

Weakest Preconditions for Loops

CptS 355 October 12, 2009

Last time we looked at the weakest precondition rules for assignment statements, statement sequences and for the if statement. There is an example of the application of the if statement rule in the semantics notes linked from the calendar on October 7, which we discussed in class today. The notes linked from the calendar on October 9 are the in-class handwritten notes that I did.

Today we turn to weakest preconditions for loops. Unlike the previously discussed rules which are completely mechanical, the rule for loops require development of insight about the way a loop works and its intended purpose. This insight is ultimately captured in a *loop invariant*.

Invariants

In general, an invariant is a property that is always true of the state of program or data structure. We discussed today the invariant for the *heap* datastructure: that the value stored at a parent node in the tree is always less than the values stored at its direct children. If this property is true of all nodes then one can conclude that the node at the root of the tree contains the minimum value. When a node is removed from or inserted into the heap the algorithms that run are tasked with *restoring the invariant*. Similarly, we talked about the invariant for a binary search tree, namely that the value stored at each node is greater than the value at the left child and less than the value at the right child node. When this property is true throughout the tree, then we can extract the values in sorted order by a depth-first left-to-right walk of the tree, and we can easily discover whether the tree contains a particular value. Again, the property is invariant in the sense that if a node is added or removed the algorithms are responsible for adjusting the tree so that the invariant property holds.

Loop Invariants

We now turn to invariants for loops of the form `while boolean-exp statements`. The standard way to handle this is to come up with an invariant property that is always true just before the boolean expression is evaluated – note that it will also be true just before the `statements` begin executing (because evaluating a boolean expression doesn't change the program state) and also when the loop

exits (for the same reason except that the boolean expression now evaluates to false).

We will use the symbol I to stand for a loop invariant – remember this is a predicate about the program state and it is not a program itself. The basic rule for a while loop, `while boolean-exp statements`, is if

$$(I \text{ and } \text{boolean-exp}) \Rightarrow \text{wp}(\text{statements}, \{I\}) \quad (*)$$

then

$$I \Rightarrow \text{wp}(\text{while boolean-exp statements}, \{I \text{ and not boolean-exp}\}) \quad (**)$$

You should think about these formulas for awhile and understand what they are saying by translating to English descriptions (answer below)¹.

Let's look at an example: here is a simple program to add the first n positive integers:

```
sum = 0;
i = 0;
while i<=n {
    sum = sum+i;
    i = i+1;
}
```

Our job is to come up with an invariant (and some other assertions) that let us conclude at the end that $sum == \sum_{j=1}^n n$. Before doing that though, let's think about how the formulas (*) and (**) above will look for this loop. Our invariant is still unknown but we can fill in the formulas for this specific case as follows: if

$$(I \text{ and } i \leq n) \Rightarrow \text{wp}(\text{sum} = \text{sum}+i; i = i+1, \{I\}) \quad (*)$$

then

$$I \Rightarrow \text{wp}(\text{while } i \leq n \text{ sum} = \text{sum}+i; i = i+1, \{I \text{ and } i > n\}) \quad (**)$$

We can see from the code that the value of i when the loop terminates is $n+1$. The $\{I \text{ and } i > n\}$ in the above line is what we can logically conclude is true after the loop according to the semantics so one component of I should be something that is invariant (always true while the loop is executing) but also when combined with $i > n$ will let us conclude $i == n+1$ when the loop terminates. $i \leq n+1$ will do for this because $i \leq n+1$ and $i > n$ implies $i == n+1$.

So now we have part of our assertion I and we can rewrite (*) and (**) once more as, if

$$(I' \text{ and } i \leq n+1 \text{ and } i \leq n) \Rightarrow \text{wp}(\text{sum} = \text{sum}+i; i = i+1, \{I' \text{ and } i \leq n+1\}) \quad (*)$$

then

$$I' \text{ and } i \leq n+1 \Rightarrow \text{wp}(\text{while } i \leq n \text{ sum} = \text{sum}+i; i = i+1, \{I' \text{ and } i \leq n+1 \text{ and } i > n\}) \quad (**)$$

where I' is the part of the invariant we haven't figured out yet.

We can figure out a candidate for I' as follows. It clearly needs to say something about sum and relate it to a summation $\sum_{j=0}^? j$. Since $i == n+1$ at

¹* defines what it means for a predicate I to actually be an invariant; ** says when I actually is an invariant and the loop starts executing in a state satisfying I , we can conclude that when the loop terminates the state satisfies I and not `boolean-exp` – that is, the invariant still holds and the loop control expression is false.

the end and we want the sum to only go to n it seems natural to try $\text{sum} == \sum_{j=0}^{i-1} j$ as the remaining part of the invariant, which gives us for (*) and (**):

if
 $(\text{sum} == \sum_{j=0}^{i-1} j \text{ and } i \leq n+1 \text{ and } i \leq n) \Rightarrow \text{wp}(\text{sum} = \text{sum}+i; i = i+1, \{\text{sum} == \sum_{j=0}^{i-1} j \text{ and } i \leq n+1\})$ (*)
 then
 $\text{sum} == \sum_{j=0}^{i-1} j \text{ and } i \leq n+1 \Rightarrow \text{wp}(\text{while } i \leq n \text{ sum} = \text{sum}+i; i = i+1, \{\text{sum} == \sum_{j=0}^{i-1} j \text{ and } i \leq n+1 \text{ and } i > n\})$ (**)

At this point we don't know whether the if part of this (*) is true – we have to check it using the previously given rules for assignment and sequencing. If it is true, we'll be able to conclude (**) and we'll know that, if i and sum are correctly initialized our loop will do what we want it to. So let's check (*). What is $\text{wp}(\text{sum} = \text{sum}+i; i = i+1, \{\text{sum} == \sum_{j=0}^{i-1} j \text{ and } i \leq n+1\})$? We just have to apply the rules.

$\text{wp}(\text{sum} = \text{sum}+i; i = i+1, \{\text{sum} == \sum_{j=0}^{i-1} j \text{ and } i \leq n+1\})$ is
 $\text{wp}(\text{sum} = \text{sum}+i, \{\text{sum} == \sum_{j=0}^{i+1-1} j \text{ and } i+1 \leq n+1\})$ by the assignment and sequence rules, is
 $\{\text{sum}+i == \sum_{j=0}^{i+1-1} j \text{ and } i+1 \leq n+1\}$ by the assignment rule.

So (*) says, does $\{\text{sum} == \sum_{j=0}^{i-1} j \text{ and } i \leq n+1 \text{ and } i \leq n\}$ imply $\{\text{sum}+i == \sum_{j=0}^{i+1-1} j \text{ and } i+1 \leq n+1\}$? Yes, because $i \leq n$ implies $i+1 \leq n+1$, and $\text{sum} == \sum_{j=0}^{i-1} j$ implies $\text{sum}+i == \sum_{j=0}^i j$. So (*) is true and therefore so is (**), hence when the loop terminates $\{\text{sum} == \sum_{j=0}^n j \text{ and } i == n+1\}$.