Űrkommunikáció
Space Communication
2023/5.

Source coding, ***Entropy Coding*** (Type III):

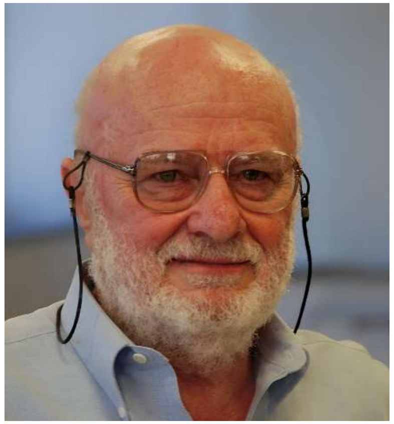
Encoding the source **with considering the Entropy is also called *Entropy Coding*** (Type III):

- Encode **variable length k source words to fixed length l code words**
- **Without a-priori knowledge of statistical properties of the source**.

Typical examples: **Run-length encoding** (**RLE**) and a generalization of RLE:

Lempel-Ziv algorithm (LZ)

- Invented by Israeli computer scientists **Abraham Lempel** and **Jacob Ziv:** *„A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory, 1977*
- "The **LZ algorithms** were the first major successful universal compression algorithms". Lempel and Ziv developed algorithm enable **perfect data reconstruction from compressed data** and are more efficient than previous algorithms.
- Uses the **source text** (series of source symbols) itself **as the dictionary, replacing later occurrences of a string by numbers (pointers)** indicating where it occurred before and its length.
- **Zip** and **gzip** use variations of the Lempel-Ziv algorithm.



IEEE: **Abraham Lempel**

IEEE: **Jacob Ziv**

Source coding, ***Entropy Coding*** (Type III):

Run-length encoding (RLE)

- A form of **lossless data compression** in which **runs of data** (sequences in which the same data value occurs in many consecutive data elements) are **stored as a single data** value **and count**, rather than as the original run.
- This is most **useful on data that contains many such runs**.
- It is not useful with files that **don't have many runs** as it could greatly **increase the file size**.

Example: consider a screen containing plain black (B) text on a solid white (W) background.

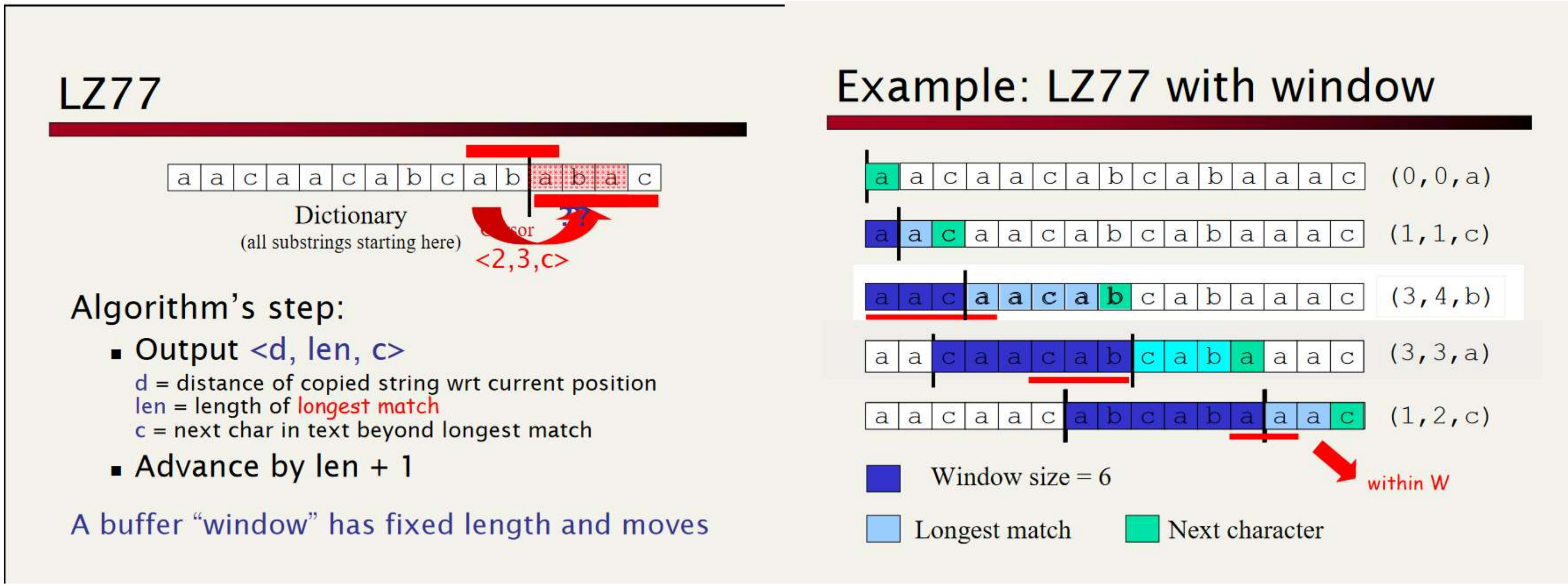
- A line could be for example:

WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWWWW

- The run-length encoded line: **12W1B12W3B24W1B14W**
- The original ***67 characters are encoded in only 18*** characters.
- In data where **runs** are **less frequent**: encodes run lengths for runs of two or more characters only. A character appears twice denotes a run: **WW12BWW12BB3WW24BWW14**
- Could be combined with other encoding methods: e.g. first RLE and thereafter Huffman

Source coding, ***Lempel-Ziv Coding*** (Type III):

Jacob Ziv and Abraham Lempel developed two lossless data compression algorithms:
Lempel-Ziv 77 (**LZ77**) in 1977 and **LZ78** the following year.
In general, the algorithms build up dynamically a **dictionary of phrases**, or so called phrase-book, where each phrase is a chained list of source symbols.



What if len > d? (overlap with text to be compressed) => Simply copy starting at the cursor
Other example: we already have abcd and the codeword (2,9,e)

abcd|cdcdcdcdce

Source coding, ***Lempel-Ziv Coding*** (Type III):

LZ78 algorithm

Dictionary: Each stored substring ***S*** has an ***id*** number.

Encoding algorithm:

- ✓ Find the longest match ***S*** in the dictionary.
- ✓ Output its ***id*** and the next character ***c*** after the match in the input string.
- ✓ Add the substring ***Sc*** to the dictionary.

Decoding: builds the **same dictionary** and looks at ids.

Problems: How do we **keep the dictionary small**?

- ➢ Throw the dictionary away when it reaches a certain size (used in GIF)
- ➢ Throw the dictionary away when it is no longer effective at compressing (e.g. compress)
- ➢ Throw the least-recently-used (LRU) entry away when it reaches a certain size (used in BTLZ, the British Telecom standard)

LZ78: Coding Example

	Output	Dict.
a a b a a c a b c a b c b	(0,a)	1 = a
a a b a a c a b c a b c b	(1,b)	2 = ab
a a b a a c a b c a b c b	(1,a)	3 = aa
a a b a a c a b c a b c b	(0,c)	4 = c
a a b a a c a b c a b c b	(2,c)	5 = abc
a a b a a c a b c a b c b	(5,b)	6 = abcb

LZ78: Decoding Example

Input		Dict.
(0,a)	a	1 = a
(1,b)	a a b	2 = ab
(1,a)	a a b a a	3 = aa
(0,c)	a a b a a c	4 = c
(2,c)	a a b a a c a b c	5 = abc
(5,b)	a a b a a c a b c a b c b	6 = abcb

LZ78 Example: Let’s consider an addressable computer memory for storing the phrases as chained characters and a symbol sequence of binary source to be compressed e.g.:

WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWWWWW
000000000000100000000000011100000000000000000000000000001000000000000000
0 00 0000000001 00000 000000 011 10000000 00000000 0000000001 000000000

LZ78 Encoder

Address (4 bits)	Stored Pointer	Stored Symbol	Codeword (e.g. l=5 bits)
0000=NIL			
0001	**0000**	**0**	**00000**
0010	**0001**	**(0)0**	**00010**
0011	**0010**	**(00)0**	**00100**
0100	**0011**	**(000)0**	**00110**
0101	**0010**	**(00)1**	**00101**
0110	**0100**	**(0..0)0**	**01000**
0111	**0110**	**(0..0)0**	**01100**
1000	**0001**	**(0)1**	**00011**
1001	**0000**	**1**	**00001**
1010	**1001**	**(1)0**	**10010**
1011	**0111**	**(0..0)0**	**01110**
1100	**1011**	**(0..0)0**	**10110**
1101	**1100**	**(0..0)1**	**11001**
1110	**1100**	**(0..0)0**	**11000**

