

Lecture 10

Loops and Loop Invariants

Zvonimir Rakamarić
University of Utah

slides acknowledgements: Z. Manna, R. Leino

Last Time

- ▶ Design by contract
- ▶ Procedures

Desugaring Procedure Call

► **procedure** $M(x,y,z)$ **returns** (r,s,t)
 requires P
 ensures Q
 $\{S\}$

► **call** $a,b,c := M(E,F,G)$
 desugared into:
 $x' := E; y' := F; z' := G;$
 assert $P';$
 assume $Q';$
 $a := r'; b := s'; c := t';$

where:

- x',y',z',r',s',t' are fresh variables
- P' is P with x',y',z' for x,y,z
- Q' is Q with x',y',z',r',s',t' for x,y,z,r,s,t

Desugaring Procedure Implementation

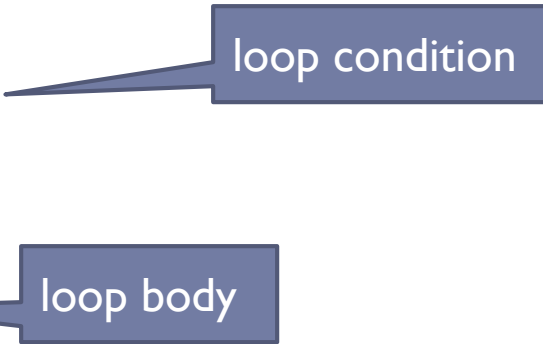
- ▶ procedure $M(x,y,z)$ returns (r,s,t)
requires P
ensures Q
 $\{S\}$
- ▶ Implementation is correct if this is correct:
assume P ;
 S ;
assert Q ;

This Time

- ▶ Loops
- ▶ Loop Invariants
- ▶ Strategies for proving programs correct

While Loop

```
while E
do
  S
end
```



The diagram illustrates the components of a while loop. A blue box labeled 'loop condition' has a pointer to the variable 'E' in the 'while E' line. Another blue box labeled 'loop body' has a pointer to the variable 'S' in the 'do S' line.

- ▶ Loop body S executed as long as loop condition E holds

Desugar While Loop by Unrolling N Times

`while E do S end =`

`if E {`

`S;`

`if E {`

`S;`

`if E {`

`S;`

`if E {assume false;} // blocks execution`

`}`

`}`

`}`

Example

```
i := 0;  
while i < 2 do i := i + 1 end
```

```
i := 0;  
if i < 2 {  
  i := i + 1;  
  if i < 2 {  
    i := i + 1;  
    if i < 2 {  
      i := i + 1;  
      if i < 2 {assume false;} // blocks execution  
    }  
  }  
}  
}
```


First Issue with Unrolling

```
i := 0;  
while i < 4 do i := i + 1 end
```

```
i := 0;  
if i < 4 {  
  i := i + 1;  
  if i < 4 {  
    i := i + 1;  
    if i < 4 {  
      i := i + 1;  
      if i < 4 {assume false;} // blocks execution  
    }  
  }  
}
```

Second Issue with Unrolling

```
i := 0;  
while i < n do i := i + 1 end
```

```
i := 0;  
if i < n {  
  i := i + 1;  
  if i < n {  
    i := i + 1;  
    if i < n {  
      i := i + 1;  
      if i < n {assume false;} // blocks execution  
    }  
  }  
}
```

While Loop with Invariant

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

Diagram illustrating the components of a while loop with an invariant:

- while E**: *E* is the loop condition.
- invariant J**: *J* is the loop invariant.
- do**: The start of the loop body.
- S**: The loop body.
- end**: The end of the loop.

- ▶ Loop body *S* executed as long as loop condition *E* holds

While Loop with Invariant cont.

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

Diagram illustrating the components of a while loop with an invariant:

- while E**: *E* is the loop condition.
- invariant J**: *J* is the loop invariant.
- do**: The start of the loop body.
- S**: The loop body.
- end**: The end of the loop.

- ▶ **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
- ▶ **Loop invariant** *J* must be *inductive*
 - ▶ Must be able to prove it by just assuming it

