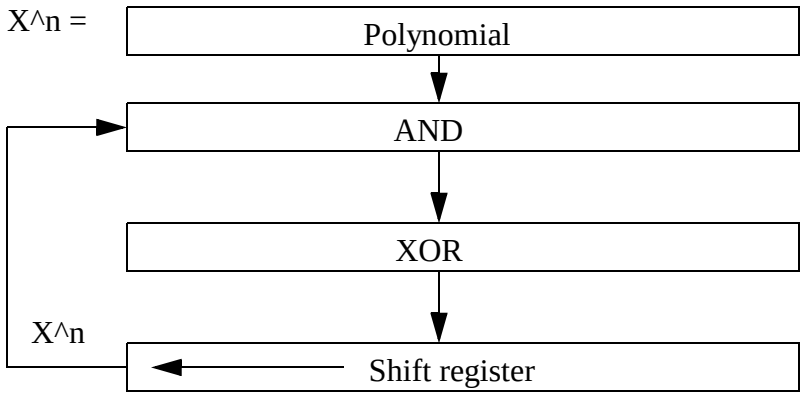
Poly.exe

This program finds polynomials for use as pseudo random sequence generators, aka linear feed-back shift registers (LFSRs). This can also be described as finding the primitive, ie relatively prime, elements of Galios (Gal-wha) fields of size 2^n-1 as used in Reed Solomon forward error correction.



The idea is that, by using these numbers as binary constants, that define the XOR feedback around a shift register, the shift register will cycle though all the non-zero values, that can be represented in as many bits. The sequence generated is of a maximal length, ie 2^n-1 values.

A while back I was looking for this kind of program, but never found one, so I wrote it myself. There are a number of references with canned lists. They generally offer one value per length. Not being a mathematician, I fully expected that there be an elegant mathematical way to find such things, but perhaps this is much the same as scanning for prime numbers. I dunno. This algorithm is essentially brute force, but efficient enough that you will not wait long for results. Up to about 12 bits in length, the results from a modern computer are basically instant. I have considered extending the size range to greater than that of a long Integer, but that would slow the program down, and as it is, values of size approaching 32 bits will try your patients.

This program is based on the idea that other values for the feedback will result in a cycle that is less than 2^n-1 in length. The program has an outer loop that test all odd values that can be represented in as many bits.

The feedback value must be odd because there is no other input to bit zero, having no lesser bits in the shift register to feed it. An even value would leave bit zero unchangeable, for ever, after the first shift. Plus, everyone knows even numbers are not prime, although we are not using normal

arithmetic here. We are using one's complimentary arithmetic, modular within a finite set of numbers, which are the binary numbers 1 to 2^n-1. This is an example of a Galios field.

The inner loop tests each value by counting shifts until the value cycles around again to one, aka alpha^0. If alpha^0 (one) is equal to no power of alpha less than 2^n-1, then this value is a primitive element in G(2^n-1); the LFSR is maximal in length.

Each shift multiplies the value by alpha (x2), and when a bit shifts out to 2^n, we add it's equivalent back into the shift register based on the polynomial we are testing, which defines 2^n=(some odd number less than 2^n), ie within our Galios field. Addition in one's comp arithmetic is simply a bit for bit exclusive-or operation. We have no carries to worry about except 2^n.

While essentially brute force, this plan has an elegant effect in that most values cycle quickly and do not waste much time being tested. Never the less, the inner loop iterations increase as the square of the size of the Galios field, two to the LFSR length x2, so the results slow down quickly above about 12 to 14 bits. I have added a bit of a "more" function to the output for ease of use.

The output also displays the mirror image of the values it finds. It is interesting that these will also be primitive in G(2^n-1). This is perhaps a pointer to a better algorithm for finding them. If fact, I originally wrote this program as a means to verify better methods, which I've yet to get around to. For smaller LFSRs, there is little point in since this program gives instant results, but for LFSRs over 20 bits in length, a more complicated approach should be worth while. Real soon now, eh?