LZ78 Decoder

Address (4 bits)	Stored Pointer	Stored Symbol	Output Symbols
0000=NIL			
0001	**0000**	**0**	**0**
0010	**0001**	**0**	**00**
0011	**0010**	**0**	**000**
0100	**0011**	**0**	**0000**
0101	**0010**	**1**	**001**
0110	**0100**	**0**	**00000**
0111	**0110**	**0**	**000000**
1000	**0001**	**1**	**01**
1001	**0000**	**1**	**1**
1010	**1001**	**0**	**10**
1011	**0111**	**0**	**000000 0**

Encoder output = Decoder input:

000000001000100001100010101000…

Decoder output:

0 00 0000000001 00000000000 01110 ….

Further LZ78 Example:

Encoder input (Text to be compressed):

ALIBABAABBABELEBLABLABLABLABLABLA…

LZ78 Encoder

Address (4 bits)	Stored Pointer	Stored Symbol	Codeword (e.g. l=5 bits)
0000=NIL			
0001	**0000**	**A**	**0000A**
0010	**0000**	**L**	**0000L**
0011	**0000**	**I**	**0000I**
0100	**0000**	**B**	**0000B**
0101	**0001**	**(A)B**	**0001B**
0110	**0001**	**(A)A**	**0001A**
0111	**0100**	**(B)B**	**0100B**
1000	**0101**	**(AB)E**	**0101E**
1001	**0010**	**(L)E**	**0010E**
1010	**0100**	**(B)L**	**0100L**
1011	**0101**	**(AB)L**	**0101L**
1100	**1011**	**(ABL)A**	**1011A**
1101	**1010**	**(BL)A**	**1010A**
1110	**1101**	**(BLA)B**	**1101B**

LZ78 Decoder

Address (4 bits)	Stored Pointer	Stored Symbol	Output Symbols
0000=NIL			

Encoder output = Decoder input:

0000A0000L0000I0000B0001B0001A0100B0101E0010E0100L0101L1011A1010A1101B

Decoder output:

Source coding, ***Entropy Coding*** (Type IV):

Encoding the source **with considering the Entropy is also called *Entropy Coding*** (Type IV):

- Encode **variable length k source words to variable length l code words**
- **a-priori knowledge of statistical properties** (first – or even higher – order PDF) of the source are needed.

Arithmetic Coding

Motivation

- **Huffman code** by encoding the source symbol by symbol **can be inefficient**
  - ➢ This can be **"solved"** through **source extension** encoding symbol blocks of the source
  - ➢ But the **number of codewords grows exponentially**
  - ➢ And **higher order statistics** are needed (e.g. 2nd, 3rd, etc. PDFs)
- **Underlying difficulty**: Huffman requires keeping track of **codewords for all possible symbol blocks**

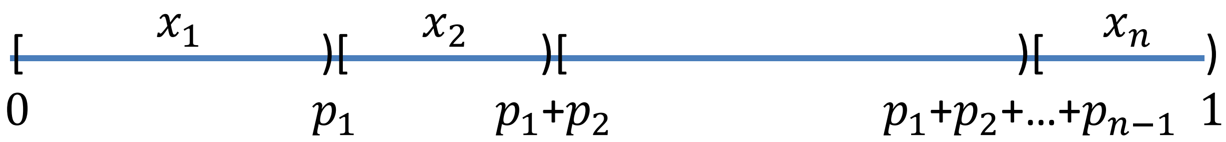
Solution

- We need a way to **assign a codeword to a particular sequence** without having to generate codewords for all possible sequences.

Source coding, **Arithmetic Coding**

- The idea is **based on Shannon's algorithm**:

Let an RV $X = \{x_1, x_2, \cdots, x_n\}$ with the PDF $p(X) = \{p(x_1) \geq p(x_2) \geq \cdots \geq p(x_n)\}$
Consider the partitioning of the zero to one interval according the event probabilities into subintervals. Each subinterval closed from the left side (lower limit) and open from the right side (upper limit) corresponds to an event. At the end we have the **cumulated probabilities** $1=\sum_{i=1}^{n} p(x_i)$



- **Rissanen and Langdon** "IBM Journal of Research and Development" around 1980.

