

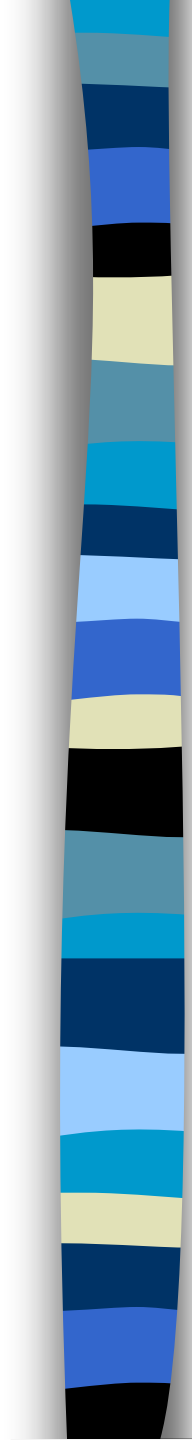
ITCT Lecture:7.2

Implementation of Arithmetic Codes

Prof. Ja-Ling Wu

Department of Computer Science
and Information Engineering
National Taiwan University



- 
- The rationale for using numbers in the interval $[0,1)$ as a **tag** was that there are **infinite number of numbers** in this interval.
 - However, in practice the number of numbers that can be **uniquely represented** on a machine is **limited by the maximum number of digits** (or bits) we can use for representing the number.
 - Consider $Newinterval_{low}$ and $Newinterval_{high}$. As n gets larger, these **values come closer and closer together**. This means that in order to represent all the subintervals uniquely, we have to increase **precision** as the length of the sequence increases.

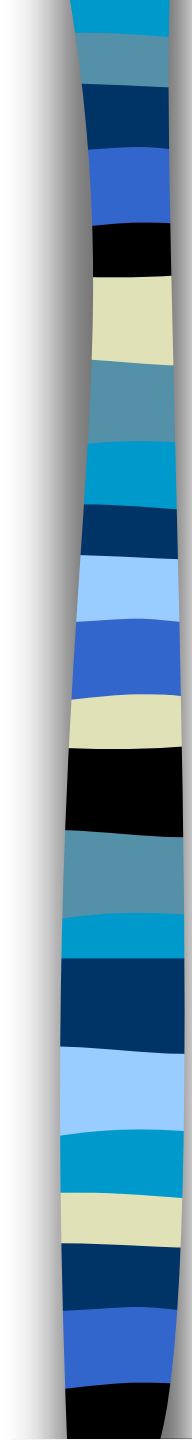


- In a system with **finite precision**, the **two values are bounded to converge**, and we will lose all information about the sequence from the point at which the two values converged.
- To avoid this situation, we need to **rescale the interval!**
- However, we have to do it in a way that will **preserve the information that is being transmitted**. We could also like to **perform the encoding incrementally** – that is, **to transmit portions of the code as the sequence is being observed**, rather than wait until the entire sequence has been observed before transmitting the first bit.



- As the interval becomes narrow, it is very likely that the interval containing the tag will be confined either to the upper or to the lower half of the $[0,1)$ interval. Therefore, the most significant bit of the tag is fully determined.
- If the tag is confined to the upper half of the unit interval, the tag is a number greater than or equal to 0.5, and the first bit of the tag has to be 1.
- If the tag is confined to the lower half of the unit interval, the value of the tag is less than 0.5 and the first bit of the tag is 0.
- In this situation, we can indicate to the decoder which half the tag is confined to be sending a 1 for the upper half and a 0 for the lower half.
- The binary value that we send is also the first bit of the tag.



- 
- Once the encoder and decoder know **which half contains the tag**, we can **ignore the half of the unit interval not containing the tag** and concentrate on the half containing the tag.
 - As our arithmetic is of finite precision, we can do this best by **mapping the half interval containing the tag to the full $[0,1)$ interval**. The mapping required are

$$E_1: [0, 0.5) \rightarrow [0, 1) : E_1(X) = 2 X$$

$$E_2: [0.5, 1) \rightarrow [0, 1) : E_2(X) = 2 (X - 0.5)$$



- As soon as we perform either of these mappings, we lose all information about the most significant bit. However, this should not matter as **we have already sent that bit to the decoder.**
- We can continue with this process, **generating another bit of the tag every time the tag interval is restricted to either half of the unit interval.**
- This process of generating the bits of the tag without waiting to see the entire sequence is called **Incremental Encoding.**



Example:

$$p(a_1) = 0.8, \quad p(a_2) = 0.02, \quad p(a_3) = 0.18.$$

Encode the sequence a_1, a_3, a_2, a_1 .

Initializing $u^{(0)}$ to 1, and $l^{(0)}$ to 0.

