



Assertion-based Loop Invariant Generation or, Counter-example Refinement Backwards

Mikoláš Janota

System Research Group,
University College Dublin, Ireland



IST-15905



- ▶ we are looking for loop invariants that will show that something “bad” will not happen

```
for (int i = 0; i < a.length - 1; i++) {  
    i = i + 1;  
    a[i] = 0;  
}
```



- ▶ we are looking for loop invariants that will show that something “bad” will not happen

```
//@ loop_invariant a != null;  
//@ loop_invariant i + 1 >= 0;  
  
for (int i = 0; i < a.length - 1; i++) {  
    i = i + 1;  
    a[i] = 0;  
}
```



Scary Slide

```
//@ requires a != null;
/*@ requires
  @ (\forall int x; (0 <= x & x < a.length) ==> a[x] != null); */
void setToZero(int [][] a) {
  /*@ loop_invariant
    @ (\forall int x; (0 <= x & x < a.length) ==> a[x] != null);
    @ loop_invariant a != null;
    @ loop_invariant i >= 0; */
  for (int i = 0; i < a.length; i++) {
    /*@ loop_invariant j >= 0;
      @ loop_invariant a != null;
      @ loop_invariant
        @ (\forall int x; (0 <= x & x < a.length) ==> a[x] != null); */
    for (int j = 0; j < a[i].length; j++)
      a[i][j] = 0;
  }
}
```



Invariants from Assertions Using Weakest Precondition

- ▶ let's assume that desired behavior is expressed as assertions
- ▶ using a weakest precondition calculus we can back-propagate assertions to the head of the loop

```
for (int i = 0; i < a.length - 1; i++) {  
    i = i + 1;  
    //@ assert a != null;  
    //@ assert i <= 0;  
    //@ assert i < a.length;  
    a[i] = 0;  
}
```



Invariants from Assertions Using Weakest Precondition

- ▶ let's assume that desired behavior is expressed as assertions
- ▶ using a weakest precondition calculus we can back-propagate assertions to the head of the loop

```
//@ loop_invariant (i < a.length - 1) ==> a != null;
```

```
for (int i = 0; i < a.length - 1; i++) {  
    i = i + 1;  
    //@ assert a != null;  
    //@ assert i <= 0;  
    //@ assert i < a.length;  
    a[i] = 0;  
}
```



Invariants from Assertions Using Weakest Precondition

- ▶ let's assume that desired behavior is expressed as assertions
- ▶ using a weakest precondition calculus we can back-propagate assertions to the head of the loop

```
//@ loop_invariant (i < a.length - 1) ==> a != null;  
//@ loop_invariant (i < a.length - 1) ==> 0 <= i + 1;
```

```
for (int i = 0; i < a.length - 1; i++) {  
    i = i + 1;  
    //@ assert a != null;  
    //@ assert i <= 0;  
    //@ assert i < a.length;  
    a[i] = 0;  
}
```



Invariants from Assertions Using Weakest Precondition

- ▶ let's assume that desired behavior is expressed as assertions
- ▶ using a weakest precondition calculus we can back-propagate assertions to the head of the loop

```
//@ loop_invariant (i < a.length - 1) ==> a != null;  
//@ loop_invariant (i < a.length - 1) ==> 0 <= i + 1;  
//@ loop_invariant (i < a.length - 1) ==> i + 1 < a.length;  
for (int i = 0; i < a.length - 1; i++) {  
    i = i + 1;  
    //@ assert a != null;  
    //@ assert i <= 0;  
    //@ assert i < a.length;  
    a[i] = 0;  
}
```



Assertion Back-propagation

- ▶ we take a *nest* of loops

```
 $P_0;$   
 $\{\mathcal{I}_1\}$  while ( $C_1$ ) do  
  ...  
   $P_{n-1};$   
   $\{\mathcal{I}_n\}$  while ( $C_n$ ) do  
     $P_n;$   
    assert  $e;$ 
```

- ▶ and propagate the invariant outwards

$$\mathcal{I}_n \equiv wlp(P_n, e)$$
$$\mathcal{I}_i \equiv wlp(P_i, \mathcal{I}_{i+1})$$



Are the Invariants OK?

- ▶ all loops must *preserve* the pertaining invariant

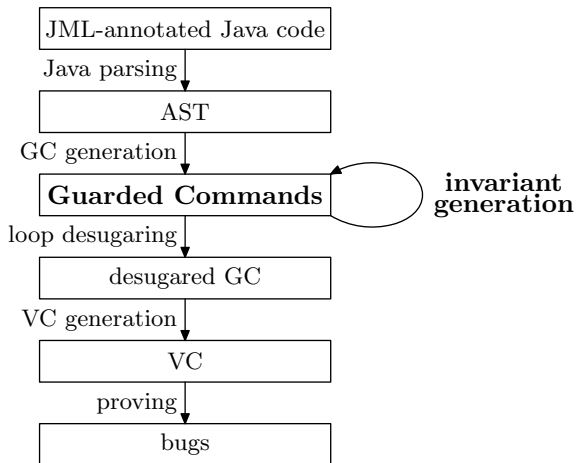
$$\models \mathcal{I}_i \Rightarrow wlp(\text{LoopBody}_i, \mathcal{I}_i)$$

- ▶ the invariant of the outermost loop must be established by the preceding command

$$\models wlp(P_0, \mathcal{I}_1)$$



Implementation in ESC/Java2



- ▶ Guarded Commands:

$$cmd := x \leftarrow expr \mid \mathbf{assume} f \mid \mathbf{assert} f \mid \\ cmd \parallel cmd \mid cmd; cmd \mid \{\mathcal{J}\}\mathbf{while} expr \mathbf{do} cmd$$

\mathcal{J}, f are first-order logic formulas

- ▶ captures JML-annotated Java, examples:
 - ▶ **assert** pre — a precondition of a called method,
 - ▶ desired behavior, such as **assert** $a \neq \text{null}$
 - ▶ **assume** post — a postcondition of a called method



Tweaking the Algorithm

- ▶ *invariant strengthening*, heuristically altering the invariant when found that it does not preserve the pertaining loop, e.g.,

$\mathcal{J}[v \mapsto v']$, v is free in \mathcal{J} and v' is a fresh variable

- ▶ applying *formula simplifications* to the inferred invariants
- ▶ when computing weakest precondition, ignoring commands that do not seem *relevant* (relying on a heuristic)



Experiences and Future Work

- ▶ works splendidly on the examples I wrote, **nevertheless**,
- ▶ little of existing code is verifiable
- ▶ generated invariants are not too big, **nevertheless**,
- ▶ there is still opportunity to prune away trivial invariants
- ▶ reporting invariants to the user would serve as valuable feedback
- ▶ extending the analysis to take into account assertions **after** the loop would make the analysis considerably more useful