✓ **Any rational number** of the given interval **could be a code for the symbol belongs to that interval.**

✓ **Encode the sequence** of source **symbols** in such way, that **the actual partial interval** selected by the current symbol is **<u>subdivided</u>** again **according to the symbol PDF** and the **consecutive symbol select the next partial interval**.

✓ Select the **shortest rational number of the finally resulting partial interval** as the **<u>code-word for the particular symbol sequence of source.</u>**

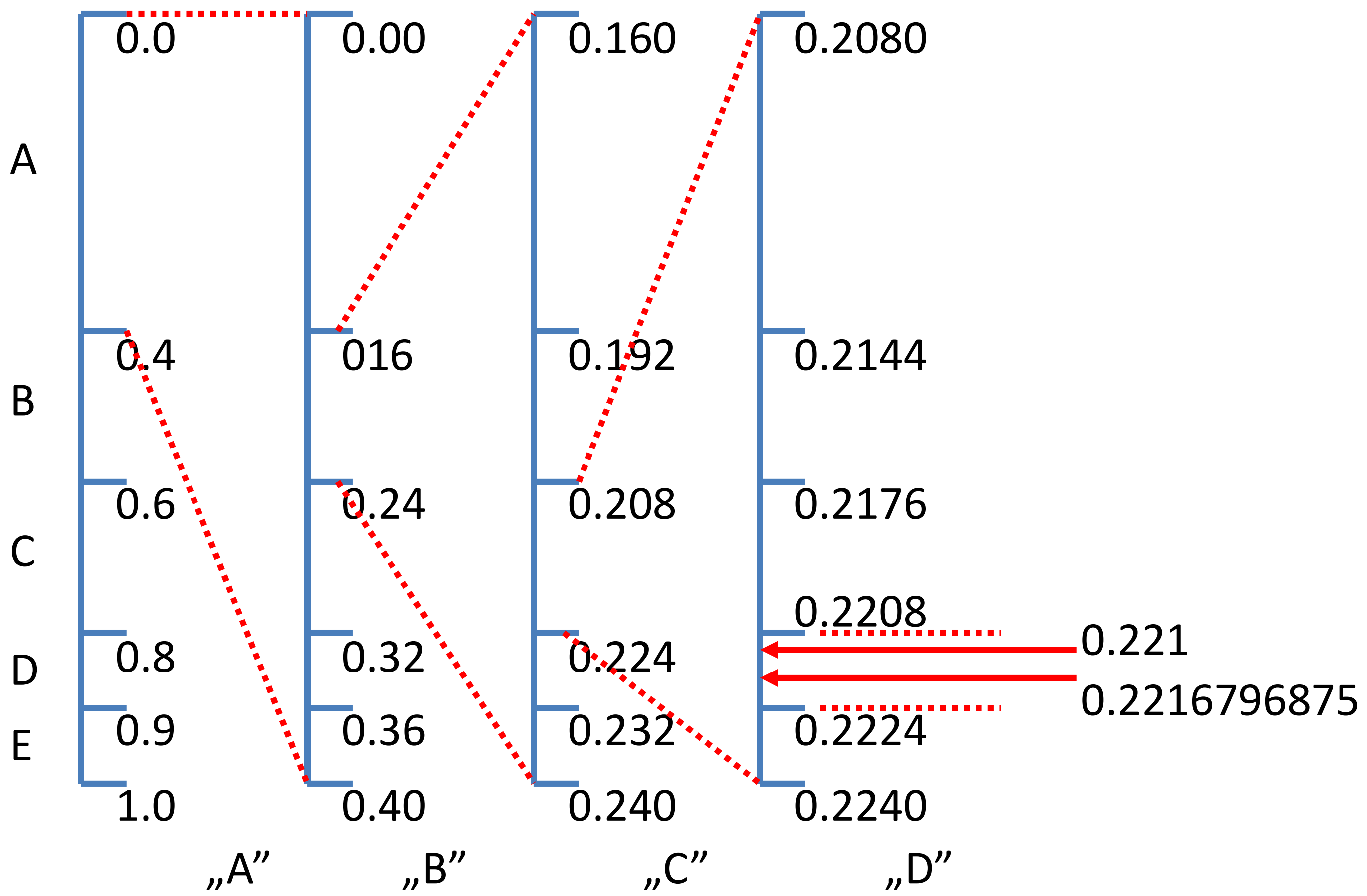
The size of the finally resulting partial interval will be low, i.e. the interval will be very small, when the particular encoded source symbol sequence has a low probability.

