Teoretične osnove računalništva 2
Teorija informacij

Source coding

Enes Pasalic UP FAMNIT študijsko leto 20/21

Topics

- Coding:
  - basics
- Trenutne kode (Prefix-free codes)
- Kraft inequality
- Kraft-McMillan result
- Code length and probability
- Shannon‘s result on source coding

- Shannon‘s coding
- Shannon-Fano coding
- Huffman coding
- Problems related to symbol coding
- Arithmetic coding

Shannonove information and coding

- <u>Game ’63’:</u>
  - How many questions with answers yes/no we need to ask for guessing the number in the range 0 to 63 ?
  - Discussion.

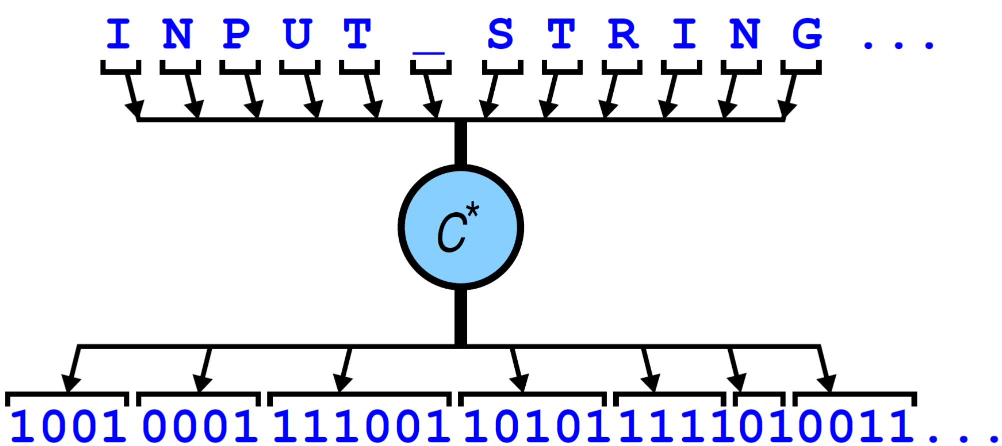
Coding of symbols

(Binary) **encoding of symbols** is a mapping from alphabet $\mathcal{X}$ to $\{0,1\}^*$, i.e.,

$$C : \mathcal{X} \to \{0,1\}^*.$$

Extended coding (to arbitrary length of source symbols) is mapping $C^* : \mathcal{X}^* \to \{0,1\}^*$ obtained by merging individual codewords $C(x_i)$ so that:

$$C^*(x_1, \ldots, x_n) = C(x_1)C(x_2)\cdots C(x_n).$$



Symbol coding

- Three important properties for symbol coding:
  - Any encoded sequence can be uniquely decoded.
  - Decoding must be simple.
  - Coding must be such that the codewords are of shortest possible average length (source compression).

Unique decoding

Unique decoding

A code C is uniquely decodable exactly when the extended coding is injective so that:

$$(x_1, \ldots, x_n) \neq (y_1, \ldots, y_n) \Rightarrow C^*(x_1, \ldots, x_n) \neq C^*(y_1, \ldots, y_n).$$

- Using codewords $\{0, 1, 10, 11\}$ we **cannot** decode uniquely.
- Using codewords $\{00, 01, 10, 11\}$ we **can** decode uniquely.
- Using codewords $\{0, 01, 011, 0111\}$ we **can** decode uniquely.

Predponske kode (prefix-free codes)

Prefix-free coding

A prefix-free code $C : \mathcal{X} \to \{0,1\}^*$ has the property that none of its codewords is a prefix of some other codeword.

- Any prefix-free coding is uniquely decodable !
- Any codeword from a prefix-free code can be instaneously decoded, called also *instantenuous codes*.
- Primeri:
  - Codewords from $\{0, 01, 011, 0111\}$ can be uniquely decoded but it is not a prefix-free code, e.g. 0 is prefix of 01.
  - Codes $\{0, 10, 110, 111\}$ belong to a prefix-free code.

