Reasoning about Programs

Jeremy Bradley, Francesca Toni and Xiang Feng

Room 372. Office hour - Tuesdays at noon. Email: jb@doc.ic.ac.uk

Department of Computing, Imperial College London

Produced with prosper and LaTeX

Loops

- In imperative languages:
  - common loop blocks include: `while`, `for`, `repeat/until`
  - all can be expressed as `while` loops
- A while loop:

```
i = 2;
while ( i > 0 ) {
    somemethod(P);
    --i;
}
```


Anatomy of a loop

```
i = 2;  // setup code
while ( i > 0 ) { // loop condition
    somemethod(P);
    --i; // counter dec/increment
}
```

- A loop typically consist of:
  - setup code
  - a looping condition which must be true for the loop to execute
  - optionally a counter operation

Loops invariants and Loop variants

- **loop invariant**
  - is a mid-condition embedded in a loop
- **loop variant**
  - is a modeller-supplied quantity that decreases at each iteration of the loop
  - it never becomes negative
  - loop variant $= 0$ should coincide with termination of loop

Reasoning about Loops

- A loop invariant is specified before a loop block and usually expresses the cumulative affect of the loop on the method variables after i completed iterations
- A loop variant is often (but doesn't have to be) expressed in terms of the loop variable
  - for a `for` loop were i varies from $1 \rightarrow n$, a loop variant would be $n - i$
  - existence of a loop variant ensures loop termination

Adding up an array

```
static int sumArray (int a[]) {
    int i;
    int res = 0;
    // Loop invariant: 0 <= i < a.length
    //   && res = \sum_{j=0}^{i-1} a[j]
    for (i = 0; i < a.length; ++i) {
        // Loop variant: a.length - i
        res = res + a[i];
    }
    return res;
}
```

- Invariant is a good place to check array bounds will not be violated

General Reasoning about Loops

1. Is loop invariant established at beginning of loop?
2. Is invariant re-established?
   - i.e. does invariant on kth iteration $\rightarrow$ invariant on $k + 1$th iteration
3. Does loop terminate?
   - i.e. does loop variant decrease on each iteration and does it have a minimum value
4. Finally, does loop termination and invariant $\rightarrow$ post-condition?

Transform into while loop...

```
static int sumArray (int a[]) {
    // Pre: none
    // Post: res = \sum_{j=0}^{a.length-1} a[j]
[1] int i=0;
[2] int res = 0;
[L] while ( i < a.length ) {
    // Loop invariant: 0 <= i_k < a.length
    //   && res_k = \sum_{j=0}^{i_k - 1} a[j]
        res = res + a[i];
        ++i;
    }
    return res;
}
```

Invariant: base case

- Need to show invariant true the first time that it is executed
- This is a standard mid-condition argument
  - Pre-condition $\vdash$ first loop invariant
  - i.e. in this case:

$$\vdash \left(0 \leq i_0 < a.length \wedge res_0 = \sum_{j=0}^{i_0-1} a[j]\right)$$

Invariant: base case

1.	$i_0 = 0$	code[1]$\mathcal{I}$
2.	$res_0 = 0$	code[2]$\mathcal{I}$
3.	$i_0 < a.length$	code[L]$\mathcal{I}$
4.	$i_0 = 0 \vee i_0 > 0$	$\vee\mathcal{I}(1)$
5.	$0 \leq i_0$	$\leq$def(4)
6.	$0 \leq i_0 < a.length$	$\wedge\mathcal{I}(3,5)$
7.	$res_0 = \sum_{j=0}^{i_0-1} a[j]$	$\sum$def(2)
8.	$0 \leq i_0 < a.length \wedge res_0 = \sum_{j=0}^{i_0-1} a[j]$	$\wedge\mathcal{I}(6,7)$

Transform into while loop...

```
static int sumArray (int a[]) {
    // Pre: none
    // Post: res = \sum_{j=0}^{a.length-1} a[j]
    int i=0;
    int res = 0;
[L] while ( i < a.length ) {
    // Loop invariant: 0 <= i_k < a.length
    //   && res_k = \sum_{j=0}^{i_k - 1} a[j]
[1]     res = res + a[i];
[2]     ++i;
    }
    return res;
}
```


