So You Want To Do Formal Verification

Basic Idea:

Use formal verification to prove that given a method's preconditions, the method satisfies its postconditions after executing its code (i.e. pre + code → post)

Steps Involved:

1. Add assertions in important places in code (e.g. **inv** assertion in loops, **term** at end, etc.)
2. Label code fragments between major assertions (e.g. **init**, **body**, **final**, etc.)
3. Outline all possible transitions from assertion to assertion through fragments of code, starting from the preconditions and stopping at the postconditions (e.g. **pre** + **init** → **inv**, $\mathbf{inv}_{\mathbf{before}}$ + **Enter** + **body** → $\mathbf{inv}_{\mathbf{after}}$, etc.)
   Also include a bounding function for all loops, to prove that the loops actually reach a stopping condition (integer, non-negative, decreasing -- see slide 89)
4. For each transition, use the assertions from the left side of the transition to prove that the assertion on the right side of the transition is correct in all cases.

Helpful Hints:

- When you are trying to figure out what assertions to write (or how the notes arrived at their assertions), take your cues from the pre- and postconditions. For example, your postconditions will likely some clue as to what your **term** assertions should be, and your **term** assertions should give some clue as to what conditions your **inv** assertions should maintain. Likewise, your preconditions will indicate what kind of initial values your **inv** assertion might have.
- If there's a statement you want to make in your proof but you're not sure you're allowed to, prove it on the side as a lemma, and refer to it when necessary (like a helper method).
- If the method you're verifying calls another method, assume that the called method works according to its pre- and postconditions, unless explicitly stated otherwise.
- Don't worry if it doesn't work out on the first try, because it is expected that you will need to revise your proof occasionally in order to make it work.

Ready? Let's rock 'n' roll….

Verification Example #1: product

Want to prove: pre + code → post

Must prove:
1. pre + init → inv
2. inv_{before} + Enter + body → inv_{after}
3. inv + Exit → term
4. term + final → post
5. Bounding Function

Proofs:

1. From pre: multiplier ≥ 0
   From init: I=0, J=0
   Show inv: - Since multiplier ≥ 0 and I=0, **multiplier ≥ I ≥ 0** after init
   - Since I=0 and J=0, **J=0=I*this.intValue()**
   - Therefore, pre + init → inv

2. From inv: $0 \le I_{before} \le$ multiplier, $J_{before} = I_{before}$*this.intValue()
   From Enter: I_{before} != multiplier
   From body: $J_{after} = J_{before}$ + this.intValue(), $I_{after} = I_{before} + 1$
   Show inv: - Since $0 \le I_{before} \le$ multiplier and I_{before} != multiplier, $0 \le I_{before} <$ multiplier.
   Also, since $I_{after} = I_{before} + 1$, we can conclude that **$0 \le I_{after} \le$ multiplier**
   - Since $J_{after} = J_{before}$ + this.intValue() and $J_{before} = I_{before}$*this.intValue(),
   $J_{after} = I_{before}$*this.intValue() + this.intValue()
   $J_{after} = (I_{before} + 1)$*this.intValue()
   $J_{after} = I_{after}$*this.intValue()
   - Therefore, inv_{before} + Enter + body → inv_{after}

3. From inv: $0 \le I \le$ multiplier, J=I*this.intValue()
   From Exit: I=multiplier
   Show term: - Since **I=multiplier** from Exit, and **J=I*this.intValue()**,
   Therefore, inv + exit → term

4. From term: I=multiplier, J=I*this.intValue()
   Show post: - Substituting I into J, we get **J=multiplier*this.intValue()**
   - From final, multiplier*this.intValue() is returned
   - Therefore, post is satisfied

5. Bounding Function (prove that while loop exits)
   - Let f =multiplier-I (integer operation)
   - Since multiplier ≥ 0 and I=0 from init, then multiplier ≥ I and f ≥ 0 (non-negative)
   - Since I increments on every iteration, f is steadily decreasing from multiplier to zero → stopping condition
   - Therefore, since there is a non-negative integer function f that decreases to a stopping condition at 0, the while loop will terminate

Therefore, pre + code → post

Formal Verification Example #2: locate

Want to prove: pre + code → post

Must prove:
1. pre + init → inv
2. inv_{before} + Enter + body → inv_{after}
3. inv + Exit → term
4. term + final → post
5. Bounding Function

Proofs:

First, a lemma:	For two integers x and y such that $x < y$,
	$x \le (x+y)/2 < y$, when "/" implies integer division.
Proof of lemma:	If $x < y$ then $y \ge x+1$, and $(x+y)/2 \ge (x+x+1)/2$
	But since $(2x+1)/2 = x$, $(x+y)/2 \ge x$
	If $x < y$ then $x \le y-1$, and $(x+y)/2 \le (y-1+y)/2$
	But since $(2y-1)/2 < y$, $(x+y)/2 < y$

1. From pre: this is an ascending sequence
   From init: lo=0, hi=size(), mid=(lo + hi) / 2
   Show inv: - Since this is not changed by init, **this is still an ascending sequence** at inv
   - **mid is set to (lo + hi) / 2** in the for loop initialization
   - Since hi=size() ≥ 0=lo, this implies that **0 ≤ lo ≤ hi ≤ size()**
   - **this[0…lo-1]<target** and **this[hi…size()-1]≥target** are both *vacuously* true, since lo=0 and hi=size()
   - Therefore, pre + init → inv

2. From inv: this is an ascending sequence, $mid_{before}=(lo + hi) / 2$,
   $0 \le lo_{before} \le hi_{before} \le size()$, this[0…$lo_{before}$-1]<target,
   this[hi_{before}…size()]≥target
   From Enter: $lo_{before} < hi_{before}$
   From body: two cases: i) if statement is true (this[mid_{before}]<target, $lo_{after} = mid_{before}+1$)
   ii) if statement is not true (this[mid_{before}]≥target, $hi_{after} = mid_{before}$)
   Show inv (i): - Since this has not been changed, **this is still an ascending sequence**.
   - The last portion of for statement sets **$mid_{after}=(lo_{after}+hi_{after})/2$**
   - Since this[mid_{before}]<target and $lo_{after} = mid_{before}+1$, this[$lo_{after}$-1] <target
   Also, since this is in ascending order, then **this[0…lo_{after}-1]<target**
   - Since $hi_{after}=hi_{before}$, **this[hi_{after}…size()]≥target** is unchanged.
   - Since $mid_{before}=(lo_{before}+hi_{before})/2$ and $lo_{before}<hi_{before}$ from Enter,
   $lo_{after}=mid_{before}+1 \le hi_{before}$ from lemma.
   Since $hi_{before}=hi_{after}$, $lo_{after}>mid_{before} \ge 0$ and $hi_{after}=hi_{before} \le size()$,
   we conclude that **$0 \le lo_{after} \le hi_{after} \le size()$**
   - Therefore, inv_{before} + Enter + body → inv_{after}

Show inv (i): - Since this has not been changed, **this is still an ascending sequence**.
- The last portion of for statement sets $\mathbf{mid_{after}=(lo_{after}+hi_{after})/2}$
- Since this[mid_{before}]≥target and $hi_{after} = mid_{before}$, this[$hi_{after}$]≥target
Also, since this is in ascending order, then **this[hi_{after}…size()]≥target**
- Since $lo_{after}=lo_{before}$, **this[0…lo_{after}]<target** is unchanged.
- Since $mid_{before}=(lo_{before}+hi_{before})/2$ and $lo_{before}<hi_{before}$ from Enter,
$lo_{after}=lo_{before}\leq mid_{before}=hi_{after}$ from lemma.
Since $hi_{after}<hi_{before}\leq$size() and $lo_{after}=lo_{before}\geq 0$,
we conclude that $\mathbf{0 \leq lo_{after} \leq hi_{after} \leq size()}$
- Therefore, inv_{before} + Enter + body → inv_{after}

3. From inv: $0 \leq lo \leq hi \leq$ size(), this[0…lo]<target, this[hi…size()]≥target
From Exit: $hi \leq lo$
Show term: - Since $lo \leq hi$ and $hi \leq lo$, hi=lo. Substituting into inv properties, we get
0 ≤ lo ≤ size(), **this[0…lo]<target**, and **this[lo…size()]≥target**
Therefore, inv + exit → term

4. From term: $0 \leq lo \leq$ size(), this[0…lo]<target, and this[lo…size()]≥target
Show post: - From final, specified index is returned
- Therefore, post is satisfied

5. Bounding Function (prove that while loop exits)
- Let f =hi-lo (integer operation)
- Since hi=size() and lo=0 after init, then f ≥ 0 at start of body (non-negative)
- On every iteration of loop, either
i) $hi_{after}-lo_{after} = hi_{before}-(mid_{before}+1) < hi_{before} - lo_{before}$
ii) $hi_{after}-lo_{after} = mid_{before}-lo_{after} < hi_{before}-lo_{before}$
- Therefore, since there is a non-negative integer function f that decreases to a stopping condition at 0, the while loop will terminate

Therefore, pre + code → post