# Hardware & Software Verification

John Wickerson & Pete Harrod

Lecture 3

# Last lecture

- We need to be able to **reason about** the programs we write, not merely **test** them. There is a large and growing need for this.
- Dafny is a **verification-oriented** programming language. Its compiler will refuse to produce executable code until it has proven the code to be **correct**.

# But what does correct mean?

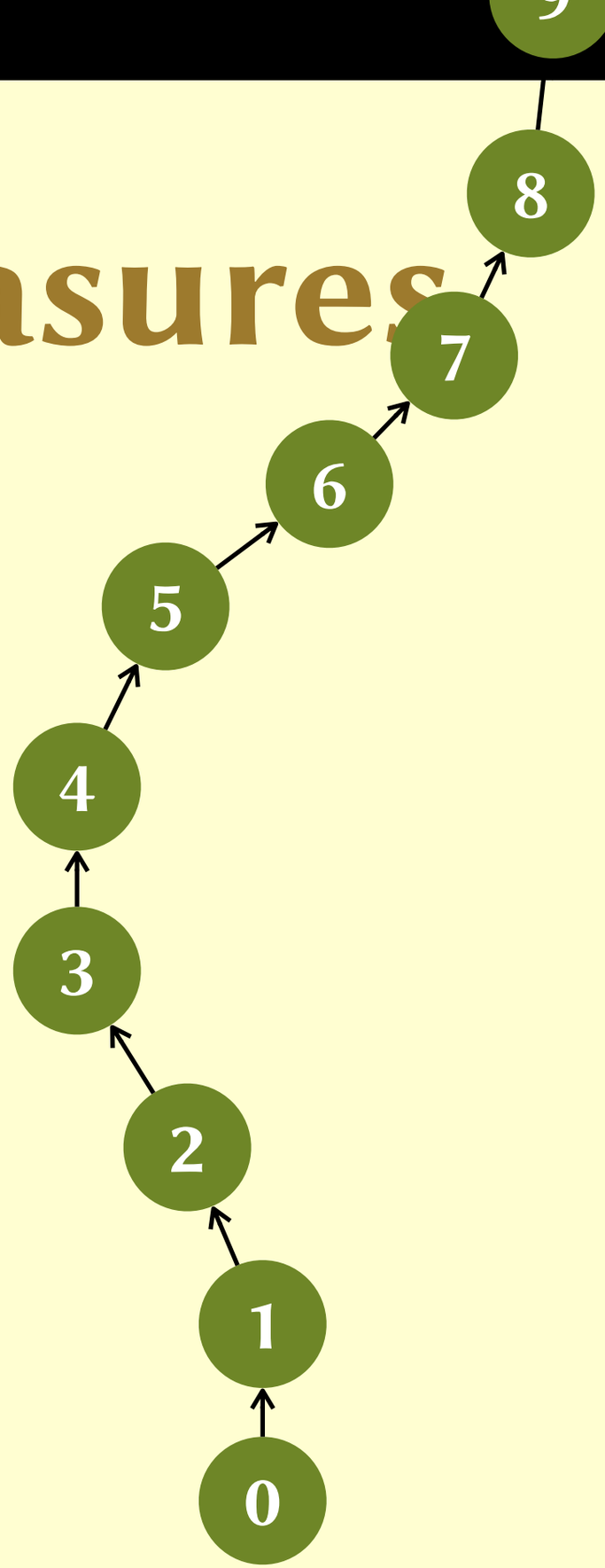
