19 'Loop invariants'

A simple C routine, performing some integer arithmetic, say, can sometimes be proved correct using mathematical induction. The most important method of proof is by means of *loop invariants.*

Proofs by induction are called for, but that might be too much. It may be enough to *state* a loop invariant, only proving it if it is easy to prove.

(19.1) Definition *A* loop invariant*, for analysing a for- or while-loop, is a formula which holds before the loop starts, and which is preserved in the sense that if it holds at the beginning of an iteration, it holds at the beginning of the next iteration (if any).*

19.1 Squaring the argument

For example, consider the following function

```
int xxx ( int r )
{
  int s = 0;
  int i;
  for ( i=0; i<r; ++i )
  {
    s = s + 2 * i + 1;
  }
  return s;
}
```

Simulation will show that `xxx()` returns r^2. Here is a guess for a loop invariant. 'The i-th iteration' means the iteration with given i. Usually the first iteration is the 0-th iteration.

> At the start of iteration i, $s = i^2$.

We must show that the formula $s = i^2$ is true initially, where i and s are both zero. Obviously true initially.

Then we should show that if it holds true at the beginning of an iteration, then it is true at the end of the iteration also.

If it holds at the beginning of iteration i (the first with $i = 0$),

$$s = i^2$$

Since s is asssigned $s + 2i + 1$, next

$$s = i^2 + 2i + 1 = (i + 1)^2$$

But this is a for-loop, so at the end of the iteration i is replaced by $i + 1$, and again

$$s = i^2$$

(Ultimately i is r and the loop ends and r^2 is returned).

19.2 Euclid's gcd algorithm

```
int euclid_gcd ( int m, int n )
{
  int x,y,z;
  x = m; y = n;
  while ( y > 0 )
  {
    z = x % y;
    x = y; y = z;
  }
  return x;
}
```

Here is an invariant condition:

$$\gcd(x, y) = \gcd(m, n)$$

It is true initially with $x = m$ and $y = n$.

It is easy to justify the invariant condition, granted tht if $x > y > 0$ then[1]

$$\gcd(x, y) = \gcd(y, x \mod y)$$

which is the basis of Euclid's algorithm.

At a given step, if $\gcd(m, n) = \gcd(x, y)$ and $y > 0$, then $\gcd(m, n) = \gcd(y, x \mod y)$.

In a given step, if $y > 0$, x is replaced by y and y by $x \mod y$. Hence $\gcd(m, n) = \gcd(x, y)$ after the step: we have an inveriant condition.

If y ever reaches the value zero, which of course it does, then $\gcd(m, n) = \gcd(x, 0) = x$, which is what is returned. The function is correct.

19.3 Highest power of a dividing b

```
int highest_power ( int a, int b ) // assume b>0 and a >= 2
{
  int p = 1;
  int x = b;
  while ( x % a == 0 )
  {
    p = p * a;
    x = x / a;
  }
  return p;
}
```

[1] $\gcd(x, y)$ is the greatest common denominator of x, y, not what `euclid_gcd` returns.

Invariant: for some i, (actually, at the i-th iteration, but i is not mentioned in the code; initially, the 0-th iteration),

$$p = a^i, \quad \text{and} \quad px = b$$

Initially, $p = 1 = a^0$ and $x = b$ so $px = b$.

Preserved: assuming that $x \mod a = 0$, and for some i,

$$p = a^i, \quad \text{and} \quad px = b$$

at the start of the iteration, then

`p = p*a` makes $p = a^{i+1}$, and

`x = x/a` together make $px = a^{i+1}x/a = a^i x = b$, since because a divides x, $x = x/a \times a$.

When the loop terminates, for some i, $a^i x = b$, but x is not divisible by a, so a^{i+1} does not divide b. So a^i is the highest power of a dividing b, and this is the value of p, the value returned.