- Our chance to find a short rational number will be low => longer code-word.

Source coding, **Arithmetic Coding, Example**

Let $X = \{A, B, C, D, E\}$, with $p(X) = \{$p(A)=0.4, p(B)=0.2, p(C)=0.2, p(D)=0.1, p(E)=0.1$\}$

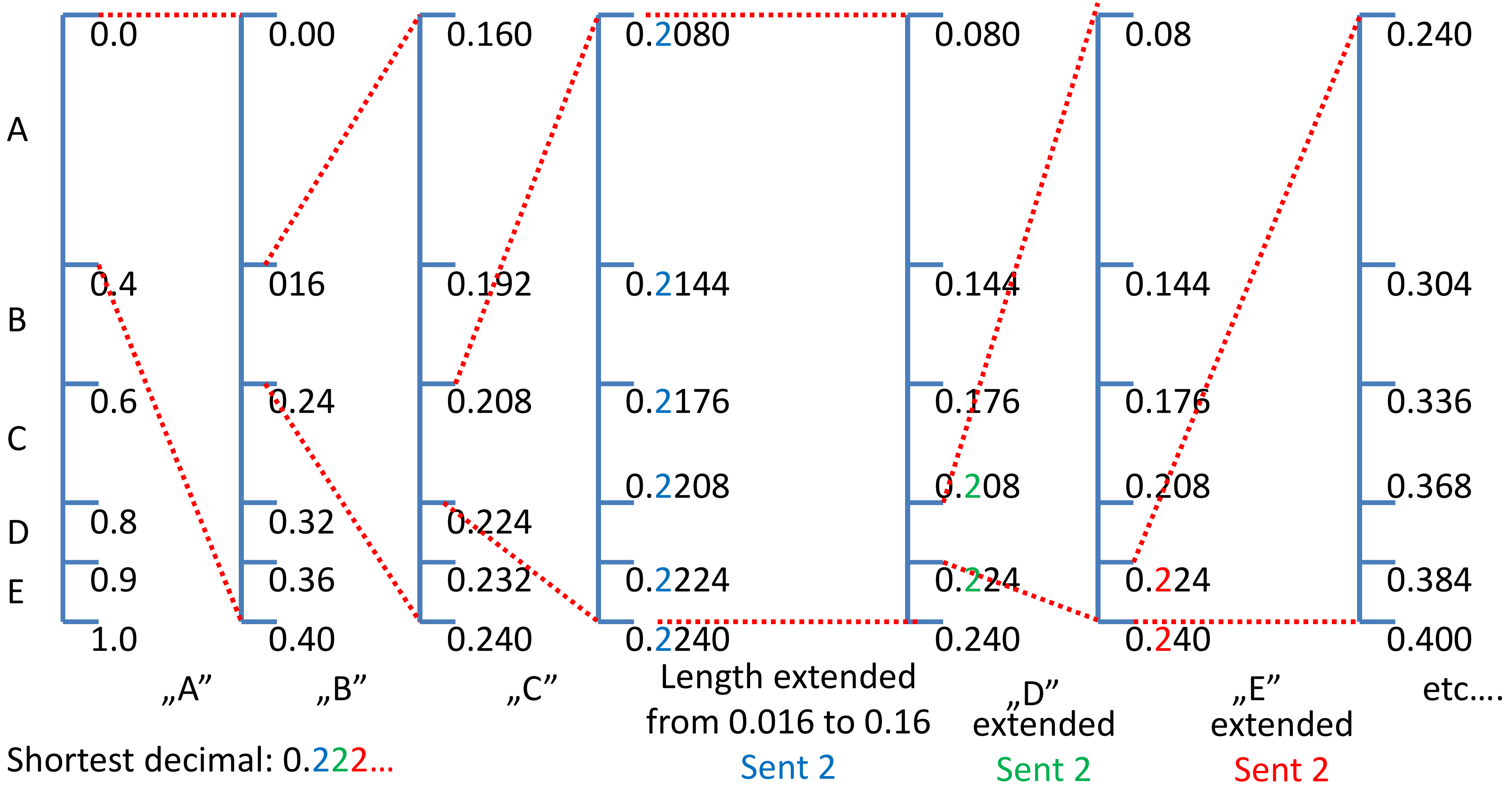
Encode the source symbol sequence „ABCD”



Shortest decimal: 0.221 Shortest binary: 0.0011100011 (in decimal: 0.2216796875)

Arithmetic Coding, Interval extension

If the first fractional digits of the **upper** and the **lower endpoint are identical**, **they cannot change** in the course of the encoding any more. These digits can be **sent to the data stream**. These parameters of the **endpoints have to be shifted accordingly**.