# Demo: max of a pair

- named output parameters
- postconditions
- overly weak/strong specifications

# Demo: max of an array

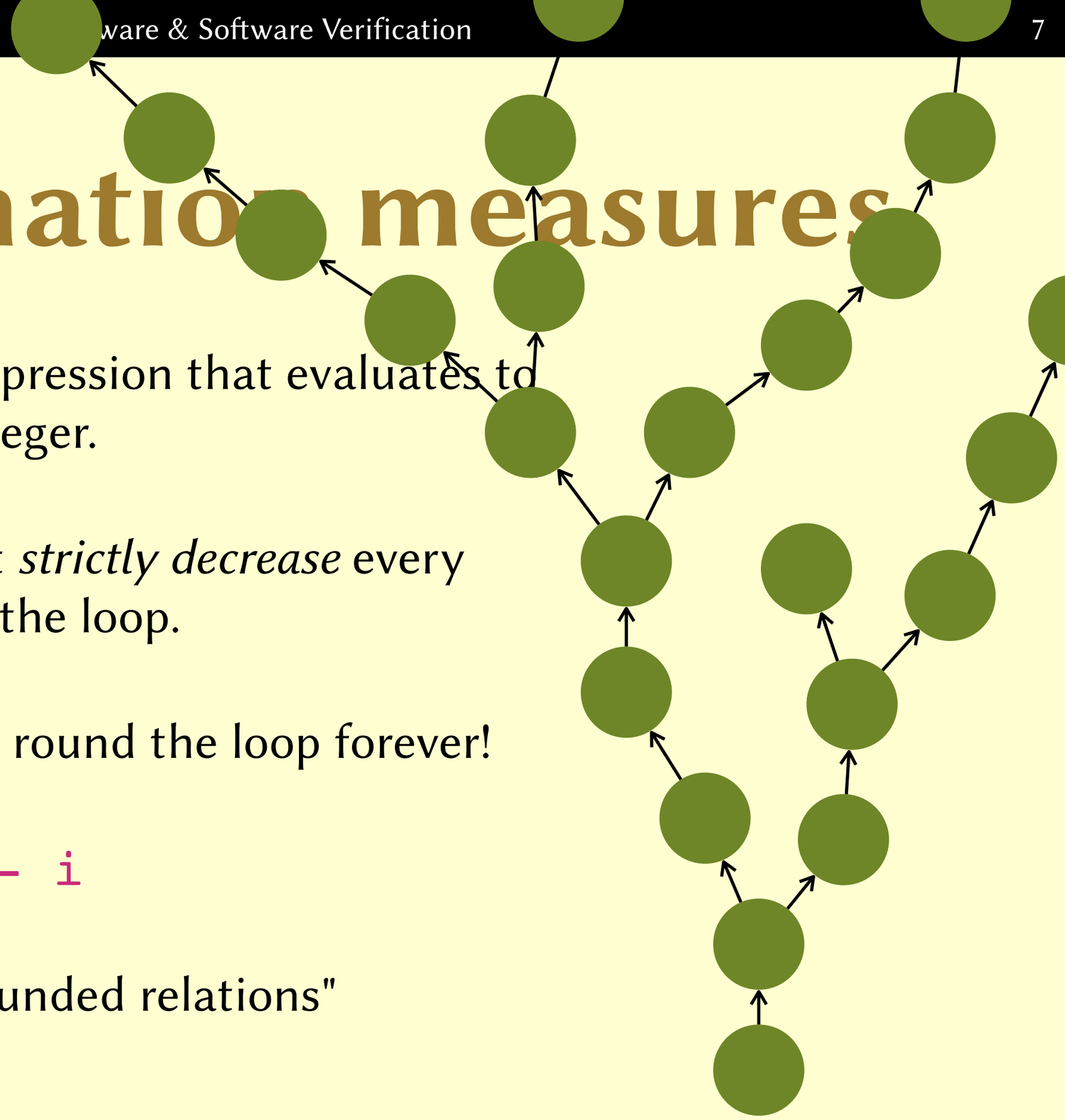
# Termination measures

- A *measure* is an expression that evaluates to a non-negative integer.
- The measure must *strictly decrease* every time we go round the loop.
- Hence we can't go round the loop forever!
- E.g.: `A.Length - i`
- "Theory of well-founded relations"



