Example of Arithmetic Coding

We have 3 symbols in the alphabet, with probabilities

$$p(1) = 0.8 \qquad p(2) = 0.02 \qquad p(3) = 0.18$$

The cdf corresponding to this pdf has values:

$$F(0) = 0 \qquad F(1) = 0.8 \qquad F(2) = 0.82 \qquad F(3) = 1$$

The input sequence we will code is

$$X_1, X_2, X_3, X_4 \ldots = 1, 3, 2, 1 \ldots$$

We initialize the lower and upper endpoints of the interval to be

$$l^{(0)} = 0 \qquad u^{(0)} = 1$$

After reading input symbol X_n, we will update the sub-interval endpoints as follows:

$$l^{(n)} = l^{(n-1)} + [u^{(n-1)} - l^{(n-1)}]F(X_n - 1)$$

$$u^{(n)} = l^{(n-1)} + [u^{(n-1)} - l^{(n-1)}]F(X_n)$$

We begin with the first symbol: $X_1 = 1$. We apply the endpoint updating equations to obtain:

$$l^{(1)} = 0 + (1 - 0) \times 0 = 0$$

$$u^{(1)} = 0 + (1 - 0) \times 0.8 = 0.8$$

This interval straddles 0.5, so there is no output of bits. So we proceed to look at the next symbol: $X_2 = 3$. We find the new sub-interval by applying the endpoint updating equations:

$$l^{(2)} = 0 + (0.8 - 0)F(2) = 0.8 \times 0.82 = 0.656$$

$$u^{(2)} = 0 + (0.8 - 0)F(3) = 0.8 \times 1 = 0.8$$

Again, we compare to 0.5, and we see this is entirely in the upper half of the unit interval. So we output the bit 1. Now we can rescale with the rescaling function E_2. Recall that the two possible re-scaling functions are

$$E_1(x) = 2x$$

$$E_2(x) = 2(x - 0.5)$$

where E_1 is used for rescaling the lower half, and E_2 is used for rescaling the upper half. Rescaling our little interval with E_2:

$$l^{(2)} = 2 \times (0.656 - 0.5) = 0.312$$

$$u^{(2)} = 2 \times (0.8 - 0.5) = 0.6$$

This straddles 0.5, so there is no further release of bits at this time. We look at the next input symbol, $X_3 = 2$.

$$l^{(3)} = 0.312 + (0.6 - 0.312)F(1) = 0.5424$$

$$u^{(3)} = 0.312 + (0.6 - 0.312)F(2) = 0.54816$$

We got a very low-probability input, so the interval suddenly got very small. So now there's going to be a release of many bits. After each release of a bit to the decoder, we rescale. We compare to the threshold of 0.5, and see we're in the upper half of the unit interval, so we release the bit 1, and rescale with E_2 to get:

$$l^{(3)} = 0.0848$$

$$u^{(3)} = 0.09632$$

Now it's in the lower half interval, so release the bit 0, and rescale with E_1 to obtain:

$$l^{(3)} = 2 \times 0.0848 = 0.1696$$

$$u^{(3)} = 2 \times 0.09632 = 0.19264$$

Still in the lower half interval, so release another bit 0, and rescale again with E_1 to obtain

$$l^{(3)} = 2 \times 0.1696 = 0.3392$$

$$u^{(3)} = 2 \times 0.192640.38528$$

Still in the lower half interval, so release another bit 0, and rescale again with E_1 to obtain

$$l^{(3)} = 2 \times 0.3392 = 0.6784$$

$$u^{(3)} = 2 \times 0.38528 = 0.77056$$

They are now both in the upper half interval, so release the bit 1, and rescale with E_2, to obtain

$$l^{(3)} = 2 \times (0.6784 - 0.5) = 0.3568$$

$$u^{(3)} = 2 \times (0.77056 - 0.5) = 0.54112$$

Now the interval straddles 0.5, so we can't output any more bits. We have to look at the next input symbol. We'll stop the example here. Note that so far we have output the bits 110001.

For comparison, let's try encoding this with an Elias encoder. We begin with the first symbol: $X_1 = 1$, and it corresponds to the interval [0, 0.8). There is no output of bits. The next symbol is $X_2 = 3$, and we get the endpoints with the same endpoint updating equations:

$$l^{(2)} = 0 + (0.8 - 0)F(2) = 0.8 \times 0.82 = 0.656$$

$$u^{(2)} = 0 + (0.8 - 0)F(3) = 0.8 \times 1 = 0.8$$

We compare to 0.5, see this is entirely in the upper half of the unit interval, and output the bit 1. So far, this is exactly like the arithmetic encoder. At this point, the arithmetic encoder would rescale and again check the threshold of 0.5. The Elias encoder doesn't rescale. Instead, it checks the threshold of 0.5 + 1/4 = 0.75 (that is, we're looking to see if the endpoints agree in the second bit of their binary expansions). The current interval straddles 0.75, so there is no further release of bits at this time.

We look at the next input symbol, $X_3 = 2$, and use the endpoint updating equations to get:

$$l^{(3)} = 0.656 + (0.8 - 0.656)F(1) = 0.7712$$

$$u^{(3)} = 0.656 + (0.8 - 0.656)F(2) = 0.77408$$

We check 0.75, looking to see if the endpoints now agree in the second bit of their binary expansions. They do, as they are both above 0.75. So we release the bit 1.

We check the 3rd bit of the binary expansion. Are the endpoints both above or both below 0.75 + 1/8 = 0.875? They are both below, so release the bit 0.

We check the 4th bit of the binary expansion. Are the endpoints both above or both below (0.875-1/16) = 0.875-0.0625 = 0.8125? They are both below, so release the bit 0.

We check the 5th bit of the binary expansion. And so forth. You can continue the example.

The point is this: The arithmetic encoder and the Elias encoder are producing the same set of output bits (110001...) for the same input sequence (132...). The difference is in the *internal* representation of the intervals, and what threshold is being used for comparison. The arithmetic encoder is always checking against 0.5. The Elias encoder is getting finer and finer precision on the representation, and is checking against a sequence of thresholds, in this case: 0.5, 0.75, 0.875, 0.8125, etc.