Compiler Support for Learning Invariants

MICHAEL LAY
SUPERVISOR: MATT ROBERTS

May 25th, 2022

Schedule

MACQUARIE University

- Introduction (5 minutes)
- Pre-Workshop Test (20 minutes)
- Mixed Content (1.5 hours)
- Post Workshop Test (20 minutes)

With short 5-minute breaks in between

Setup

MACQUARIE University

- gitpod.io/#https://github.com/MicAu/Workshop
- or
- tinyurl.com/invariant10

Pre-Workshop Test

- No notes or looking things up!
  - Trying to find out what you **DON'T** know

- Your results don't affect your unit grades or chances at the prizes!

Learning Invariants

ENSURING CORRECTNESS

- Why should we care about program correctness?

```
# -- Calculate the year number y for today,
# -- where d is the number of days since 1/1/1980.
y= 1980
while d>365:
  if leapYear(y):
    if d>366:
      d= d-366
      y= y+1
  else:
    d= d-365
    y= y+1
# The current year is y.
```

Learning Invariants

VERIFICATION TECHNIQUES

- Wide variety of verification techniques
- Dynamic verification
  - **Unit testing**
  - Functional testing
  - Runtime assertions
- Static verification
  - Linting
  - **Formal verification**

Learning Invariants

UNIT TESTING

- Test individual units or components of a program
- Must develop representative test cases
- Must check edge cases
- **How do you know you have enough tests?**
- **How do you know you have the correct tests?**

Learning Invariants

UNIT TESTING

- Does the below unit test check sufficient cases?

```
public static boolean isOdd(int n) {
    return n % 2 == 1;
}

@Test
public void testIsOdd() {
    assertEquals(true, isOdd(5));
    assertEquals(false, isOdd(0));
    assertEquals(false, isOdd(-2));
    assertEquals(true, isOdd(7));
    assertEquals(true, isOdd(131));
    assertEquals(false, isOdd(-8));
}
```

Pre and Post Conditions

- Contract between a user and the programmer
- Pre-condition - Specification of what must be true at the start of the program

  (what the user must ensure)
- Post-condition – Specification of what must be true at the end of the program

  (what the programmer must ensure)

```
int sqrt(double n) {
    //Pre-condition: n >= 0

    //Post-condition: returns the square root of n
    return -1;
}
```

Hoare Logic

HOARE LOGIC

- How do we show that a program will reach the post condition?
- Hoare triples
  - {P} S {Q}
  - P and Q are predicates representing our pre and post conditions
  - S is the program
- If the program is in state P and we execute S (and S terminates), Q will be true.
  - {k = 3} k = 15 {k = 15}
  - {x = y} y = y + 3 {y = x + 3}

Hoare Logic

WEAKEST AND STRONGEST PRE/POST

- Are the following Hoare triples correct?
  - {x = 5} x = x * 2 {x = 10}
  - {x = 5} x = x * 2 {x > 0}
  - {x = 5} x = x * 2 {true}
  - {true} x = x * 2 {true}

- What about this?
  - {false} k = 3 {k = -4}

Invariants

LOOPS?

- How do we deal with the loop in the following code?
- Are Hoare triples enough?

```
def pow(x, N):
    # {N >= 0}

    result = 1
    i = 0

    while i < N:
        result = result * x
        i = i + 1

    # {result = x ^ n}
    return result
```

Invariants

- For code without loops, we are simulating execution directly
  - We prove one Hoare Triple for each statement, and each statement is executed once

- For code with loops, we are doing one proof of correctness for multiple loop iterations
  - Don’t know how many iterations there will be
  - Need our proof to cover all of them

Invariants

MACQUARIE University

- The invariant is a general condition that must be true for every execution of the loop, but still be strong enough to provide us the postcondition.

- An invariant is correct when it adheres to the following rules:
  - Is true before it starts
  - Is maintained on each loop iteration
  - Loop termination implies the post condition

Invariants

EXAMPLES

- Given x and n, calculate x^n

```python
def pow(x, N):
    # {N >= 0}

    result = 1
    i = 0

    while i < N:
        result = result * x
        i = i + 1

    # {result = x ^ n}
    return result
```

- What is the invariant for the program?