Symbol encoding - examples

Example for symbols and prob. distrib.: $\mathcal{X} = \{a, b, c, d\}$
$p_X = \{1/2, 1/4, 1/8, 1/8\}$

Source entropy: $H(X) = 1.75\text{bita}$

Encoding example:

x_i	$c(x_i)$	$p(x_i)$	$l(x_i)$
a	0	1/2	1
b	10	1/4	2
c	110	1/8	3
d	111	1/8	3

Average length of codewords:

$$E(l(x)) = \sum_{x_i \in \mathcal{X}} p(x_i) l(x_i) = 1.75\text{bita}$$

Summary: $2^{-l(x_i)} = p(x_i)$

Symbol encoding – examples II

Example for symbols and prob. distrib :

$$\mathcal{X} = \{a, b, c, d\}$$
$$p_X = \{1/2, 1/4, 1/8, 1/8\}$$

Source entropy: $H(X) = 1.75\text{bita}$

Encoding example:

x_i	$c(x_i)$	$p(x_i)$	$l(x_i)$
a	00	1/2	2
b	01	1/4	2
c	10	1/8	2
d	11	1/8	2

Average length of codewords:

$$E(l(x)) = \frac{1}{2}2 + \frac{1}{4}2 + \frac{1}{8}2 + \frac{1}{8}2 = 2\text{bita}$$

Conclusion: $E(l(x)) > H(X)$

Symbol encoding – examples III

Example for symbols and prob. distrib :
:

$$\mathcal{X} = \{a, b, c, d\}$$
$$p_X = \{1/2, 1/4, 1/8, 1/8\}$$

Source entropy: $H(X) = 1.75\text{bita}$

Encoding example:

x_i	$c(x_i)$	$p(x_i)$	$l(x_i)$
a	0	$1/2$	1
b	1	$1/4$	1
c	00	$1/8$	2
d	11	$1/8$	2

Average length of codewords:

$$E(l(x)) = \frac{1}{2}1 + \frac{1}{4}1 + \frac{1}{8}2 + \frac{1}{8}2 = 1.25\text{bita}$$

Conclusion: $E(l(x)) < H(X)$

No unique decoding:
acdbac -> 000111000 -> cabdca

Kraft inequality

- Length of the codewords for any prefix-free code satisfy the following property:

