }}
}
```

Solutions to Worksheet Problems

Problem 1

```
{{ n >= 0 and i >= 0 and i + n <= A.length }}
int moveFront(int[] A, int i, int n, int x) {
      int L = i;
      {{ L = i and n >= 0 and i >= 0 and i + n <= A.length }}
      int R = i + n;
      {{ R = i + n and L = i and n >= 0 and i >= 0 and i + n <= A.length }}
      {{ Inv: A[i], ..., A[L-1] <= x < A[R], ..., A[i+n-1] }}
      while (L != R) {
            {{ A[i], …, A[L-1] <= x < A[R], …, A[i+n-1] and L != R}}
            if (A[L] > x) {
                  {{ A[i],..., A[L-1] <= x < A[L], A[R], …,A[i+n-1] }}
                  swap(A[L], A[R - 1]);
                  {{ A[i], …, A[L-1] <= x < A[R-1], A[R], …, A[i+n-1] }}
                  R--;
                  {{ A[i], …, A[L-1] <= x < A[R], …, A[i+n-1] }}
            } else {
                  {{ A[i], …, A[L-1], A[L] <= x < A[R], …, A[i+n-1] }}
                  L++;
                  {{ A[i], …, A[L-2], A[L-1] <= x < A[R], …, A[i+n-1] }}
            }
            {{ A[i], …, A[L-1] <= x < A[R], …, A[i+n-1] }}
      }
      {{ A[i], ..., A[L-1] <= x < A[L], ..., A[i+n-1] }}
      return L-1;
}
```

Explanation through swap:
The swap statement switches the positions of A[L] and A[R-1], thus in the assertion following the swap, we see that A[L] has been replaced A[R-1].

1. Does the invariant hold initially?
2. Is the invariant preserved through the loop body?
3. Does the postcondition hold on termination?

Problem 2

1. Does the invariant hold initially?
2. Does the invariant hold after the loop body is executed?
3. Does the invariant imply the post-condition upon termination of the loop?

```
{{ 0 < n <= A.length }}
void reverse(int[] A, int n) {
    i = -1;
    j = n;
    {{ i = -1, j = n }}
    {{ Inv: A[0] = A[n-1]_1, …, A[i] = A[n-1-i]_1 and A[j] = A[n-1-j]_1, …, A[n-1] = A[0]_1 and j = n-1-i }}
    while (i < j) {
        {{ A[0] = A[n-1]_1, …, A[i] = A[n-1-i]_1 and A[j] = A[n-1-j]_1, …, A[n-1] = A[0]_1 and j = n-1-i }}
        i = i + 1;
        {{ A[0] = A[n-1]_1, …, A[i-1] = A[n-1-i+1]_1 and A[j] = A[n-1-j]_1, …, A[n-1] = A[0]_1 and j-1 = n-1-i }}
        j = j - 1;
        {{ A[0] = A[n-1]_1,…,A[i-1] = A[n-1-i+1]_1, A[i] = A[n-1-j]_1, A[j] = A[n-1-i]_1, A[j+1] = A[n-1-j-1]_1,…,A[n-1] = A[0]_1, j = n-1-i }}
        swap A[i], A[j];
        {{ A[0] = A[n-1]_1,…,A[i-1] = A[n-1-i+1]_1, A[i] = A[n-1-i]_1, A[j] = A[n-1-j]_1, A[j+1] = A[n-1-j-1]_1,…, A[n-1] = A[0]_1, j = n-1-i }}
    }
    {{ A[0] = A[n-1]_1, …, A[n-1] = A[0]_1 }}
}
```

The loop performs an additional swap when dealing with an even number of elements (when n is even). On the last iteration of the loop, the values will swap an additional time thus violating the loop invariant since A[i] will no longer be equal the original value at A[n-1-i] and A[j] will no longer be equal to the original value at A[n-1-j].

Problem 3

{{ n >= 0 and n = dst.length - 1}}

```
void sortedInsert(int[] dst, int src, int n) {
     int i = 0;

     {{ Inv: dst[0], …, dst[i-1] < src and dst is sorted  }}
     while ( i < n && dst[i] < src ) {
          i++;
     }
     {{ (dst[0], …, dst[n-1] < src or dst[0],..., dst[i-1] < src <= dst[i]) and dst is sorted }}
     int j = n + 1;

     {{ Inv: dst[n] = dst[n-1]0, …, dst[j] = dst[j-1]0  }}
     while (j > i + 1) {
          j--;
          dst[j] = dst[j-1];
     }

     dst[i] = src;
}
```