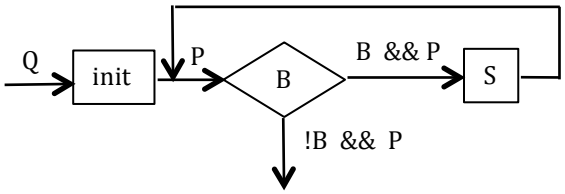
Generalization: Replace a constant by a fresh variable

Replace a "constant" by a fresh variable

Consider the postcondition of a loop to sum the integers of m..n-1:

R: s is the sum of m..n–1

Both m and n are "constants" of the algorithm; they won't be changed. Precondition Q is just the fact about ranges: $m \le n$.



!B && P implies R

The flow chart suggests generalizing R to an invariant P that a simple initialization init can truthify. That's most easily done if the sum can be over the empty range m..m–1. So, we find a suitable invariant by replacing n in the postcondition by a fresh variable k, putting an appropriate limit on k:

P: s is the sum of m..k–1 and $m \le k \le n$

P is a generalization of R! Now, P can be truthified by the initialization

k= m; s= 0; // The sum of no values is 0.

Look at how we generalized to ind an invariant: We replaced n in R by a fresh variable k, a simple step.

R: s is the sum of m..n–1
P: s is the sum of m..k–1 and $m \le k \le n$

This leads to the following loop. You can easily check the four loopy questions.

```
k= m; s= 0;
// invariant: P: s is the sum of m..k–1   and   m ≤ k ≤ n
while (k != n) {
   s= s + k;  k= k+1;
}
{R}
```

The loop can be rewritten as a for-loop, which is what most people would write for this problem:

```
s= 0;
// invariant: P: s is the sum of m..k–1   and   m ≤ k ≤ n
for (int k= m; k != m; k= k+1) {
   s= s + k;
}
{R}
```

Replacing R's other constant:

Postcondition R has another constant, m. Let's replace m by a fresh variable, placing appropriate bounds on it:

P1: s is the sum of h..n–1 and $m \le h \le n$

Now, to make the range empty we set h to n. Since h = m on termination, each iteration has to *decrease* h, and we end up with this algorithm:

```
h= n; s= 0;
// invariant P1:  s is the sum of h..n–1   and   m ≤ h ≤ n
while (h != m) {
          h= h – 1; s= s + h;
}
{R}
```

Conclusion

Replacing a constant by a fresh variable is one basic way to find a loop invariant. It works in many cases. There are other variations of it, for example, we might just extend the range of an existing variable.

You will not be responsible for being able to use this method for finding an invariant on a problem you haven't seen before, for it takes much practice to become facile at it. But, now that you know it, we can use it in presenting algorithms.