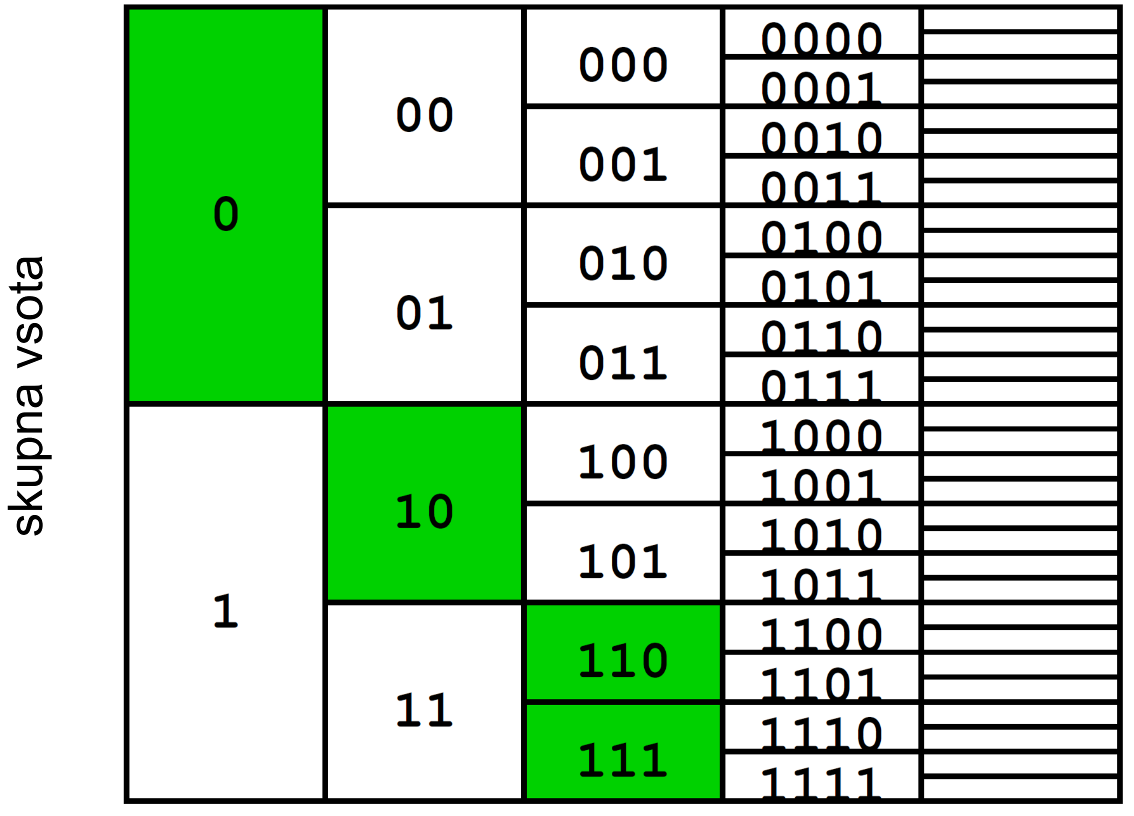
Kraft inequality

The lengths $\ell_1, \ldots, \ell_m$ of any (binary) prefix-free code satisfy the following inequality:

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 1.$$

