Verifying the logic of our code, part 2

A Reminder of What we are Doing:

1) Decide on the desired goal (also called the postcondition) for the method.

2) Decide on a reasonable precondition for the method.

3) The key part: verify that the given precondition is really all that must be true before the method is run for it to achieve the desired goal.

- if the precondition is not sufficient, then there is a logic flaw in your code.

We will use the isEnglishPalindrome method as our example.

Goal: The method returns true if (and only if) the English letters of s read the same forwards as backwards, ignoring the case.

Desired pre-condition: The input String s is not null.

Weakest precondition argument:

Given a goal, we will calculate the minimum that must be true before executing the code such that the goal is true after we execute the code.

We will then compare this "weakest precondition" with the desired precondition we listed above. If they match (or if the desired precondition is stronger than the weakest precondition), then the method is logically correct.

Loop Subgoal/Loop Invariant

1) We first get a goal for the loop.

2) Then we figure a subgoal (called the loop invariant) that must be achieved after each loop interation. The subgoal should have the following properties:

2a) When the loop condition becomes false, the subgoal logically implies the goal of the loop.

2b) The subgoal is true after each loop iteration.

2c) The subgoal is true before starting the first loop iteration.

The subgoal is also called the "loop invariant".

3) Show that the loop will eventually halt.

Here is the code:

public static boolean isEnglishPalindrome(String s) {

int front = 0, back = s.length() - 1;

// after each iteration, the letters that appear before front read the

// same as the letters after back, but in reverse order

while (front < back) {

if (!isLetter(s.charAt(front)) // front is not a letter

front = front + 1; // so skip it

}

else if (!isLetter(s.charAt(back)) // back is not a letter

back = back - 1; // so skip it

}

else if (Character.toLowerCase(s.charAt(front)) != // front and back not are the same letter

Character.toLowerCase(s.charAt(back))) {

return false;

}

else { // otherwise front and back are the same letter

front = front + 1; // so keep going

back = back - 1;

}

}

return true;

}

Recall the goal of the loop that we found last lecture:

The goal is "The English letters of s read the same forwards as backwards, ignoring the case."

We then figured out the subgoal for each iteration. We verified that the subgoal AND (front >= back) logically implies the goal of the loop.

The subgoal is "After each iteration, the English letters in s from index 0 up to index front-1 read the same, ignoring case, as the English letters in s from index length-1 down to back + 1"

Step (2b): We must verify that the loop subgoal is true after each loop iteration.

We will use the weakest precondtion. Why? Assume the loop subgoal is the goal for the back loop iteration. The weakest preconditon for the back iteration becomes the goal of the second to back iteration.

The weakest precondition of the second to back iteration becomes the goal of the third to back iteration, and so on.

If the subgoal is true after each iteration, then the subgoal should also be the weakest precondition of the loop body.

The loop body is a giant if statement, so we need to find the weakest precondition of each branch, and then combine that with the if conditions to get the weakest precondition of the if statement.

Since the loop subgoal is the goal of the loop body, it will be the goal for each branch of the if statement.

Loop subgoal: "After each iteration, the English letters in s from index 0 up to index front-1 read the same, ignoring case, as the English letters in s from index length-1 down to back + 1"

Branch 1: front = front + 1;

Weakest precondtion: "The English letters in s from index 0 up to index front read the same, ignoring case, as the English letters in s from index length-1 down to back+1"

Branch 2: back = back - 1;

Weakest precondtion: "The English letters in s from index 0 up to index front-1 read the same, ignoring case, as the English letters in s from index length-1 down to back"

Branch 3: return false;

Here the weakest precondition must relate to the method goal because we are returning.

Weakest preconditon: "The English letters of s do NOT read the same forwards as backwards."

Branch 4: front = front + 1;

back = back - 1;

Weakest precondtion: "The English letters in s from index 0 up to index front read the same, ignoring case, as the English letters in s from index length-1 down to back"

Now, let us combine each with the if conditions, and hopefully each branch will have the loop subgoal as a valid precondition.

Branch 1 with condition:

Combining condition of this branch with weakest precondition for it gives:

"If s(front) is not a letter, the English letters in s from index 0 up to index front read the same, ignoring case, as the English letters in s from index length-1 down to back+1"

Since s(front) is not a letter, we only need to match the letters of s(0) up to s(front-1) with the end of the string. So, requiring the above statement is equivalent to requiring

"The English letters in s from index 0 up to index front-1 read the same, ignoring case, as the English letters in s from index length-1 down to back+1"

Branch 2 with condition:

Combining condition of this branch with weakest precondition for it gives:

"If s(front) is a letter but s(back) is not a letter, the English letters in s from index 0 up to index front-1 read the same, ignoring case, as the English letters in s from index length-1 down to back"

Since s(back) is not a letter, we only need to match the letters of s(length-1) down to s(back+1) with the front of the string. So, requiring the above statement is equivalent to requiring

"The English letters in s from index 0 up to index front-1 read the same, ignoring case, as the English letters in s from index length-1 down to back+1"

Branch 3 with condition:

Combining condition of this branch with weakest precondition for it gives:

"If s(front) and s(back) are English letters that do not match, the English letters of s do NOT read the same forwards as backwards."

For this statement to be true, we need to make sure that we have the same number of letters before front as we do after back (do you see why?).

So the weakest preconditon becomes "There are the same number of English letters in s from index 0 up to index front-1 as there are from index back+1 up to index length-1."

Branch 4 with condition:

Combining condition of this branch with weakest precondition for it gives:

"If s(front) matches s(back), the English letters in s from index 0 up to index front read the same, ignoring case, as the English letters in s from index length-1 down to back"

Since we know s(front) == s(back), ignoring case, we can ignore those two indeces in the precondition. So, requiring the above statement is equivalent to requiring

"The English letters in s from index 0 up to index front-1 read the same, ignoring case, as the English letters in s from index length-1 down to back+1"

Three of the four branches have the same weakest precondition, but a fourth does not. However, the fourth is a weaker precondition than our loop subgoal. So, if the subgoal, a stronger condition is true before the

loop iteration, all four branches will produce their desired goals. We would only have problems if one of the branches had a stronger necessary precondition than our loop subgoal.

To summarize step (2b). We showed, by working backwards, that if the loop subgoal is the precondition for an iteration of the loop body, then the loop subgoal will also be the goal for that iteration of the loop body.

This shows that the subgoal is true after each loop iteration.

Step (2c): Show the subgoal is true before entering the loop. This subgoal will become the weakest precondition for the loop.

Before the loop, front = 0 and back = length - 1.

The subgoal (i.e. the weakest precondtion for the loop) becomes "The English letters in s from index 0 up to -1 read the same, ignoring case, as the English letters in s from index length-1 down to length."

Note that there ARE NO letters from index 0 "up to" -1 or index length-1 "down to" length. So the above statement is ALWAYS true.

This is great! It means there are NO preconditions for our loop except that the string exists (i.e. is not null)!

Since there are no other statements for our method, the method preconditon that s is not null is the ONLY precondition for our method to work correctly.

Assuming we have no typos, our method will be correct. We can now use the testing technique given in the previous lecture to try all the paths through the code to make sure there are no typos.