Shortest decimal: 0.222…

Arithmetic Coding, Decoding

Problem: **When to stop**? Find the variable length code-word for a variable length source symbol sequence.

- Fixed length code-word (e.g. 3 digits) identify different source symbol sequences.
- Fixed length source symbol sequences would be encoded in code-words of different lengths.

Solution: Insert a **virtual "STOP" symbol** – meaning this has just a probability in the source symbol sequence.

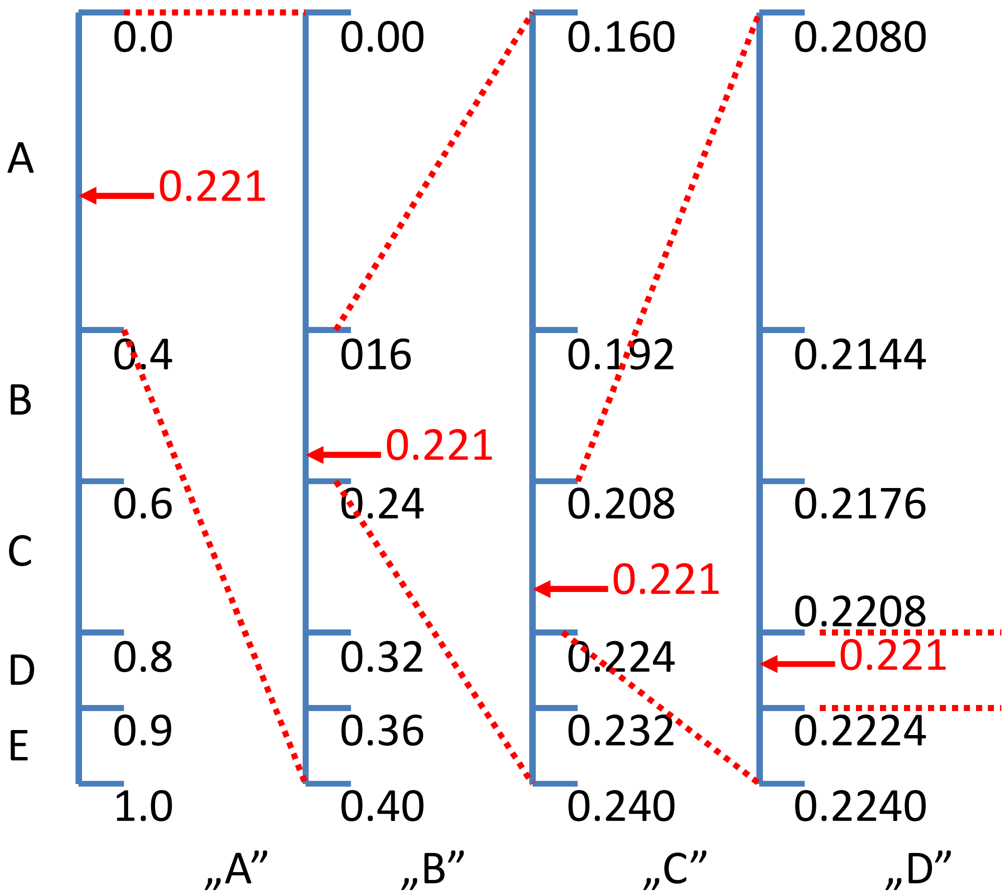
E.g. "EOL", "EOF", "EOT", etc.

„AAAA" [0.0 .. 0.064)

–> 0 (one digit)

„ABCD" [0.2208 .. 0.114)