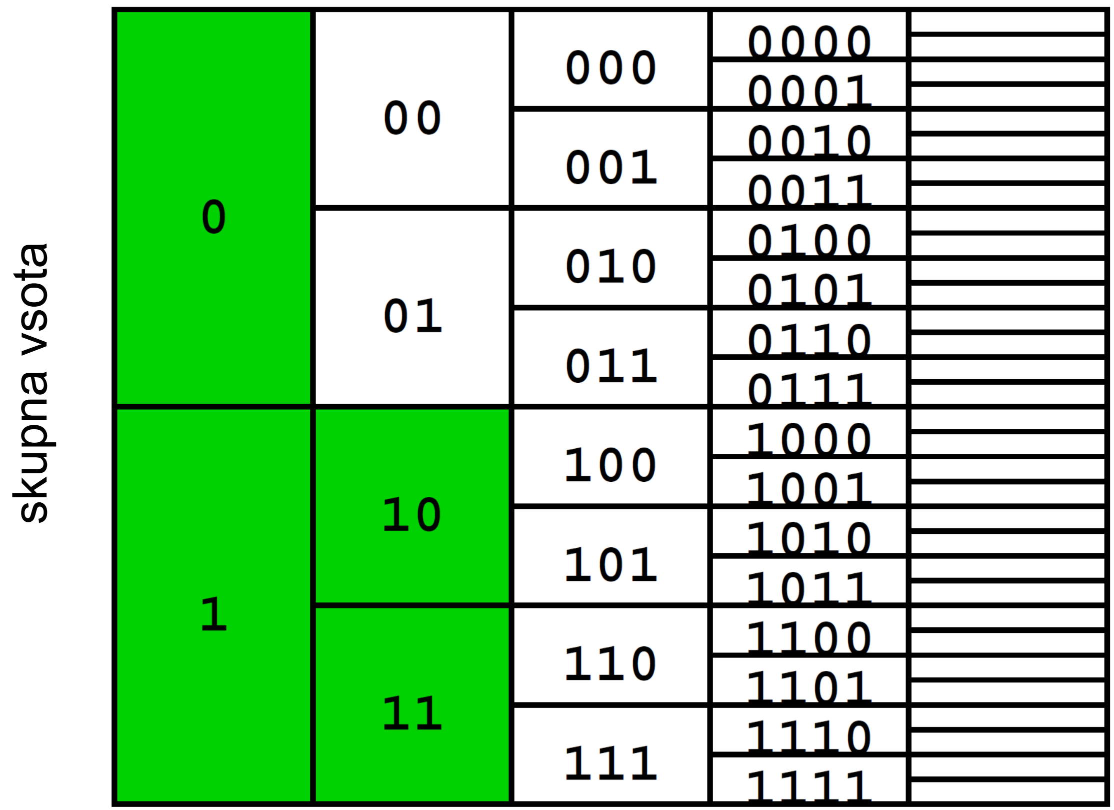
Conversely: If we have the lengths ℓ_i that satisfy the above inequality then there exists a prefix-free code with these lengths.