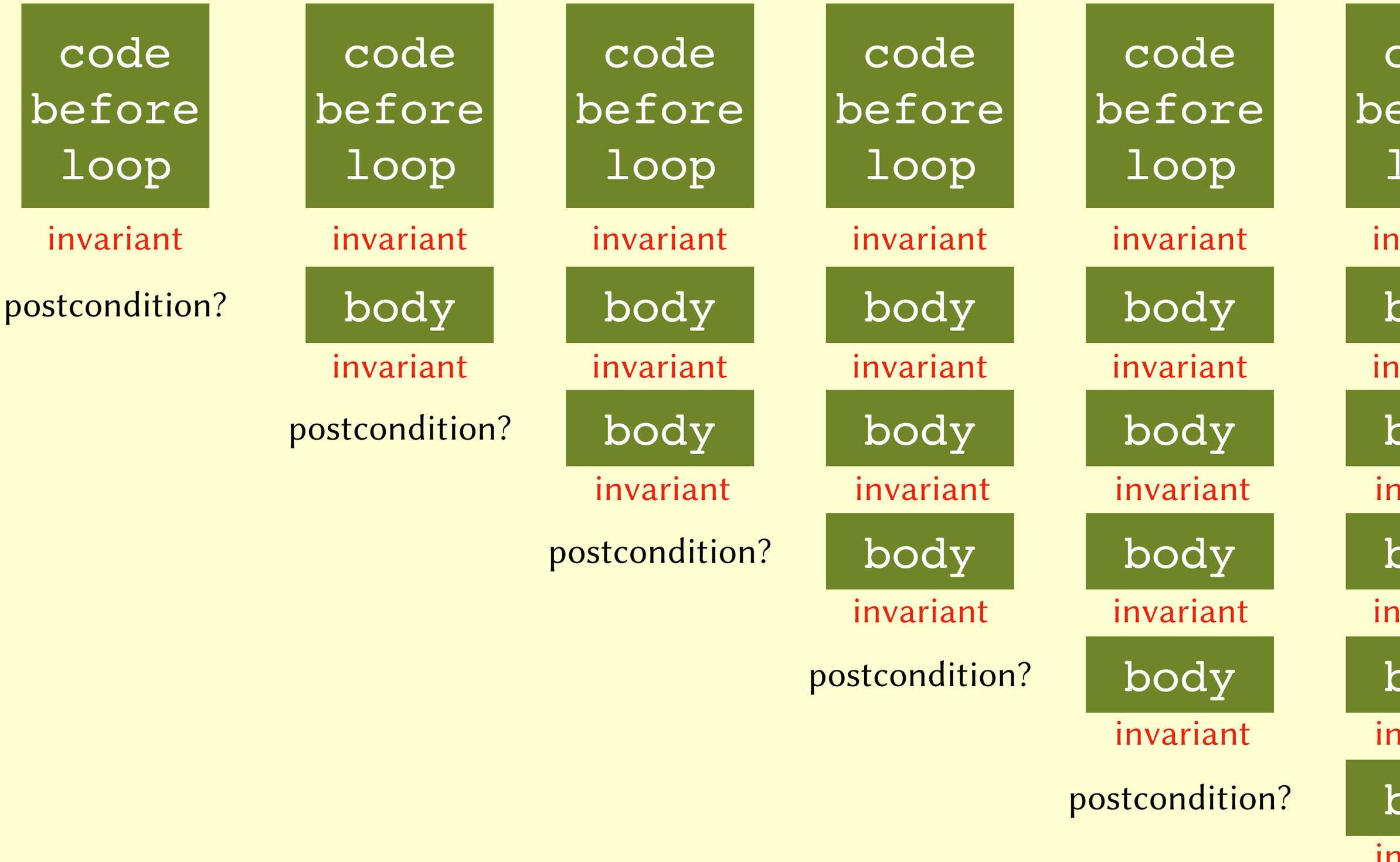
# Termination measures

- A *measure* is an expression that evaluates to a non-negative integer.
- The measure must *strictly decrease* every time we go round the loop.
- Hence we can't go round the loop forever!
- E.g.: `A.Length - i`
- "Theory of well-founded relations"



# Demo: max of an array

# The problem with loops



# Loop invariants

1.

`code before loop`

⇓

invariant

2.

invariant

`body`

⇓

invariant

3.