Invariants

EXAMPLES

- What is the loop invariant?
- Is it true before the first iteration?
- Is it true after each iteration?

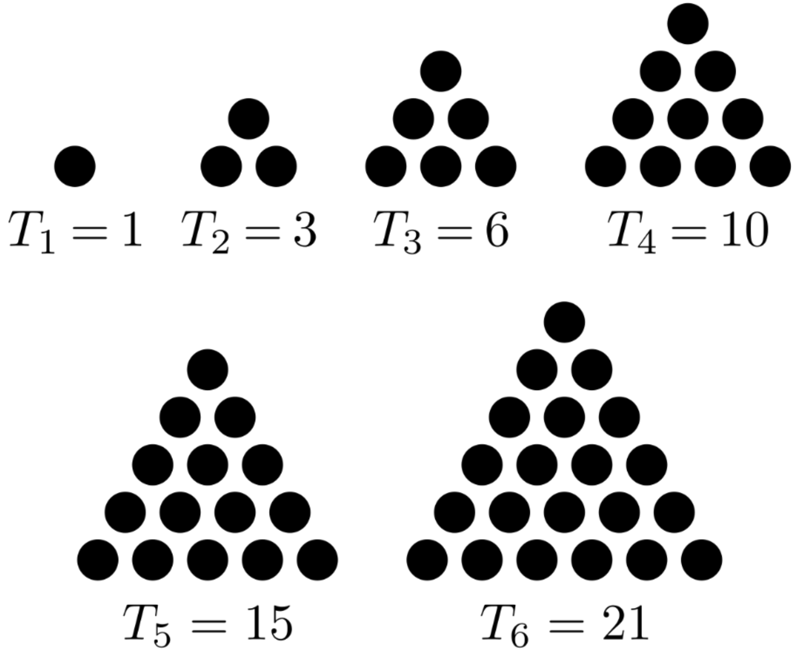
```
def pow(x, N):
    # {N >= 0}

    result = 1
    i = 0

    while i < N:
        result = result * x
        i = i + 1

    # {result = x ^ n}
    return result
```

Invariants

EXAMPLES

- Given n, calculate Tn

```
def triangle(N):
    # {N >= 0}
    n = 0
    t = 0
    while n < N:
        n = n + 1
        t = t + n
    # {t == N * (N + 1) / 2}
    return t
```

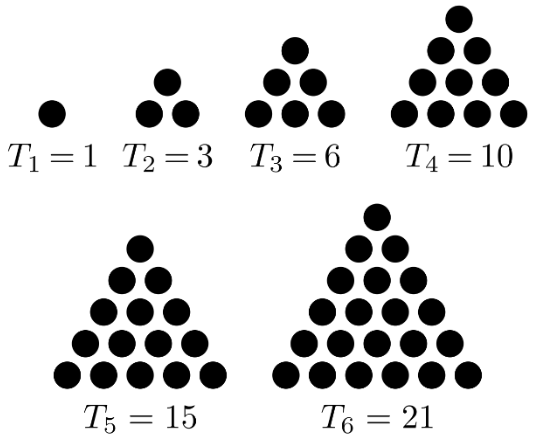


Invariants

EXAMPLES

- What is the loop invariant?
- Is it true before the first iteration?
- Is it true after each iteration?

```python
def triangle(N):
    # {N >= 0}
    n = 0
    t = 0
    while n < N:
        n = n + 1
        t = t + n
    # {t == N * (N + 1) / 2}
    return t
```



Invariants

EXAMPLES

- Given an array, return the sum of all numbers

```python
def sum_array(a):
    i = 0
    sum = 0
    while i < len(a):
        sum = sum + a[i]
        i = i + 1
    return sum
```

Invariants

EXAMPLES

- What is the loop invariant?

- Is it true before the first iteration?