Kraft inequality and coding



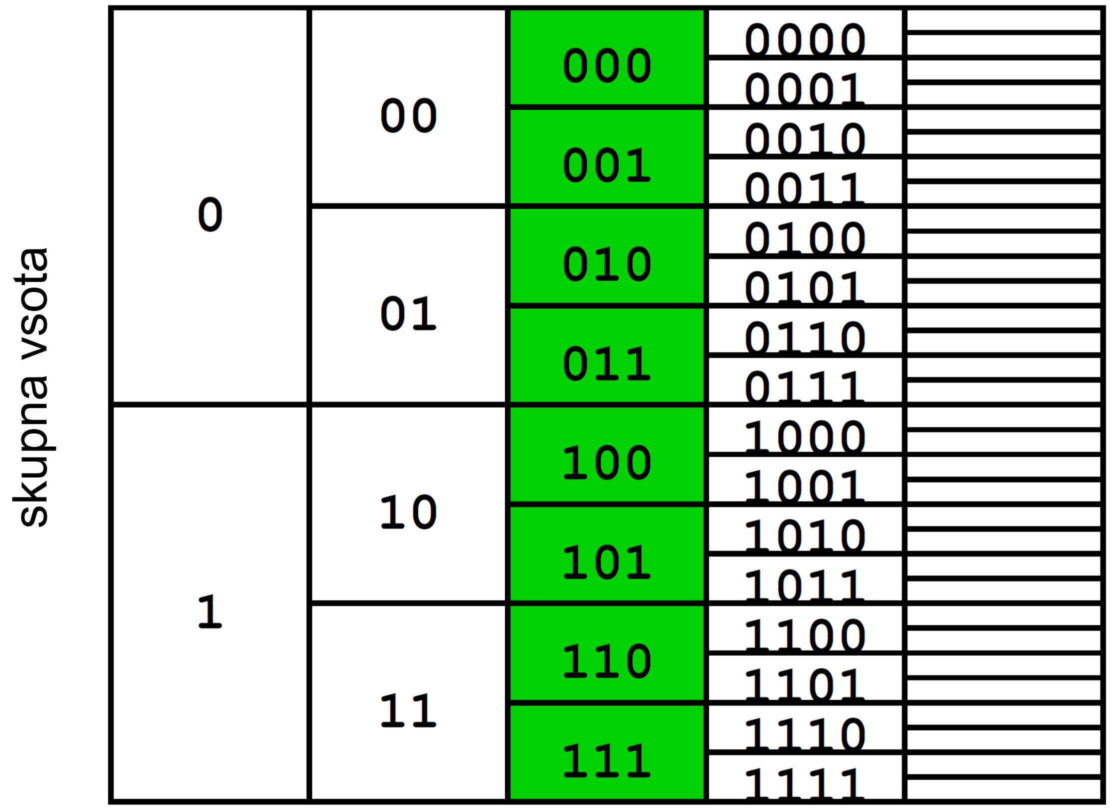
codewords {0, 10, 110, 111}

Kraft inequality and coding



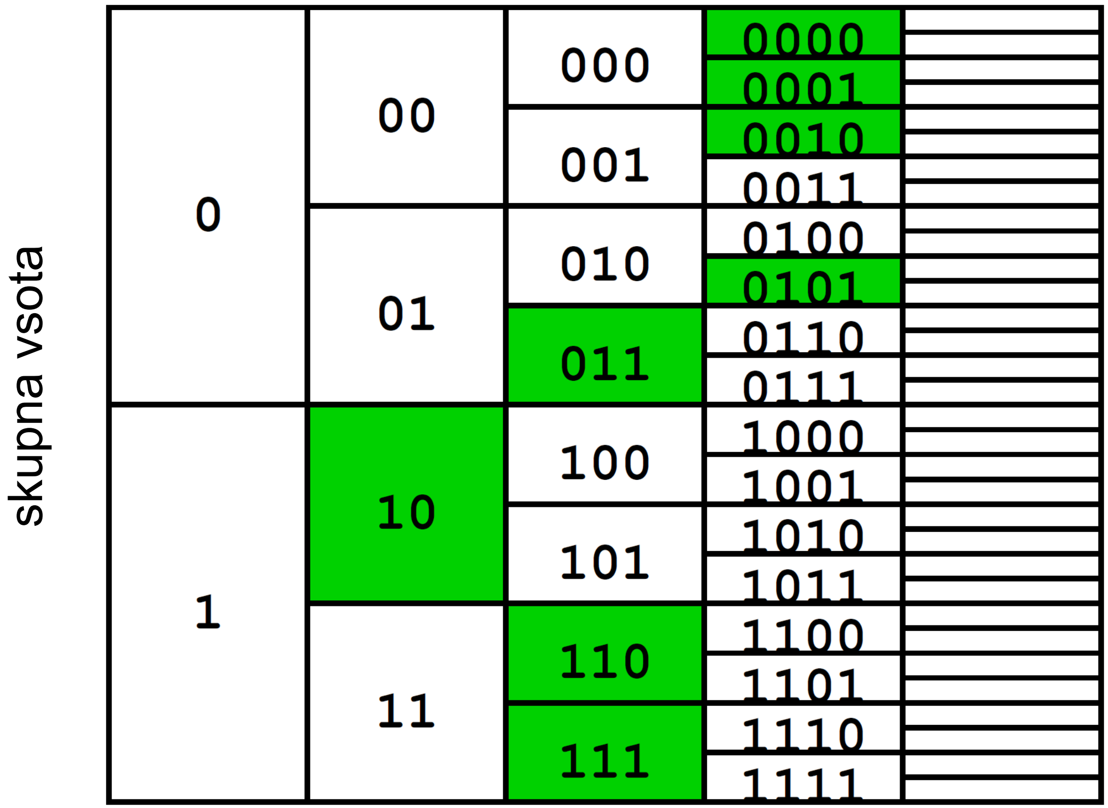
Kraft inequality violated, cannot have unique decoding.