`sumArray`: Re-establishing invariant

- Trying to show that:

$$0 \leq i_k < a.length \wedge res_k = \sum_{j=0}^{i_k-1} a[j]$$

$$\vdash \quad 0 \leq i_{k+1} < a.length \wedge res_{k+1} = \sum_{j=0}^{i_{k+1}-1} a[j]$$

- where var_k in a loop invariant context means the value of the variable after the kth loop iteration
- To show this: need to take into account both code and loop condition

`sumArray`: Re-establishing invariant

1.	$0 \le i_k < a.length \wedge res_k = \sum_{j=0}^{i_k-1} a[j]$	giv
2.	$0 \le i_k < a.length$	$\wedge\mathcal{E}(1)$
3.	$res_k = \sum_{j=0}^{i_k-1} a[j]$	$\wedge\mathcal{E}(1)$
4.	$res_{k+1} = res_k + a[i_k]$	code[1]$\mathcal{I}$
5.	$i_{k+1} = i_k + 1$	code[2]$\mathcal{I}$
6.	$i_{k+1} < a.length$	code[L]$\mathcal{I}$
7.	$1 \le i_{k+1}$	$=$subs$(5,2)$
8.	$0 \le i_{k+1}$	$\le$trans(7)
9.	$0 \le i_{k+1} < a.length$	$\wedge\mathcal{I}(8,6)$
10.	$res_{k+1} = \sum_{j=0}^{i_k-1} a[j] + a[i_k]$	$=$subs$(3,4)$
11.	$res_{k+1} = \sum_{j=0}^{i_k} a[j]$	$\sum$def(10)
12.	$res_{k+1} = \sum_{j=0}^{i_{k+1}-1} a[j]$	$=$subs$(5,11)$
13.	$0 \le i_{k+1} < a.length \wedge res_{k+1} = \sum_{j=0}^{i_{k+1}-1} a[j]$	$\wedge\mathcal{I}(9,12)$

Re-establishing invariant

1. Prove loop invariant holds on entry to loop (i.e. base case)
2. Assume loop invariant holds on kth iteration:

$$\text{invariant}(k) \wedge \text{code}(k) \wedge \text{loop condition} \\ \rightarrow \text{invariant}(k+1)$$

3. (c.f. induction step $P(k) \rightarrow P(k+1)$)

Find an element in Array

```
int find (int a[], int x) {
    int res,i;
    for (i=0; a[i] != x && i < a.length; ++i) {}
    res=i;
    return res;
}
```

- What post-condition do we need:
  - $a[res] = x$?
  - $0 \le res < a.length \wedge a[res] = x$?
- ...and if we want to say it finds the first matching element in the array?

Find an element in Array

```
  int find (int a[], int x) {
     // Pre: there exists j. 0 <= j < a.length
     //         && a[j] = x
     // Post: <next slide>
[1]  int i=0;
[2]  int res=0;
[L]  while ((i < a.length) && (a[i] != x)) {
         // Invariant: <next slides>
[3]      ++i;
     }
[4]  res = i;
     return res;
  }
```

- Pre-condition: $\exists j. 0 \le j < a.length \wedge a[j] = x$

Post-condition for `find`

- Post-condition:
  - $a = a_0$:keep array unchanged
  - $\wedge\ 0 \leq res < a.length$:keep within array bounds
  - $\wedge\ a[res] = x$:res is correct index
  - $\wedge\ (0 \leq j < res) \rightarrow a[j] \neq x$
    :no elements before res matched
- So how can we use this to design an invariant?

