

Lecture 9

Program Correctness: Strategy

Zvonimir Rakamarić
University of Utah

slides acknowledgements: Z. Manna, R. Leino

Announcements

- ▶ Project proposals due at midnight
 - ▶ References (**and only references**) can be on page 2

Last Time

- ▶ Loops
- ▶ Loop Invariants
- ▶ PiVC verifying compiler with examples

While Loop with Invariant

```
while E
  invariant J
do
  S
end
```

The diagram illustrates the components of a while loop with annotations. A callout box labeled "loop condition" points to the expression E . A callout box labeled "loop invariant" points to the expression J . A callout box labeled "loop body" points to the statement S .

- ▶ Loop body S executed as long as loop condition E holds
- ▶ Loop invariant J must hold on every iteration
 - ▶ J must hold initially and is evaluated before E
 - ▶ J must hold even on final iteration when E is false
 - ▶ Provided by a user or inferred automatically

Desugaring While Loop Using Invariant

▶ while E invariant J do S end

assert J;

check that the loop invariant holds initially

havoc x; assume J;

jump to an arbitrary iteration of the loop

(
 assume E; S; assert J; assume false

where x denotes the assignment targets of S

□
 assume $\neg E$
)

check that the loop invariant is maintained by the loop body

exit the loop

This Time

- ▶ Examples, examples, examples...
- ▶ Some strategies for proving correctness

Example: Swap

- ▶ **Signature:**

```
int[] swap(int[] a, int i, int j)
```

- ▶ **Specification:**

- ▶ Swaps elements of array a at indices i and j and returns the updated array

- ▶ **Note:**

- ▶ No need for loops, arrays in P_i are copied on assignment

While Loop in Pi

```
i := 0;
while
  @L: 0 <= i && i <= n
  (i < n)
{
  a[i] := i;
  i := i + 1;
}
```


For Loop in Pi

```
for
```

```
  @L: 0 <= i && i <= n
```

```
  (int i := 0; i < n; i := i + 1)
```

```
{
```

```
  a[i] := i;
```

```
}
```

Quantifiers in Pi

```
exists x. (0 <= x && x <= 10 &&
  a[x] = 0)
```

```
forall x. (0 <= x && x <= 10 ->
  a[x] = 0)
```

(Dumb) Example: Multiply by 2

```
int Multiply2(int n) {  
  int r := 0;  
  for  
    (int i := 0; i < n; i := i + 1)  
  {  
    r := r + 2;  
  }  
  return r;  
}
```

► Specification:

- Given a non-negative integer n , function `Multiply2` multiplies it by 2

Example: Initialize Array

- ▶ **Signature:**

```
int[] InitializeArray(int[] a, int e)
```

- ▶ **Specification:**

- ▶ Initializes elements of array `a` to `e` and returns the updated array

Example: Linear Search I

- ▶ **Signature:**

```
bool LinearSearch(int[] a, int e)
```

- ▶ **Specification:**

- ▶ Returns `true` if `e` is found in array `a`, otherwise returns `false`

Example: Linear Search II

- ▶ **Signature:**

```
bool LinearSearch(int[] a, int l,  
                 int u, int e)
```

- ▶ **Specification:**

- ▶ Returns `true` if `e` is found in array `a` between `l` and `u`, otherwise returns `false`

Binary Search

```
bool BinarySearch(int[] a, int l, int u, int e) {
    if (l > u)
        return false;
    else {
        int m := (l + u) div 2;
        if (a[m] = e)
            return true;
        else if (a[m] < e)
            return BinarySearch(a, m + 1, u, e);
        else
            return BinarySearch(a, l, m - 1, e);
    }
}
```

Bubble Sort

```
@pre true
@post sorted(rv, 0, |rv| - 1)
int[] BubbleSort(int[] a_0) {
    int[] a := a_0;
    for
        @ true
        (int i := |a| - 1; i > 0; i := i - 1)
    {
        for
            @ true
            (int j := 0; j < i; j := j + 1)
        {
            if (a[j] > a[j + 1]) {
                int t := a[j];
                a[j] := a[j + 1];
                a[j + 1] := t;
            }
        }
    }
    return a;
}
```