Kraft inequality and coding



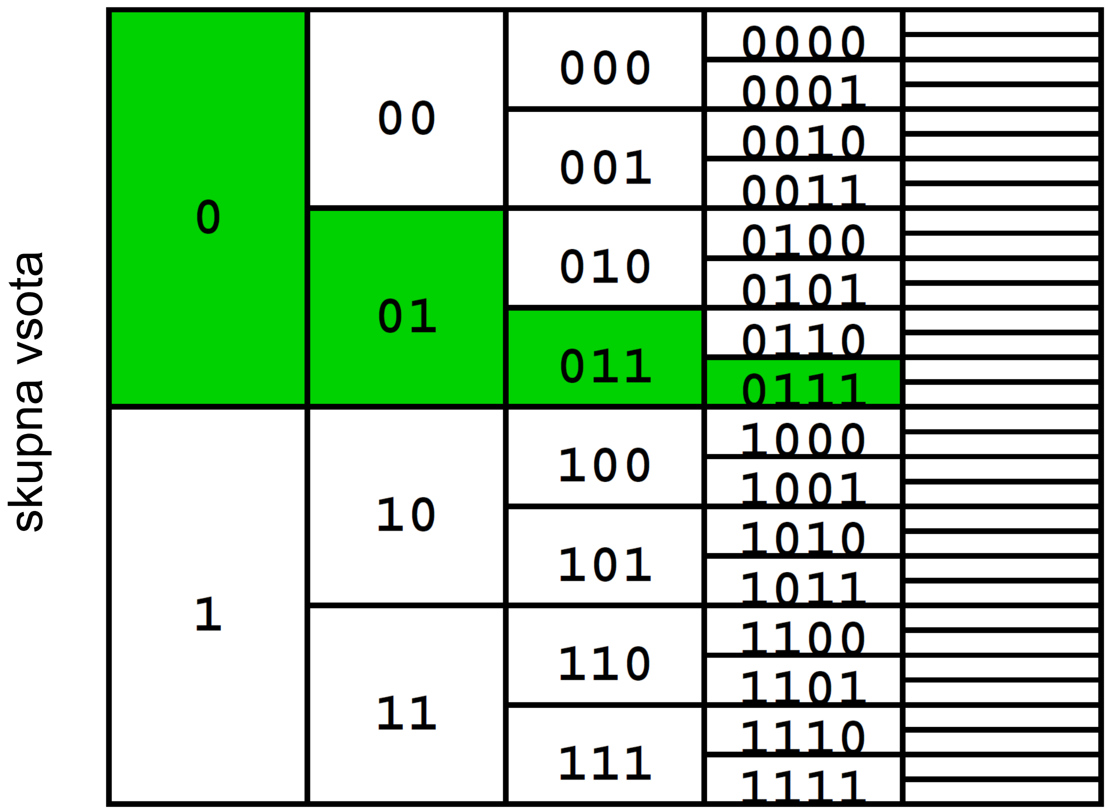
Codewords of equal length

Kraft inequality and coding



Unique decoding possible, prefix-free coding

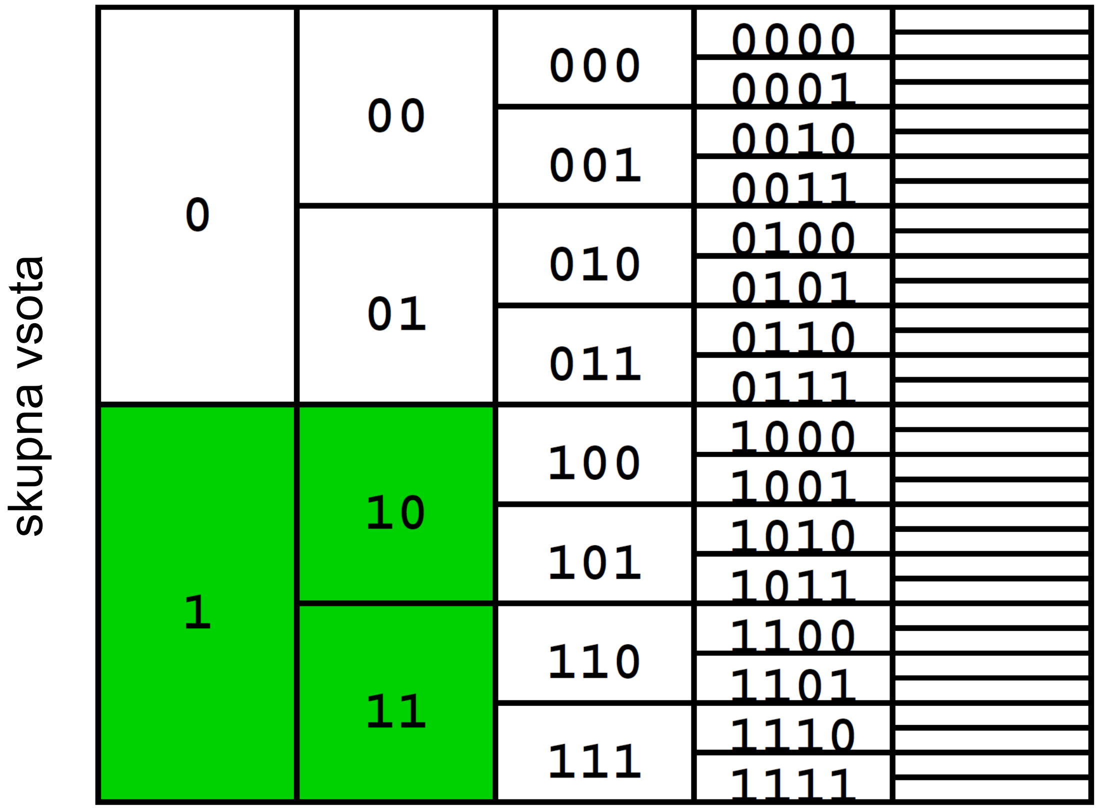
Kraft innequality and coding



Kraft? Unique decoding ?

Prefix-free coding ?

Kraft inequality and coding



Kraft? Unique decoding?

Prefix-free coding ?

