CS403: Programming Languages

Axiomatic Semantics (part 3)

Deriving preconditions for loops

Recall that there are two flavors associated with axiomatic semantics. The first is given a precondition and a postcondition, prove that the intervening code fragment is correct. The second is given a postcondition, derive the least-restrictive, or *weakest*, precondition such that when a code fragment is executed, the postcondition holds. We've looked at the first flavor for loops, but not the second. Deriving preconditions for loops requires some bookkeeping, followed by looking for a pattern. All in all, though, the method for finding loop preconditions is straight forward. Consider this loop:

```
while (x > 0)
    {
    x = x - 1;
    y = y + 1;
    }
<<ensures: y == 0 AND x == 0>>
```

Our process will be to push the postcondition through the loop body 0 times, 1 time, 2 times, and so, essentially unraveling the loop. In this case, for up to 3 iterations, we get:

number of iterations	Z	how to get that number of iterations?
0	$y \equiv \quad 0 \wedge x \equiv 0$	$x \leq 0$
1	$y \equiv -1 \wedge x \equiv 1$	$x \equiv 1$
2	$y \equiv -2 \wedge x \equiv 2$	$x \equiv 2$
3	$y \equiv -3 \wedge x \equiv 3$	$x \equiv 3$

At this point, we abandon mechanistic transformations and simply grok the pattern that's generated, noting that for each iteration count, Z and *how?* are ANDed together. First we note that for 0 iterations, $Z \wedge$ *how*? is only true when $x \equiv 0$. Clearly, in this case, the pattern is:

$$x \equiv -y \wedge x \geq 0$$

and this is indeed the weakest precondition. In general, deriving preconditions for loops requires this process of generating and recognizing a pattern.

Another example

Suppose that the post condition had been $y \equiv 0$ with the while loop staying the same. Our table becomes:

number of iterations	Z	how to get that number of iterations?
0	$y \equiv \quad 0$	$x \leq 0$
1	$y \equiv -1$	$x \equiv 1$
2	$y \equiv -2$	$x \equiv 2$
3	$y \equiv -3$	$x \equiv 3$

We note that the 0 iterations row has a different form than the other rows, so let's break that out as a separate part of the precondition. The remaining rows have the pattern:

$$y < 0 \wedge y \equiv -x \wedge x > 0$$

which can be simplified to:

$$y \equiv -x \wedge x > 0$$

Combined with the first row, the precondition is:

$$(y \equiv 0 \wedge x \leq 0) \vee (y \equiv -x \wedge x > 0)$$

which is indeed the weakest precondition.

Yet another example

Consider the loop:

```
while (x > 0)
    {
    x = x / 2;
    y = y + 1;
    }
<<ensures: y == 0>>
```

We generate the following table:

number of iterations	Z	how to get that number of iterations?
0	$y \equiv 0$	$x \leq 0$
1	$y \equiv -1$	$x \equiv 1$
2	$y \equiv -2$	$x \geq 2 \wedge x < 4$
3	$y \equiv -3$	$x \geq 4 \wedge x < 8$
4	$y \equiv -4$	$x \geq 8 \wedge x < 16$

The first two rows seem separate so we treat them individually. We note that the constants in the remaining rows are powers of two so we rewrite them to reflect this fact:

number of iterations	Z	how to get that number of iterations?
0	$y \equiv 0$	$x \leq 0$
1	$y \equiv -1$	$x \equiv 1$
2	$y \equiv -2$	$x \geq 2^1 \wedge x < 2^2$
3	$y \equiv -3$	$x \geq 2^2 \wedge x < 2^3$
4	$y \equiv -4$	$x \geq 2^3 \wedge x < 2^4$

Now the pattern is easier to see. All the rows from 2 and beyond can be summed up with:

$y \leq -2 \wedge x \geq 2^{-y-1} \wedge x < 2^{-y}$

Therefore our weakest precondition is:

$(y \equiv 0 \wedge x \leq 0) \vee (y \equiv -1 \wedge x \equiv 1) \vee (y \leq -2 \wedge x \geq 2^{-y-1} \wedge x < 2^{-y})$

Finally, we note that the *how?* entry for row 1 could have been written as $x \geq 2^0 \wedge x < 2^1$ which also fits the patterns of the subsequent rows, so our precondition can be simplified to:

$(y \equiv 0 \wedge x \leq 0) \vee (y \leq -1 \wedge x \geq 2^{-y-1} \wedge x < 2^{-y})$