Assignment to a simple variable

The Hoare triple for assignment to a simple variable (not an array element) defines that assignment in terms of how one prove it correct. You will be surprised you at its simplicity!

Suppose we want to find the precondition under which execution of x= 2; terminates with $0 \le x \le 4$:

{?} x= 2; $\{0 \le x \le 4\}$

Since the assignment is x= 2; the precondition is the postcondition

$0 \le x \le 4$

but with every occurrence of x replaced by 2:

$0 \le 2 \le 4$

Since the precondition is equivalent to true, we can write:

{true} x= 2; $\{0 \le x \le 4\}$

Thus, in *all* initial states, execution of x= 2; terminates with $0 \le x \le 4$.

Note what we did: Because the assignment was x= 2; the precondition was the postcondition with every occurrence of x in the postcondition by 2.

Here's another example:

{?} x= x+1; $\{x \ge 5\}$

Since the assignment is x= x+1; the precondition is the postcondition $x \ge 5$ but with every occurrence of x in it replaced by x+1:

$x+1 \ge 5$

Which we can rewrite as $x \ge 4$: $\{x \ge 4\}$ x= x+1; $\{x \ge 5\}$

This way of figuring out the precondition works for *every* assignment x= e; to a simple variable x. To define this carefully, we introduce this notation R[x:= e] to denote a copy of assertion R but with each occurrence of x replaced by e. For example, consider this:

$(x*x \ge 5)[x:= x+1]$ is $(x+1) * (x+1) \ge 5$

The value is the expression but with every occurrence of x replaced by x+1.

Definition of assignment

We then *define* the assignment statement like this:

Hoare-triple for x= e;
{R[x:= e] && (evaluation of e terminates normally)} x= e; {R}

The extra term is needed to eliminate cases where evaluation of e aborts (say by dividing by 0) or goes into an infinite loop. For example, in no state will the this statement terminate with y = 5.

{false} x = 6/0; {y = 5}

We take that it for granted that evaluation will terminate normally to simplify later discussions and write the definition this way:

Hoare-triple for x= e;
{R[x:= e]} x= e; {R}

Note that we can think of this as the definition of assignment to a simple variable —not in terms of how to execute it but in terms of how to prove it correct. That's neat!

Definition of assignment

Here's another example. We want find to the precondition for the following sequence:

```
k= k+1;
s= s + k;
{s = sum of m..k}
```

Using the rule just given, we find the precondition of s= s+k:

```
k= k+1;
{s+k  = sum of m..k}
s= s + k;
{s = sum of m..k}
```

Using the rule again, we find the precondition of k= k+1:

```
{s + k+1 = sum of m..k+1}
k= k+1;
{s+k  = sum of m..k}
s= s + k;
{s = sum of m..k}
```

We can simplify this precondition. To make it absolutely clear, we first split off the last tem of the sum and then subtracting k+1 from both sides:

```
   s+k+1 = sum of m..k+1
=     <split off last term of sum>
   s+k+1 = sum of m..k  +  k+1
=     <subtract k=1 from both sides>
   s = sum of m..k
```

So the precondition is s = sum of m..k

```
{s = sum of m..k}
k= k+1;
{s+k  = sum of m..k}
s= s + k;
{s = sum of m..k}
```

We see that the assignments k= k=1; s= s+k; leave the assertion s = sum of m..k true while increasing k by 1.

Two exercises

We give you two exercises to try. In each of the sequences below, calculate the precondition starting from the postcondition. It may help to simplify as you go. Since x and y are being replaced, it helps to keep the number of occurrences of them to a minimum.

For example, you can rewrite x = B && y = x + A as x = B && y = B + A .

The first exercise is relatively easy, because you know that the sequence of statements swaps the values of x and y. What does the second sequence do?

```
{                    }
t= x;
{                    }
x= y;
{                    }
y= t;
{x = B  and  y = C}
```

```
{                    }
x= x + y;
{                    }
y= x – y;
{                    }
x= x – y;
{x = B  and  y = C}
```