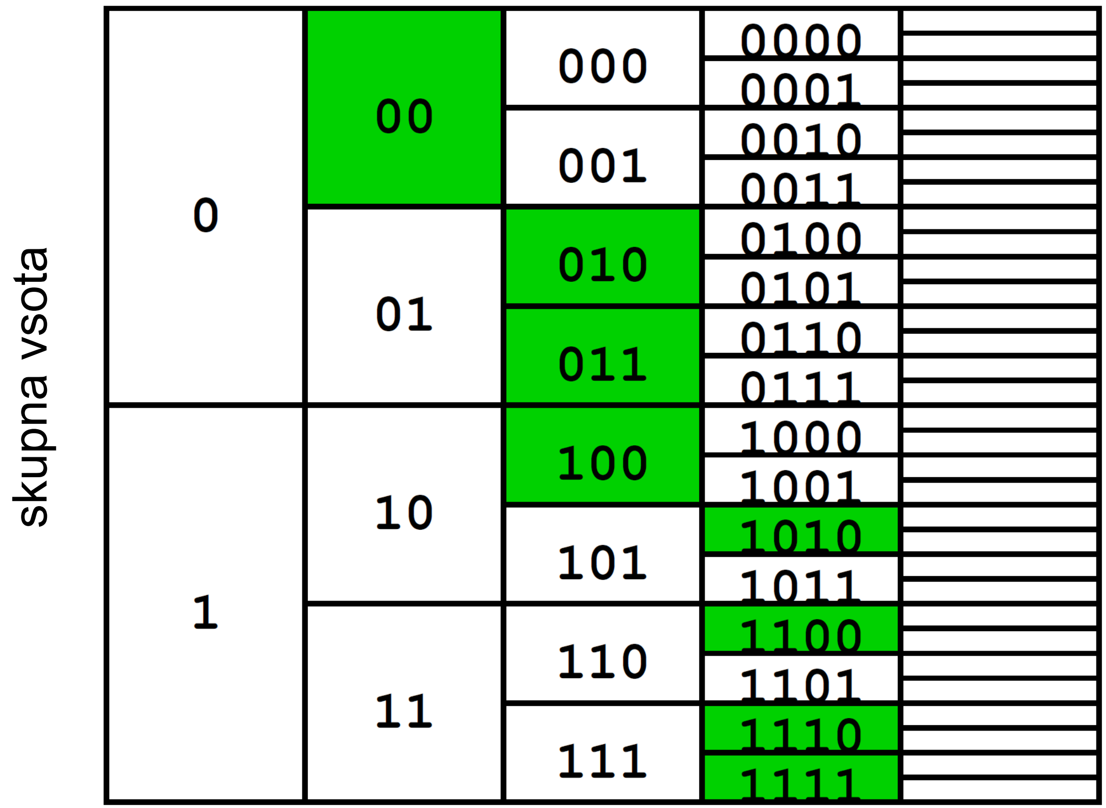
Kraft inequality

- Comment: If in Kraft inequality we have strictly <, thus:

$$\sum_{i=1}^{m} 2^{-\ell_i} < 1$$

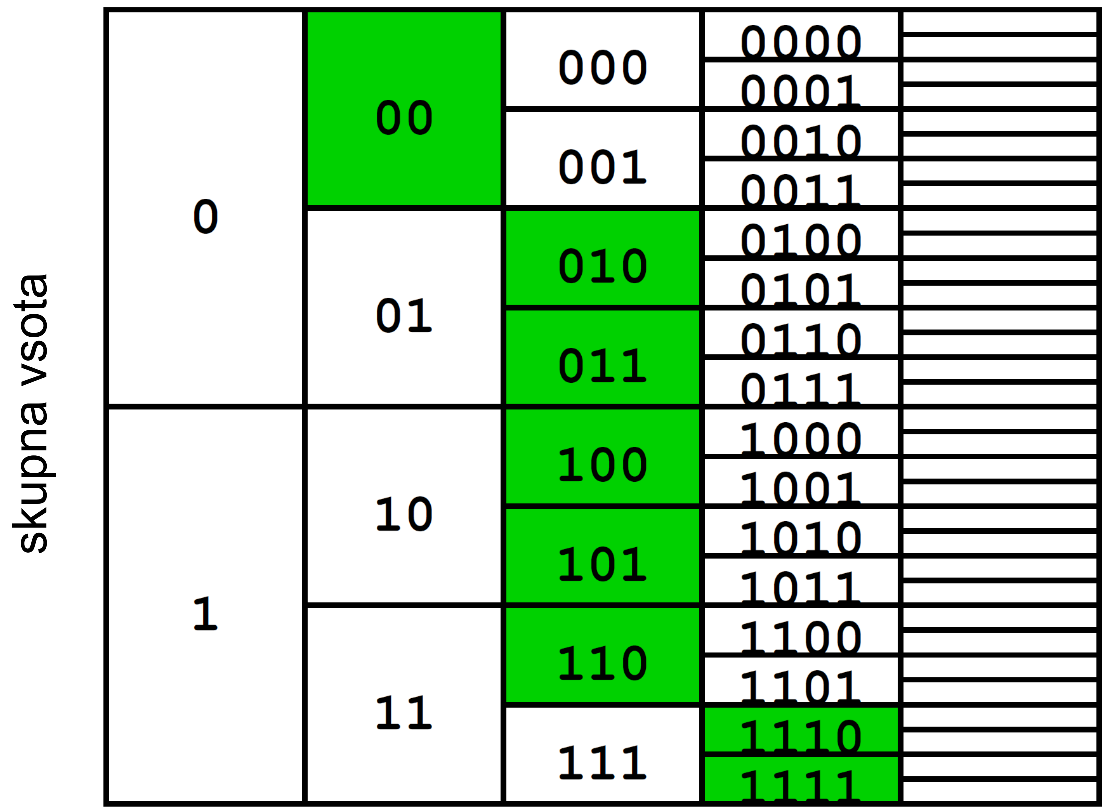
- Then it might be possible to encode with shorter codewords (in average) which we can uniquely decode (prefix-free coding)

Kraft inequality and coding



We can shorten certain codewords.

