This note shows how a nontrivial but still simple algorithm `npow` is proved correct.

Given x and n, how quickly can x^n be computed? We assume that n is an integer and that x is a real number or a matrix or a polynomial for which multiplication takes so much longer than everything else in our algorithm that only the number of such multiplications matters. In the algorithm that follows, "`Matrix`" or "`Polynomial`" can be substituted for "`Real`", whereas a formula like $x^n = \exp(n \cdot \ln(x))$ would not be practical for polynomials x.

As it happens, no simple algorithm is known that computes x^n for every integer $n \geq 0$ in the least possible number of multiplications (and no divisions) for that n. However, here is an algorithm that is optimal in this regard for $|n| < 15$ and otherwise not much worse than optimal:

```
Real Function  npow(Real x, Integer n) :  ... returns x^n .
     Real  z ;
     Begin
        If ( n < 0 ) Return reciprocal( npow(x, -n) ) and Exit ;
        If  ( n = 0 )  Return  Real(1)  and  Exit ;
        While  ( n  is even )  Do  { x := x^2 ;  n = n/2 } ;
        z := x ;
        Do  {  n := floor(n/2) ;
               If  ( n = 0 )  Return  z  and  Exit ;
               x := x^2 ;
               If  ( n  is odd )  z := x·z ;  }  forever :
     End  npow .
```

Let's follow the progress of `npow(X, 15)` as each variable changes its value:

`n = 15 , x = X ,` (Arguments are copied, "passed by value".)
`z = X ,`
`n = 7 , x = X^2 , z = X^3 ,` 2 Real multiplications
`n = 3 , x = X^4 , z = X^7 ,` 2 more Real multiplications
`n = 1 , x = X^8 , z = X^15 ,` 2 more Real multiplications
`n = 0 , return X^15` in 6 Real multiplications.

But there is a faster way: $X^{15} = [(X^2)^2 \cdot X]^2 \cdot [(X^2)^2 \cdot X]$ in 5 Real multiplications. Anyway, the algorithm's number of multiplications must grow like $\Omega(\log_2(|n|))$ as $n \longrightarrow \infty$ since k multiplications are insufficient to compute x^n if $n > 2^k$; can you see why?

How can `npow` be "proved" correct? At best we can try to prove that this algorithm fulfils some expectation expressed as a set of specifications. Here a question arises reminiscent of the question raised by the Roman satirist Juvenal (*ca*. 60 - 140 AD.):

"Sed quis custodiet ipsos Custodes?" ("But who is to watch the Watchers?")

Many a program has been proved correctly to implement incorrect specifications. This is no excuse for shirking such proofs; rather it is a warning that, first, our specifications also have to be proved correct instead of mere wishful thinking perhaps inconsistent, and, secondly, good reasons exist to test programs even if tests can prove no more than that something is wrong.

The best known specifications for `npow(x, n)` = x^n are recursive:

$x^0 := 1$; $x^n := x \cdot x^{n-1}$ for all integers $n > 0$.

The first specification, for $n = 0$, is a *convention* valid for all x including $x = 0$ and $x = \infty$; here " 1 " is the familiar number if x is a number, the identity matrix of the same dimension as x if it is a square matrix, and the constant polynomial 1 if x is a polynomial in variables not mentioned. The second specification leaves x^n undefined for integers $n < 0$; by relaxing the constraint $n > 0$ we can deduce by induction that $x^n = (x^{-n})^{-1} = (x^{-1})^{-n}$ for integers $n < 0$ provided x is a finite nonzero scalar or an invertible (nonsingular) matrix. Presumably `reciprocal(z)` will return `1/z` or z^{-1}, if it exists, and otherwise an $\infty = 0^{-1}$ or a NaN (Not-a-Number) or something like it or, in desperation, an error-message. These contentious details will be left to another occasion together with the roundoff-related reasons for preferring `reciprocal(npow(x, –n))` to `npow(reciprocal(x), –n)` .

The first step in our proof of the correctness of `npow` is the conversion of its specifications to …

$x^0 := 1$; (0) $x^n := (x^{-n})^{-1}$ for $n < 0$; (–1)

$x^n := x \cdot (x^{n-1})$ for odd $n > 0$; (1) $x^n := (x^2)^{n/2}$ for even $n \geq 0$. (2)

The formal proof of that last specification (2) exercises both multiplication's associativity and mathematical induction; it is left to the diligent reader. Confirming formally that if n is odd then $\text{floor}(n/2) = (n-1)/2$ is left to the reader too. Next we merge specifications (1) and (2) into a single recursive rule for the computation of $y = z \cdot x^n$ for nonnegative integers n , namely

if n is odd then $y = (z \cdot x) \cdot (x^2)^{\text{floor}(n/2)}$ else $y = z \cdot (x^2)^{\text{floor}(n/2)}$,

whose formal proof is left again to the reader. Finally, to prove the correctness of `npow` we interpolate comments into its text:

```
Real Function  npow(Real x, Integer n) :  ... invoked to return Y := X^N .
     Real  z ;
     Begin  … having copied the arguments x = X and n = N .
        If ( n < 0 ) Return reciprocal( npow(x, -n) ) and Exit ;
        If  ( n = 0 )  Return  Real(1)  and  Exit ;
           … Specs. (–1) and (0) have been met. Henceforth N > 0 . The next loop
           … invokes spec. (2) repeatedly to keep Y = X^N = x^n for x = X^2 if n = N/2 .
        While  ( n  is even )  Do  { x := x^2 ;  n = n/2 } ;
           … Now Y = X^N for a new X := x and a new reduced odd N := n > 0 , and …
        z := x ; …  z = X for which we shall return Y = X^N = z·(X^2)^((N–1)/2) .
           … The next loop starts with Z = z , N = n > 0 , X = x and Y = Z·(X^2)^floor(N/2).
        Do  {  n := floor(n/2) ;  … New n = floor(N/2) < N .
               If  ( n = 0 )  Return  z  and  Exit ;  … if N was 1 ;
               x := x^2 ;  … else n > 0 and new x = X^2 , so Y = Z·x^n .
               If  ( n  is odd )  z := z·x ;  … new z = Z·x ; else …
                                          … if n is even leave z = Z .
               … Now Y = old( Z·(X^2)^floor(N/2) ) = new( z·(x^2)^floor(n/2) ) .
               … This formula for Y is Loop Invariant.  }  forever ;
     End  npow  … which terminates because 0 < n < N can't repeat forever.
```