Error Detection and Correction

Manual for a demonstration program

Prof. Dr. W. Kowalk

Universität Oldenburg

kowalk@informatik.uni-oldenburg.de

Version 1.0 (20. September 2006)

Polynomials

A polynomial represents a sequence of bits, where bit n with value $b_n \in \{0,1\}$is represented by a term $b_n \cdot X^n$. The polynomial from the sum of all those terms represents the sequence of bits uniquely.

Example:

111 is represented by $P(X) = X^0+X^1+X^2 = 1+X+X^2$.

0000 is represented by $P(X) = 0$.

101011101101 is represented by $P(X) = X^0+X^2+X^4+X^5+X^6+X^8+X^9+X^{11}$.

We set in these examples the first (or left) term of the bit sequence as bit 0, the last one represents the highest degree of the polynomial (*high ending*), in the example the degree is 11 (which means 12 terms, where those with coefficient 0 are not written). The length of a bit sequence with trailing 0s cannot be represented uniquely by the polynomial.

In other examples, where it is more convenient for us, the last one is defined a bit 0 (*low ending*). We will always state explicitly the corresponding setting.

Tabulators

The tabulators of the program grant access to different functionality.

Logical operations

With bit sequences or polynomials some mathematical operations are useful. Interpreting 0 as *false* and 1 as *true*, bit sequences can be logically manipulated, where *logical and* ($\wedge$), *logical or* ($\vee$), and *logical xor* (*excluvise or* $\oplus$) are useful. The logic is bit by bit or term by term:

Examples

$11111 \wedge 10101 = 10101$,

$11001 \wedge 10101 = 10001$,

$00010 \wedge 10101 = 00000$;

$11111 \vee 10101 = 11111$,

$11001 \vee 10101 = 11101$,

$00010 \vee 10101 = 10111$;

$11111 \oplus 10101 = 01010$,

$11001 \oplus 10101 = 01000$,

$00010 \oplus 10101 = 10111$.

Logical operations of polynomials and division can be found in tabulator: *Logical operations*.

Arithmetical operations

Logical xor can also be interpreted as addition modulo 2. In this case a polynomial *P(X)* is called a polynomial in *X* over the field {0,1}. Thus polynoms can be added, subtracted, multiplied, and divided. While the first three operations can be performed arbitrarily and result always in another polynomial, division is in general possible only with remainder.

Addition and subtraction yield the same result, since 1+1=1-1=0 in the field {0,1}. Multiplication is performed by multiplying two polynomials in the usual way, e.g.

$(X^0+X^2+X^5) \cdot (X^2+X^4) = X^2+X^4+X^7 + X^4+X^6+X^9 = X^2+\underline{X^4+X^4}+X^6+X^7+X^9 = X^2+X^6+X^7+X^9$.

Division is performed by finding to $P(X)$ and $D(X)$ a $Q(X)$ and an $R(X)$ with degree less than $D(X)$ so that

$P(X) = D(X) \cdot Q(X) + R(X)$,

where $P(X)$ is called *dividend*, $D(X)$ is called *divisor* (also called *generator*), $Q(X)$ is called *quotient*, and $R(X)$ is called *remainder*. In context of error detection and correction, the divisor is often called *generator*.

A *prime* $P(X)$ is a polynomial that can be divided with remainder 0 only by itself and the polynomial $A(X) = 1$, which divides any polynomial. A prime factor *P(X)* of a polynomial *S(X)* is a prime that divides *S(X)* with no remainder.

Multiplication of polynomials and estimation of prime factors can be found in tabulator: *Arithmetical operations*.

Tabulator 1's complement

This tabulator demonstrates addition in *One's Complement*, as used by error detection in TCP and IP. A number in *Two's Complement* represents (in *n* bits) a negative number *a* by $2^n-|a|$. A number in *One's Complement* represents (in *n* bits) a negative number *a* by $2^n-1-|a|$. In *One's Complement* there are two zeros, namely $0=0000_b$, and $-0=1111_b$. -1 is represented by $-1=1110_b$.

This type of error detection is relatively unsafe, e.g. not all two bit errors can be detected and not all odd errors.

Examples:

0010+0000=0010, but 00<u>0</u>0+00<u>1</u>0=0010.

0010+0000=0010, but 0001+0001=0010.

Tabulator Bitfilter

A bitfilter is defined in the corresponding script. This tabulator generates one or all bitfilters to a generator.

Tabulator Hamming

Hamming coding can be used for error correction. The tabulator generates a new sequence (depending on length and random seed), the Hamming bits, that control this sequence, and when a bit is altered it determines the position where that changed bit is found.

Hamming coding is called perfect, since all coded values result in a valid bit sequence. Thus perfect codes have no redundancy and cannot distinguish between e.g. 1 and 2 bit errors. See also CRC-Correction.

Tabulator CRC Division

This tabulator demonstrates the method of CRC-Division by shifting in corresponding registers. Bits are ordered low ending, i.e. the first bit is that of the highest order. The steps to be performed by the user are:

1. Set generator by selecting generator's terms (0, .. 15), the term X^{16} is always 1.
2. Set registers to 0.
3. Append 0's before dividing.
4. Divide by generator (use Step or Run).
5. Remove 0's.
6. Append the remainder (i.e. content of registers) computed from division.
7. Transfer output to input field.
8. Set Registers to 0.
9. Divide by generator (use Step or Run).
10. Remainder (i.e. content of registers) should now be 0(x)=0000000...
11. The button 'Make all' generates transfer code (step 2 to 6) in one step.

The remainder can be stored in the field.

Tabulator CRC Correction

This tabulator computes the position of an erroneous bit. The bit sequence is low ending, i.e. the leftmost bit is bit 0 (or X^0=1). The steps to be performed are the following:

1. Generate a random bit sequence (depending on random and length).
2. Make remainder (appends it automatically to the bit sequence, i.e. at the left side).

3. Divide shows remainder is 0.
4. You can alter a bit at an arbitrary position (set position and click 'Alter at').
5. Compute remainder.
6. Clicking 'Find Error' computes the position of the error from the remainder.
7. Clicking 'Find all Errors' sets errors at all positions and computes from the resulting remainder this position (i.e. step 4 to 6 for position 0,1,...,length). Also no error and two errors are tested.

CRC-Correction can distinguish between one and two bit errors, since all even number of bit errors leave a remainder with even number of terms. Thus this redundancy can be used to avoid error correction in case of two bit errors. However, three or any other odd number of bit errors cannot be distinguished (in general) by a single bit error and may result in wrong error 'correction'.

References

Kowalk/Burke: Rechnernetze (1992, Teubner).

Kowalk. Bitfilter. (Online: http://einstein.informatik.uni-oldenburg.de/papers/Bitfilter.pdf)

Kowalk. Bitfilter (engl).
(Online: http://einstein.informatik.uni-oldenburg.de/papers/CRC-BitfilterEng.pdf)