invariant

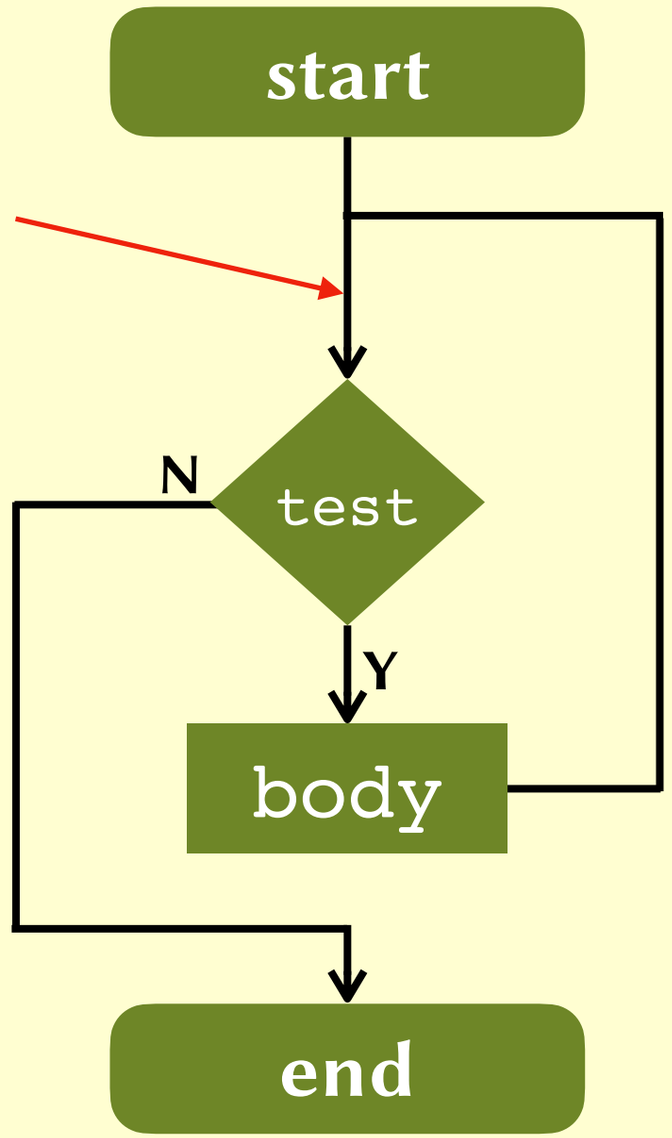
⇓

postcondition

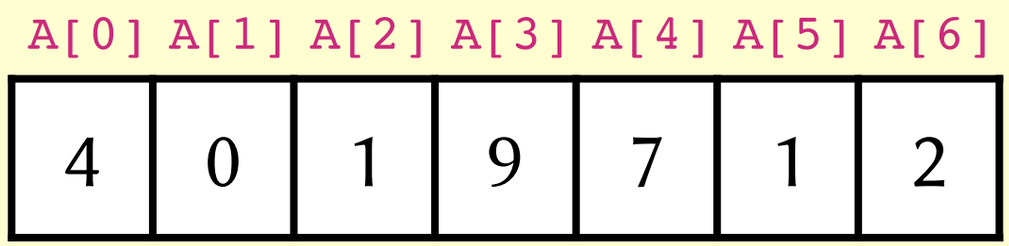
# Loop invariants

```
while test
   invariant foo
{
   body
}
```

foo must hold here!



# Finding invariants




```
r := A[0];
var i := 1;
while i < A.Length {
   if r < A[i] {
      r := A[i];
   }
   i := i+1;
}
```