- Is it true after each iteration?

```python
def sum_array(a):
    i = 0
    sum = 0
    while i < len(a):
        sum = sum + a[i]
        i = i + 1
    return sum
```

Invariants

DESIGNING WITH INVARIANTS

- Given an array, return the highest number in the array

```
def max(a):
    if len(a) == 0:
        return 0
    max = a[0]
    i = 1
    while i < a.Length:
        if a[i] > max:
            max = a[i]
        i = i + 1
    return max
```

Invariants

EXAMPLES

- What is the loop invariant?
- Is it true before the first iteration?
- Is it true after each iteration?

```
def max(a):
    if len(a) == 0:
        return 0
    max = a[0]
    i = 1
    while i < a.Length:
        if a[i] > max:
            max = a[i]
        i = i + 1
    return max
```

Invariants

EXAMPLES

- Given an array, sort it in ascending order

```python
def selectionSort(a):
    n = 0
    while n != len(a):
        mindex = n
        m = n + 1
        while m != len(a):
            if(a[m] < a[mindex]):
                mindex = m
            m = m + 1

        a[n] , a[mindex] = a[mindex], a[n]
        n = n + 1
```

Invariants

EXAMPLES

- What is the loop invariant?
- Is it true before the first iteration?
- Is it true after each iteration?

```python
def selectionSort(a):
    n = 0
    while n != len(a):
        mindex = n
        m = n + 1
        while m != len(a):
            if(a[m] < a[mindex]):
                mindex = m
            m = m + 1

        a[n] , a[mindex] = a[mindex], a[n]
        n = n + 1
```

Extra Slides

DESIGNING WITH INVARIANTS

Invariants

EXAMPLES

- Given two n bit binary numbers x and y, calculate x the sum of both numbers

Algorithm 1: Algorithm to sum two binary numbers

Data: $\boldsymbol{x} = x_n, x_{n-1}, \ldots, x_1$ and $\boldsymbol{y} = y_n, y_{n-1}, \ldots, y_1$: two n-bit binary numbers

Result: $\boldsymbol{z} = z_{n+1}, z_n, z_{n-1}, \ldots, z_1$: the sum of $\boldsymbol{x}$ and $\boldsymbol{y}$

$c \leftarrow 0$

for $i = 1, 2, 3, \ldots, n$ **do**

  $z_i \leftarrow (x_i + y_i + c) \mod 2$

  $c \leftarrow (x_i + y_i + c) \div 2$

end

$z_{n+1} \leftarrow c$

return $\boldsymbol{z} = z_{n+1}, z_n, \ldots, z_1$

Invariants

EXAMPLES

- What is the loop invariant?

- Is it true before the first iteration?

- Is it true after each iteration?

Algorithm 1: Algorithm to sum two binary numbers

Data: $\boldsymbol{x} = x_n, x_{n-1}, \ldots, x_1$ and $\boldsymbol{y} = y_n, y_{n-1}, \ldots, y_1$: two n-bit binary numbers

Result: $\boldsymbol{z} = z_{n+1}, z_n, z_{n-1}, \ldots, z_1$: the sum of $\boldsymbol{x}$ and $\boldsymbol{y}$

$c \leftarrow 0$

for $i = 1, 2, 3, \ldots, n$ **do**

$\quad z_i \leftarrow (x_i + y_i + c) \mod 2$

$\quad c \leftarrow (x_i + y_i + c) \div 2$

end

$z_{n+1} \leftarrow c$

return $\boldsymbol{z} = z_{n+1}, z_n, \ldots, z_1$

Invariants

EXAMPLES

- Given an array, sort it in ascending order

Algorithm 3: Insertion Sort

Data: a: an array of n real numbers

Result: The non-decreasingly ordered permutation of a

```
for j = 2, 3, ..., n do
    x ← a[j]
    i ← j − 1
    while i > 0 and a[j] > x do
        a[i + 1] ← a[i]
        i ← i − 1
    end
    a[i + 1] ← x
end
return a
```

Invariants

EXAMPLES

- What is the loop invariant?

- Is it true before the first iteration?

- Is it true after each iteration?

Algorithm 3: Insertion Sort

Data: a: an array of n real numbers
Result: The non-decreasingly ordered permutation of a

```
for j = 2, 3, ..., n do
    x ← a[j]
    i ← j − 1
    while i > 0 and a[j] > x do
        a[i + 1] ← a[i]
        i ← i − 1
    end
    a[i + 1] ← x
end
return a
```