

# Introduction to Logplex Encoding

*Binary universal codes of asymptotically optimal efficiency which increase monotonically with their corresponding whole numbers.*

Russell Leidich

<https://agnentropy.blogspot.com>

May 11, 2017

Keywords: logplex, universal, code, encoding, compression, discrete, whole, integer, affine, infinite, recursive, Elias, omega, exponential, Golomb, Turing

## 0. Abstract

Logplex codes are universal codes, that is, bitstrings which map one-to-one to the whole numbers, regardless of the bits which follow them in memory. The codes are dense, in the sense that there is no finite series of bits which does not map to at least one whole number. Their asymptotic efficiency (size out divided by size in) is one, as with Elias omega codes[1], but they have some convenient features absent in the latter:

Given whole numbers M and N. If ( $M < N$ ) then ( $\text{logplex}(M) < \text{logplex}(N)$ ). This provides for more efficient searching and sorting, as such tasks can be done without the need to allocate separate memory for the corresponding decoded whole numbers.

For all nonzero M, M itself is encoded verbatim in the high bits of its logplex. In all cases, the high (last) bit of a logplex is one.

Representation of all subparts of logplexes are bitwise little endian. This is in contrast to Elias omega codes, the endianness of the subparts of which are opposite to the expansion direction.

Finally, logplexes are scale-agnostic: there is no need to assume that  $(\log_2 M)$  has any particular maximum value. This feature stems from their recursive structure, which is analogous to that of Elias omega codes.

## 1. The Encoding Algo

Given whole number  $M$ . To  $\text{logplex}(M)$  using simple bitstring operations starting from a null string, use the following algo. Note that "MSB" means "most significant bit", which actually means the *position* of the highest one bit, as opposed to an actual bit.

1. Set the bitstring to one.
2. If  $(M \leq 1)$ , then goto 13.
3. Set  $B$  to the MSB of  $M$ .
4. Append all  $(B+1)$  bits of  $M$  to the bitstring, most significant first. Thus the first bit stored will be a one.
5. Decrement  $B$ .
6. Set  $(M=B)$ .
7. If  $(M \leq 1)$ , then goto 12.
8. Set  $B$  to the MSB of  $M$ .
9. Clear bit  $B$  of  $M$ .
10. Append  $(B+1)$  bits of  $M$  to the bitstring, most significant first. Thus the first bit stored will be a zero.
11. Goto 5.
12. Append a zero to the bitstring.

13. Append a single bit equal to M to the bitstring.

14. Reverse the order of all bits heretofore written to the bitstring. The result is  $\text{logplex}(M)$ .

## 2. Examples

The following table provides a series of logplexes and their equivalent Elias omega codes (offset by one, as the omega scheme provides no encoding for zero). The leftmost bit is stored at position zero in all cases. Spaces have been inserted to distinguish subparts.

| M    | $\text{logplex}(M)$ | $\text{elias\_omega}(M+1)$ |
|------|---------------------|----------------------------|
| 0    | 01                  | 0                          |
| 1    | 11                  | 10 0                       |
| 01   | 00 01               | 11 0                       |
| 11   | 00 11               | 10 100 0                   |
| 001  | 10 001              | 10 101 0                   |
| 101  | 10 101              | 10 110 0                   |
| 011  | 10 011              | 10 111 0                   |
| 111  | 10 111              | 11 1000 0                  |
| 0001 | 00 00 0001          | 11 1001 0                  |
| 1001 | 00 00 1001          | 11 1010 0                  |
| 0101 | 00 00 0101          | 11 1011 0                  |
| 1101 | 00 00 1101          | 11 1100 0                  |
| 0011 | 00 00 0011          | 11 1101 0                  |

|          |                 |                   |
|----------|-----------------|-------------------|
| 1011     | 00 00 1011      | 11 1110 0         |
| 0111     | 00 00 0111      | 11 1111 0         |
| 1111     | 00 00 1111      | 10 100 10000 0    |
| 00001    | 00 10 00001     | 10 100 10001 0    |
| 10001    | 00 10 10001     | 10 100 10010 0    |
| ...      | ...             | ...               |
| 1001011  | 10 100 1001011  | 10 110 1101001 0  |
| 11011101 | 10 010 11011101 | 10 111 10111011 0 |

### 3. Asymptotic Behavior

The asymptotic size of  $\text{logplex}(M)$  has ratio one with respect to the following series:

$$\log_2 M + \log_2(\log_2 M) + \log_2(\log_2(\log_2 M)) + \dots$$

Thus as expected the ratio of the logplex size to the size of  $M$  itself also approaches one.

### 4. The Decoding Algo

A bitstring in memory can be decoded to a whole number  $M$  by the following algo:

1. Load the first 2 bits. Set  $M$  to the low (first) bit. If the next bit is one, then stop. Otherwise, set ( $B=M+1$ ).
2. Set  $M$  to the next ( $B+1$ ) bits of the bitstring.
3. If the high bit of  $M$  is one, then stop.

4. Set bit B of M.

5. Increment M.

6. Goto 2.

## Bibliography

[1] [https://en.wikipedia.org/wiki/Elias\\_omega\\_coding](https://en.wikipedia.org/wiki/Elias_omega_coding)