| i | r |
|---|---|
| 1 | 4 |
| 2 | 4 |
| 3 | 4 |
| 4 | 9 |
| 5 | 9 |
| 6 | 9 |
| 7 | 9 |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
   if r < A[i] {
      r := A[i];
   }
   i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] |
|---|---|---|
| 1 | 4 | |
| 2 | 4 | |
| 3 | 4 | |
| 4 | 9 | |
| 5 | 9 | |
| 6 | 9 | |
| 7 | 9 | |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
  if r < A[i] {
    r := A[i];
  }
  i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] |
|---|---|---|
| 1 | 4 | ✔ |
| 2 | 4 | |
| 3 | 4 | |
| 4 | 9 | |
| 5 | 9 | |
| 6 | 9 | |
| 7 | 9 | |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
  if r < A[i] {
    r := A[i];
  }
  i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] |
|---|---|---|
| 1 | 4 | ✔ |
| 2 | 4 | ✔ |
| 3 | 4 | |
| 4 | 9 | |
| 5 | 9 | |
| 6 | 9 | |
| 7 | 9 | |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
  if r < A[i] {
    r := A[i];
  }
  i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] |
|---|---|---|
| 1 | 4 | ✔ |
| 2 | 4 | ✔ |
| 3 | 4 | ✔ |
| 4 | 9 | |
| 5 | 9 | |
| 6 | 9 | |
| 7 | 9 | |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
  if r < A[i] {
    r := A[i];
  }
  i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] |
|---|---|---|
| 1 | 4 | ✔ |
| 2 | 4 | ✔ |
| 3 | 4 | ✔ |
| 4 | 9 | ✔ |
| 5 | 9 | |
| 6 | 9 | |
| 7 | 9 | |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
  if r < A[i] {
    r := A[i];
  }
  i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] |
|---|---|---|
| 1 | 4 | ✔ |
| 2 | 4 | ✔ |
| 3 | 4 | ✔ |
| 4 | 9 | ✔ |
| 5 | 9 | ✔ |
| 6 | 9 | |
| 7 | 9 | |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
  if r < A[i] {
    r := A[i];
  }
  i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] |
|---|---|---|
| 1 | 4 | ✔ |
| 2 | 4 | ✔ |
| 3 | 4 | ✔ |
| 4 | 9 | ✔ |
| 5 | 9 | ✔ |
| 6 | 9 | ✔ |
| 7 | 9 | |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
  if r < A[i] {
    r := A[i];
  }
  i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] |
|---|---|---|
| 1 | 4 | ✔ |
| 2 | 4 | ✔ |
| 3 | 4 | ✔ |
| 4 | 9 | ✔ |
| 5 | 9 | ✔ |
| 6 | 9 | ✔ |
| 7 | 9 | ✔ |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
   if r < A[i] {
      r := A[i];
   }
   i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] | 1 ≤ i ≤ A.Length |
|---|---|---|---|
| 1 | 4 | ✔ | |
| 2 | 4 | ✔ | |
| 3 | 4 | ✔ | |
| 4 | 9 | ✔ | |
| 5 | 9 | ✔ | |
| 6 | 9 | ✔ | |
| 7 | 9 | ✔ | |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
   if r < A[i] {
      r := A[i];
   }
   i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] | 1 ≤ i ≤ A.Length |
|---|---|---|---|
| 1 | 4 | ✔ | ✔ |
| 2 | 4 | ✔ | |
| 3 | 4 | ✔ | |
| 4 | 9 | ✔ | |
| 5 | 9 | ✔ | |
| 6 | 9 | ✔ | |
| 7 | 9 | ✔ | |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
   if r < A[i] {
      r := A[i];
   }
   i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] | 1 ≤ i ≤ A.Length |
|---|---|---|---|
| 1 | 4 | ✔ | ✔ |
| 2 | 4 | ✔ | ✔ |
| 3 | 4 | ✔ | |
| 4 | 9 | ✔ | |
| 5 | 9 | ✔ | |
| 6 | 9 | ✔ | |
| 7 | 9 | ✔ | |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
   if r < A[i] {
      r := A[i];
   }
   i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] | 1 ≤ i ≤ A.Length |
|---|---|---|---|
| 1 | 4 | ✔ | ✔ |
| 2 | 4 | ✔ | ✔ |
| 3 | 4 | ✔ | ✔ |
| 4 | 9 | ✔ | |
| 5 | 9 | ✔ | |
| 6 | 9 | ✔ | |
| 7 | 9 | ✔ | |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
  if r < A[i] {
    r := A[i];
  }
  i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] | 1 ≤ i ≤ A.Length |
|---|---|---|---|
| 1 | 4 | ✔ | ✔ |
| 2 | 4 | ✔ | ✔ |
| 3 | 4 | ✔ | ✔ |
| 4 | 9 | ✔ | ✔ |
| 5 | 9 | ✔ | |
| 6 | 9 | ✔ | |
| 7 | 9 | ✔ | |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
  if r < A[i] {
    r := A[i];
  }
  i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] | 1 ≤ i ≤ A.Length |
|---|---|---|---|
| 1 | 4 | ✔ | ✔ |
| 2 | 4 | ✔ | ✔ |
| 3 | 4 | ✔ | ✔ |
| 4 | 9 | ✔ | ✔ |
| 5 | 9 | ✔ | ✔ |
| 6 | 9 | ✔ | |
| 7 | 9 | ✔ | |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
  if r < A[i] {
    r := A[i];
  }
  i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] | 1 ≤ i ≤ A.Length |
|---|---|---|---|
| 1 | 4 | ✔ | ✔ |
| 2 | 4 | ✔ | ✔ |
| 3 | 4 | ✔ | ✔ |
| 4 | 9 | ✔ | ✔ |
| 5 | 9 | ✔ | ✔ |
| 6 | 9 | ✔ | ✔ |
| 7 | 9 | ✔ | |

# Finding invariants

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] |
|---|---|---|---|---|---|---|
| 4 | 0 | 1 | 9 | 7 | 1 | 2 |

```
r := A[0];
var i := 1;
while i < A.Length {
  if r < A[i] {
    r := A[i];
  }
  i := i+1;
}
```

| i | r | ∃j. 0≤j<i ∧ r=A[j] | 1 ≤ i ≤ A.Length |
|---|---|---|---|
| 1 | 4 | ✔ | ✔ |
| 2 | 4 | ✔ | ✔ |
| 3 | 4 | ✔ | ✔ |
| 4 | 9 | ✔ | ✔ |
| 5 | 9 | ✔ | ✔ |
| 6 | 9 | ✔ | ✔ |
| 7 | 9 | ✔ | ✔ |

# Demo: max of an array

- syntax for variables (`var`) and arrays (`array<...>`)
- preconditions (`requires`)
- termination measures (`decreases`)
- universal (`forall`) and existential (`exists`) quantification
- loop invariants (`invariant`)
- predicates (`predicate`)