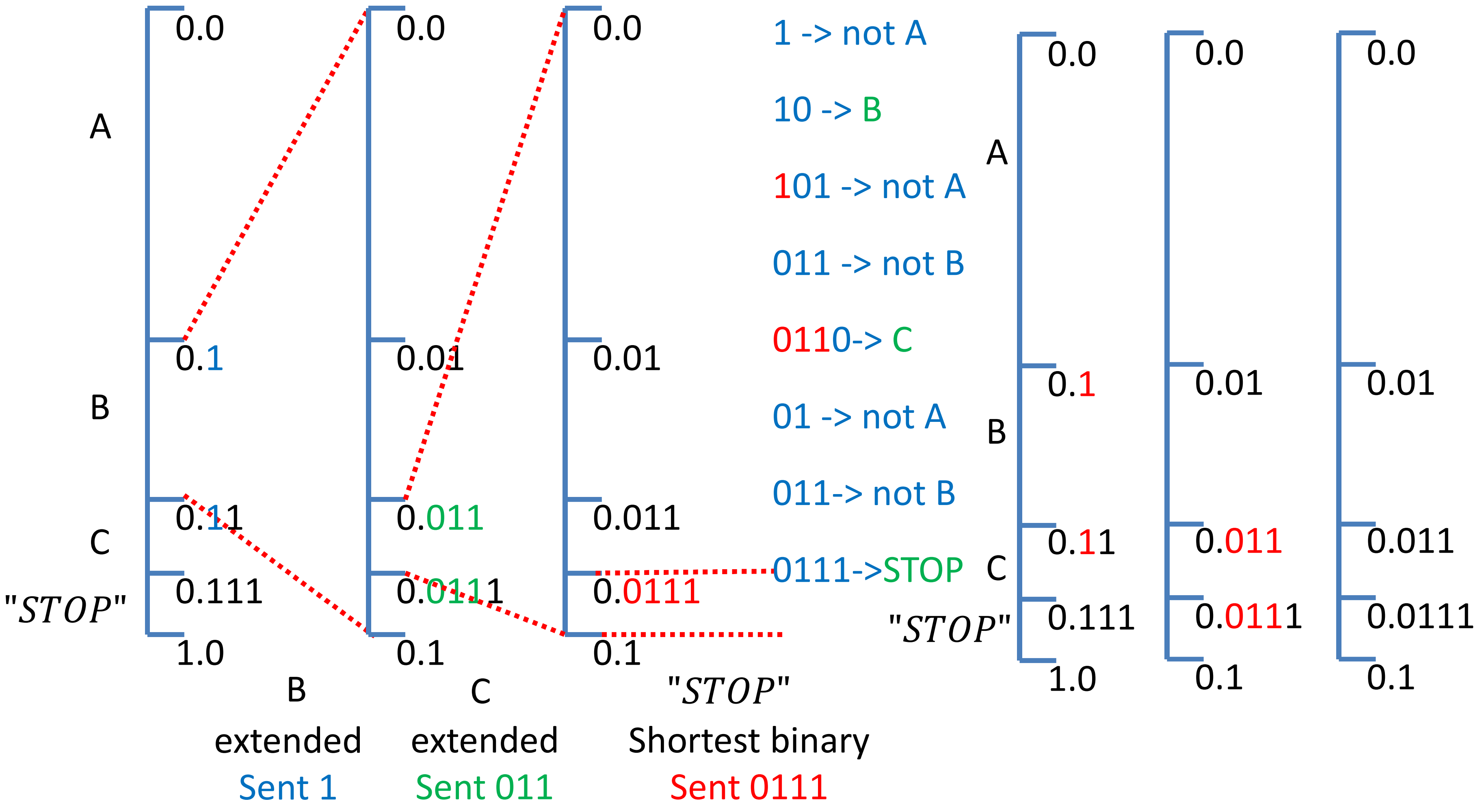
–> 221 (three digits)



Source coding, **Arithmetic Coding, Example**

Let $X = \{A, B, C, "STOP"\}$, with $p(X) = \{$p(A)=1/2, p(B)=1/4, p(C)=1/8, p("STOP")=1/8$\}$

Encode the source symbol sequence BC "$STOP$" **Decode** the received binary: 10110111