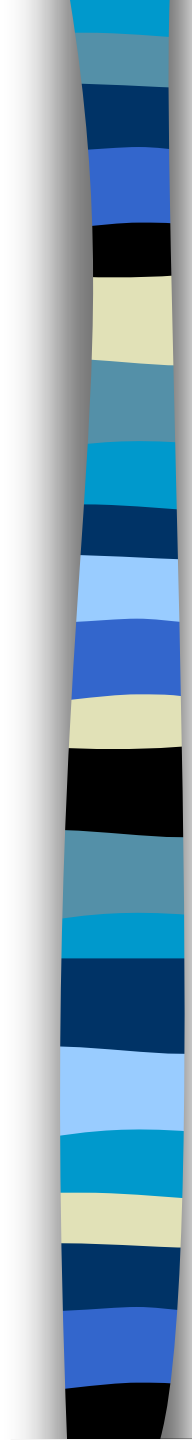
The first element of the sequence, a_1 , results in the following update:

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

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

The interval $[0, 0.8)$ is not confined to either the upper or lower half of the unit interval, so we proceed.



- 
- The second element of the sequence is a_3 . This results in the following update:

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

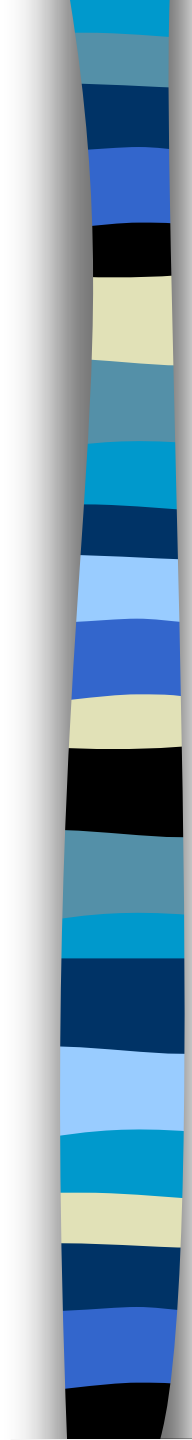
$$u^{(2)} = 0 + (0.8 - 0) \times 1.0 = 0.8 .$$

- The interval $[0.656, 0.8)$ is contained entirely in the upper half of the unit interval, so we send the binary code 1 and rescale:

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

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



- 
- The third element, a_2 , results in the following update equation:

$$l^{(3)} = 0.312 + (0.6 - 0.312) \times 0.8 = 0.5424$$

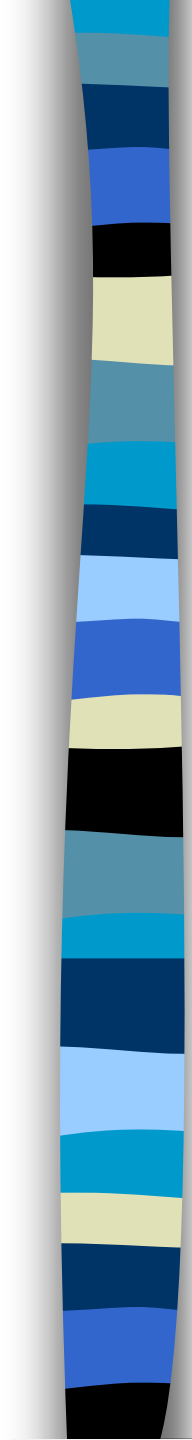
$$u^{(3)} = 0.312 + (0.6 - 0.312) \times 0.82 = 0.54816 .$$

- The interval for the tag is $[0.5424, 0.54816)$, which is contained entirely in the upper half of the unit interval. We transmit a 1 and go through another rescaling:

$$l^{(3)} = 2 \times (0.5424 - 0.5) = 0.0848$$

$$u^{(3)} = 2 \times (0.54816 - 0.5) = 0.09632 .$$



- 
- This interval is contained entirely in the lower half of the unit interval, so we send a 0 and use the E_1 mapping to rescale:

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

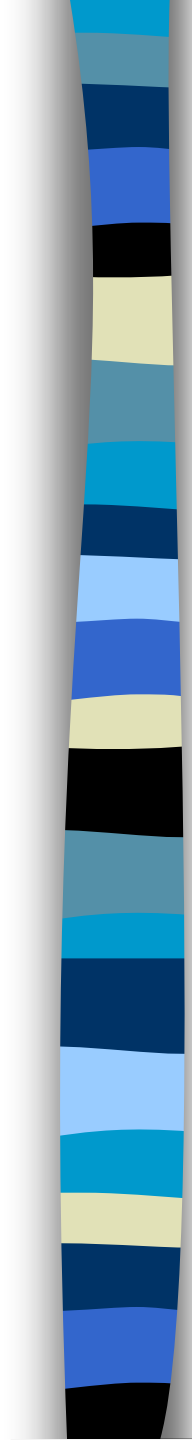
$$u^{(3)} = 2 \times (0.09632) = 0.19264 .$$

- This interval is still contained entirely in the lower half of the unit interval, so we send another 0 and go through another rescaling:

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

$$u^{(3)} = 2 \times (0.19264) = 0.38528 .$$



- 
- Because the interval containing the tag remains in the lower half of the unit interval, we send another 0 and rescale one more time:

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

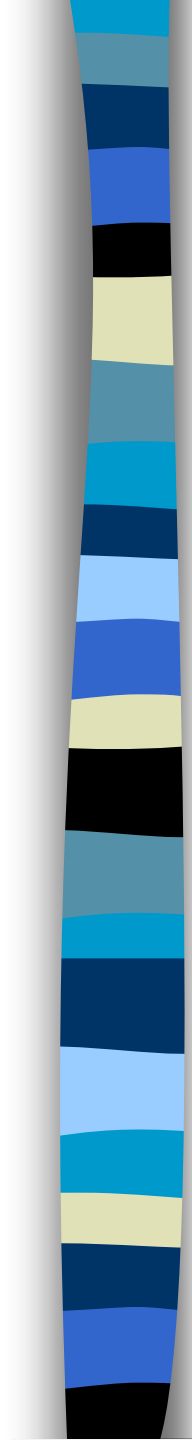
$$u^{(3)} = 2 \times (0.38528) = 0.77056 .$$

- Now the interval containing the tag is contained entirely in the upper half of the unit interval. We transmit a 1 and rescale using the E_2 mapping:

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

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



- 
- The interval $[0.3528, 0.54112)$ is not confined to either the upper or lower half of the unit interval, so we proceed. The fourth element of the sequence is a_1 . This results in the following update:

$$l^{(4)} = 0.3568 + (0.54112 - 0.3568) \times 0 = 0.3568$$

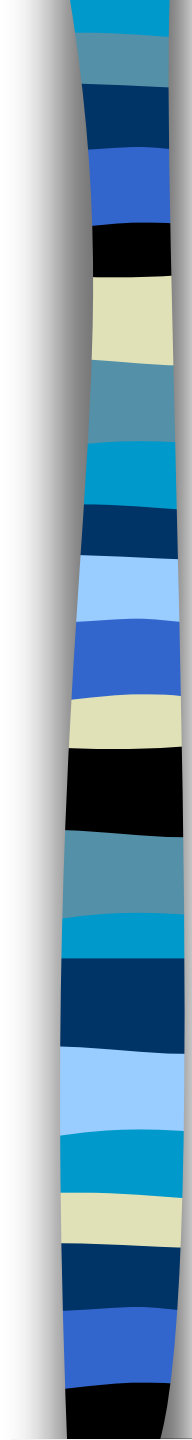
$$u^{(4)} = 0.3568 + (0.54112 - 0.3568) \times 0.8 = 0.504256 .$$

- At this point, if we wish to **stop encoding**, all we need to do is **inform the receiver of the final status of the tag value**.

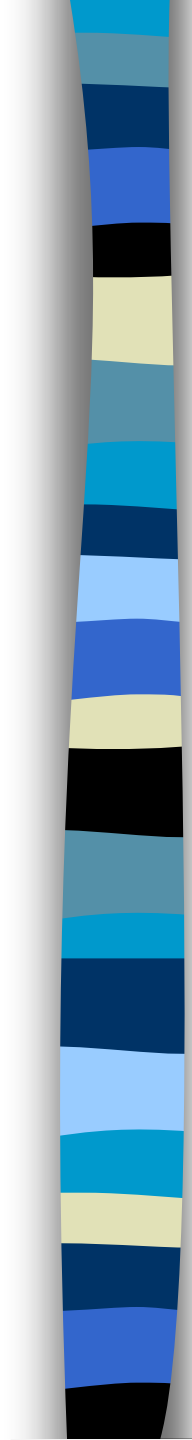


- We can do so by sending the binary representation of any value in the final tag interval. Generally, this value is taken to be $l^{(n)}$.
- In this particular example, it is convenient to use the value of 0.5.
- The binary representation of 0.5 is $.100\dots 0$. Thus, we would transmit a 1 followed as many as 0's as required by the word length of the implementation being used.
- The binary sequence we sent is: 1100011.
- A binary number $.1100011$ corresponds to the decimal number 0.7734375.
- By Arithmetic Encoding $\overline{F}(a_1, a_3, a_2, a_1) \in [0.7712, 0.773504)$.



- 
- References:
 - 1. Special Issue on Q-coder (an adaptive binary arithmetic coder): IBM J. Research & Develop. Nov. 1998.
 - 2. Moffat, Neal Witten, “Arithmetic Coding Revisit,” ACM T. on Inform. Systems, pp. 256 – 294, July 1998.
 - 3. Detlev Marpe et. al., “Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard,” IEEE T. on CSVT, pp. 620 -636, July 2003. (**Best Paper Award**)



- 
- 4. Detlev Marpe, et. al., “ Probability Interval Partitioning Entropy Codes,” submitted to IEEE T. on Information Theory.
 - A novel approach to entropy coding is described that provides the coding efficiency and simple probability modeling capability of arithmetic coding at the complexity level of Huffman coding.