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

(

where x denotes the
assignment targets of S

assume E; S; assert J; assume false

□

assume $\neg E$

check that the loop invariant is
maintained by the loop body

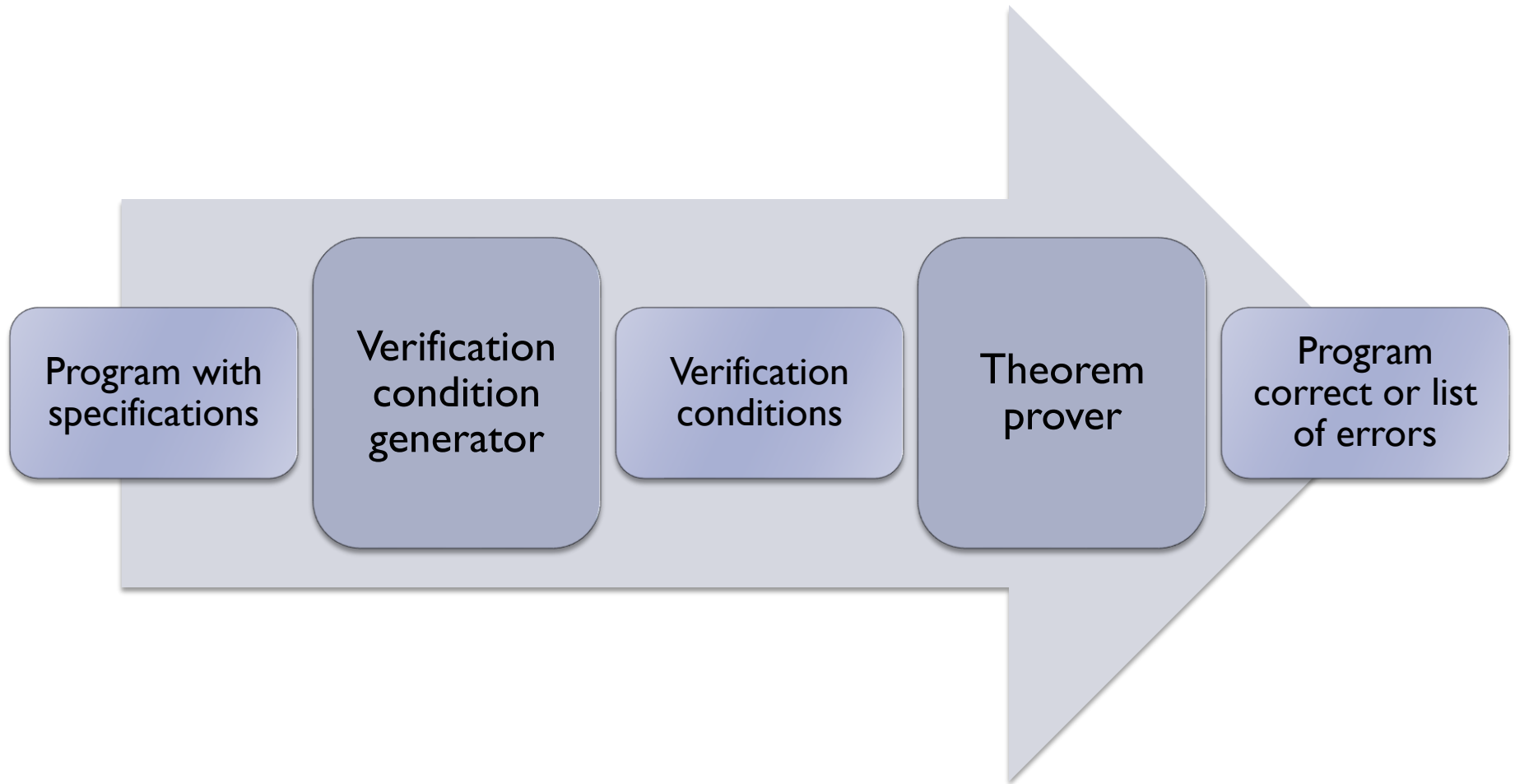
)

exit the loop

Dafny

- ▶ Simple “verifying compiler”
 - ▶ Proves procedure contracts statically for all possible inputs
 - ▶ Uses theory of weakest preconditions
- ▶ Input
 - ▶ Annotated program written in simple imperative language
 - ▶ Preconditions
 - ▶ Postconditions
 - ▶ Loop invariants
- ▶ Output
 - ▶ Correct or list of failed annotations

Dafny Architecture



Proving Correctness: Strategies

- ▶ Read Chapter 6
- ▶ Heuristics, requires intuition and practice
- ▶ Loop invariants are key and typically hardest part
- ▶ Strategies
 - ▶ Include all basic simple facts
 - ▶ For example, loop counter should be between 0 and n
 - ▶ Come up with complex invariants using the “precondition method”
 - ▶ Figure out which fact is failing, and compute its weakest precondition up to loop header
 - ▶ Comes more naturally with practice

(Dumb) Example: Multiply by 2

```
method Multiply2(n:int) returns (r:int)
{
    r := 0;
    var i:int := 0;
    while (i < n)
    {
        r := r + 2;
        i := i + 1;
    }
}
```

► Specification:

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

Example: Initialize Array

- ▶ Signature:

`InitializeArray(a:array<int>, e:int)`

- ▶ Specification:

- ▶ Initializes elements of array **a** to **e**

Vacuous Proof in Dafny

- ▶ Show an example
- ▶ Implications

Example: Linear Search

- ▶ Signature:

LinearSearch(**a**:array<int>, **l**:int,
u:int, **e**:int) returns (**r**:bool)

- ▶ Specification:

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