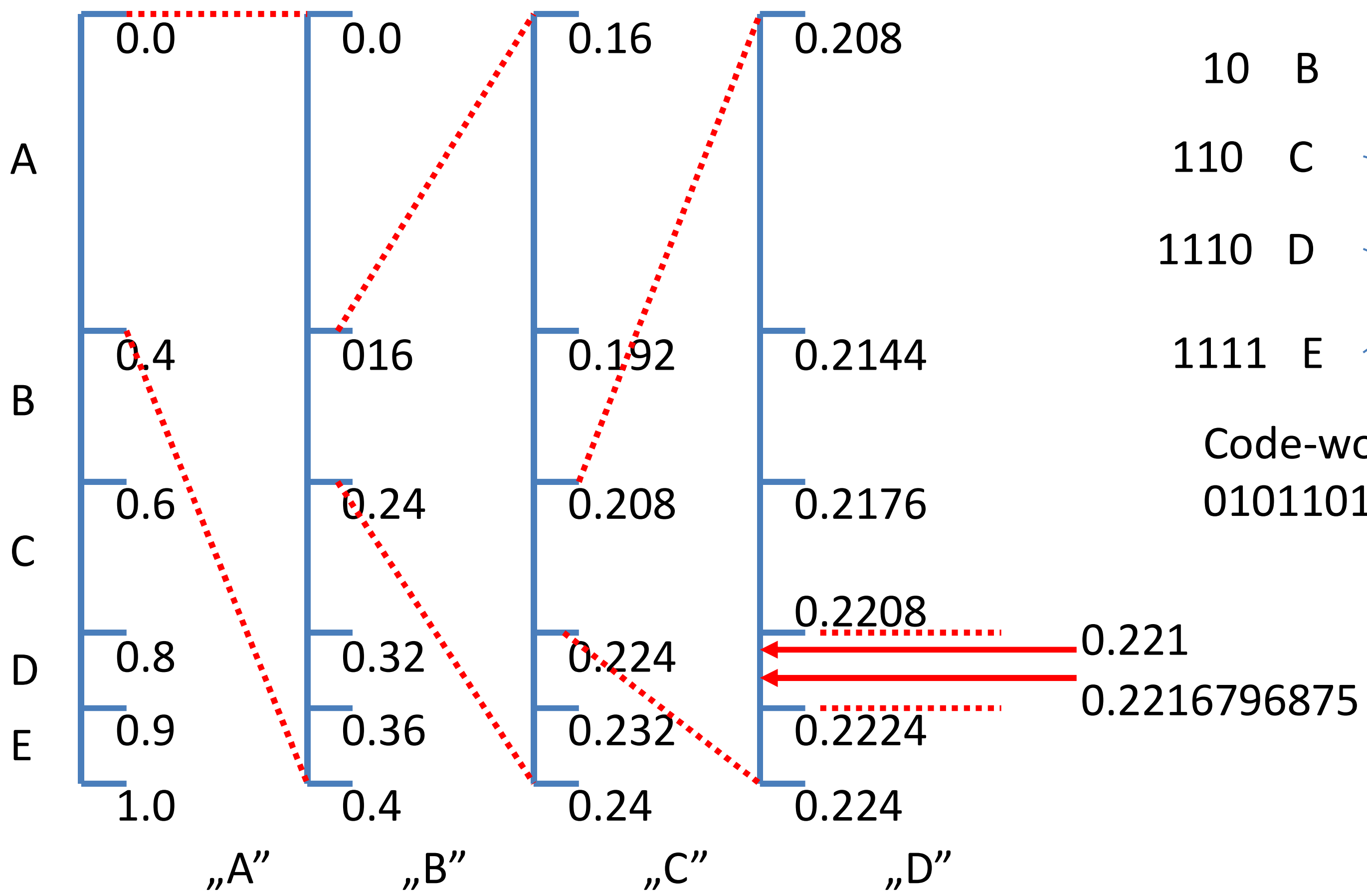


sent binary: 10110111

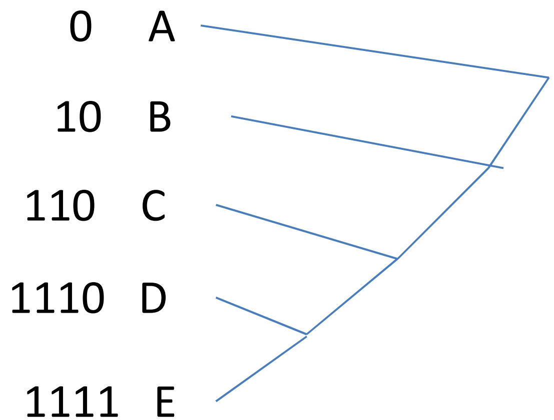
Arithmetic Code vs. Huffman Code

The efficiency of **an arithmetic code is always better or at least identical to a Huffman** code.

Encode the source symbol sequence „ABCD"



Compared to Huffman:



Code-word for „ABCD"
0101101110

Shortes decimal: 0.221

Shortest binary: 0.0011100011 (in decimal: 0.2216796875)