Invariant design

- Use post-condition to help generate invariant
- Take into account any lines of code that are executed between invariant and post-condition
- For `find` use loop invariant:
  - $a = a_0$:keep array unchanged
  - $\wedge\ 0 \leq i_k < a.length$:keep within array bounds
  - $\wedge\ (0 \leq j \leq i_k) \rightarrow a[j] \neq x$
    :no elements before i_k matched

`find`: Invariant base case

- Need to establish invariant with: pre-condition $\vdash$ first loop invariant

1.	$\exists j. 0 \leq j < a.length \wedge a[j] = x$	giv
2.	$a = a_0$	var$\mathcal{I}$
3.	$i_0 = 0$	code[1]$\mathcal{I}$
4.	$i_0 = 0 \vee i_0 > 0$	$\vee\mathcal{I}(3)$
5.	$i_0 \geq 0$	$\geq$def(4)
6.	$res_0 = 0$	code[2]$\mathcal{I}$
7.	$a[i_0] \neq x \wedge i_0 < a.length$	code[L]$\mathcal{I}$
8.	$i_0 < a.length$	$\wedge\mathcal{E}(5)$
9.	$0 \leq i_0 < a.length$	$\wedge\mathcal{I}(5,8)$

`find`: Invariant base case

10.	$a[i_0] \neq x$	$\wedge\mathcal{E}(5)$
11.	$0 \leq j \leq i_0$	ass
12.	$0 \leq j \leq 0$	=subs$(3,11)$
13.	$j = 0$	=def(12)
14.	$a[0] \neq x$	=subs$(3,10)$
15.	$a[j] \neq x$	=subs$(13,14)$
16.	$(0 \leq j \leq i_0) \rightarrow a[j] \neq x$	$\rightarrow\mathcal{I}(11,15)$
17.	$a = a_0 \wedge 0 \leq i_0 < a.length$ $\wedge\ (0 \leq j \leq i_0) \rightarrow a[j] \neq x$	$\wedge\mathcal{I}(2,9,16)$

`find`: Re-establishing invariant

- To re-establish invariant for `find`, we need to show that kth iteration invariant $\vdash (k+1)$th iteration invariant
  - $$\begin{aligned} & a = a_0 \\ \wedge\ & 0 \leq i_k < a.length \\ \wedge\ & (0 \leq j \leq i_k) \rightarrow a[j] \neq x \end{aligned}$$
    $$\begin{aligned} \vdash\ & a = a_0 \\ \wedge\ & 0 \leq i_{k+1} < a.length \\ \wedge\ & (0 \leq j \leq i_{k+1}) \rightarrow a[j] \neq x \end{aligned}$$

`find`: Re-establishing invariant

1.	$a = a_0 \wedge 0 \leq i_k < a.length$ $\wedge (0 \leq j \leq i_k) \rightarrow a[j] \neq x$	giv
2.	$a = a_0$	$\wedge\mathcal{E}(1)$
3.	$0 \leq i_k < a.length$	$\wedge\mathcal{E}(1)$
4.	$0 \leq i_k$	$\wedge\mathcal{E}(3)$
5.	$(0 \leq j \leq i_k) \rightarrow a[j] \neq x$	$\wedge\mathcal{E}(1)$
6.	$i_{k+1} = i_k + 1$	code[3]$\mathcal{I}$
7.	$a[i_{k+1}] \neq x \wedge i_{k+1} < a.length$	code[L]$\mathcal{I}$
8.	$a[i_{k+1}] \neq x$	$\wedge\mathcal{E}(6)$
9.	$i_{k+1} < a.length$	$\wedge\mathcal{E}(6)$
10.	$1 \leq i_{k+1}$	$=$subs$(6,4)$
11.	$0 \leq i_{k+1}$	$\leq$trans(10)
12.	$0 \leq i_{k+1} < a.length$	$\wedge\mathcal{I}(11,9)$

`find`: Re-establishing invariant

13.	$0 \leq j \leq i_{k+1}$	ass
14.	$0 \leq j \leq i_k$	ass
15.	$a[j] \neq x$	$\rightarrow\mathcal{E}(5)$
16.	$j = i_{k+1}$	ass
17.	$a[j] \neq x$	$=$subs$(8,16)$
18.	$a[j] \neq x$	$\vee\mathcal{E}(13,14,15,16,17)$
19.	$(0 \leq j \leq i_{k+1}) \rightarrow a[j] \neq x$	$\rightarrow\mathcal{E}(13,18)$
20.	$a = a_0 \wedge 0 \leq i_{k+1} < a.length$ $\wedge (0 \leq j \leq i_{k+1}) \rightarrow a[j] \neq x$	$\wedge\mathcal{I}(2,12,19)$