Kraft inequality and coding

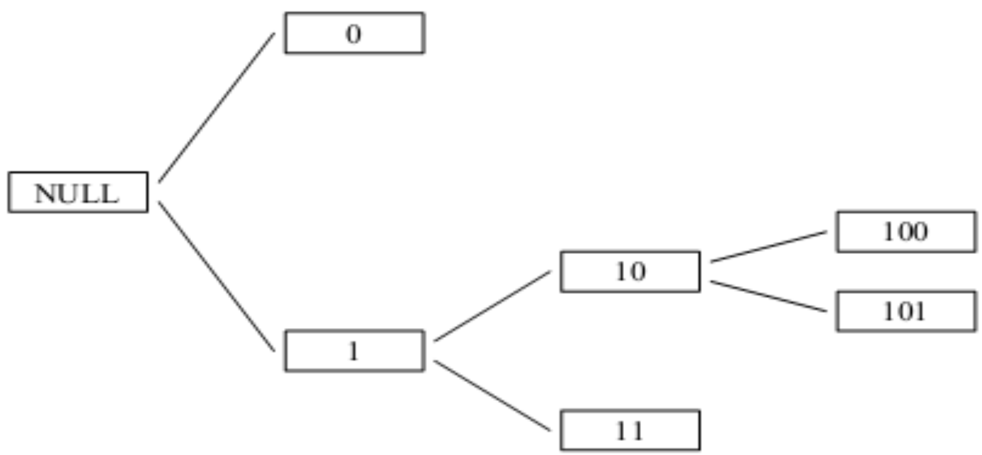


Kraft inequality becomes equality: **complete encoding**

Associating binary tree

We can view codewords of an instantaneous (prefix) code as leaves of a tree. The root represents the null string; each branch corresponds to adding another code symbol.

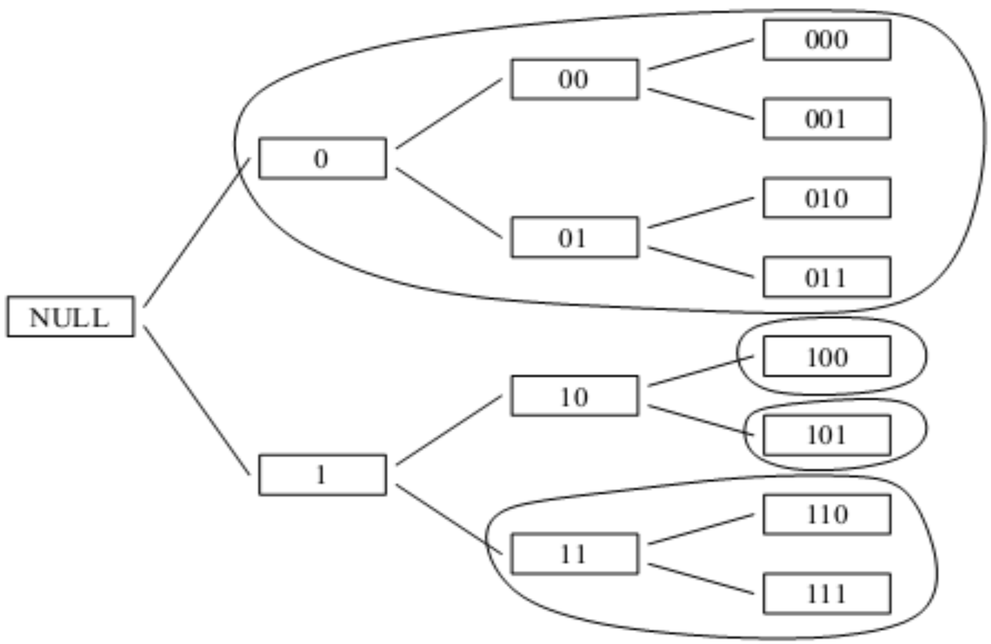
Here is the tree for a code with codewords 0, 11, 100, 101:



Extending to get `the full tree`

We can extend the tree to the depth of the longest codeword. Each codeword then corresponds to a subtree.

Here's the extension of the previous tree, with each codeword's subtree circled:



Proving Kraft inequality

- Assume there exists D-ary prefix-free code with
code lengths $l_1, l_2, \ldots, l_k$ and let $n = max_i\ l_i + 1$.

Construct a full D-ary tree of depth n ! Each time we assign a node (corresponding to a codeword) at depth l_i we cut D^{n-l_i} nodes of the full tree !

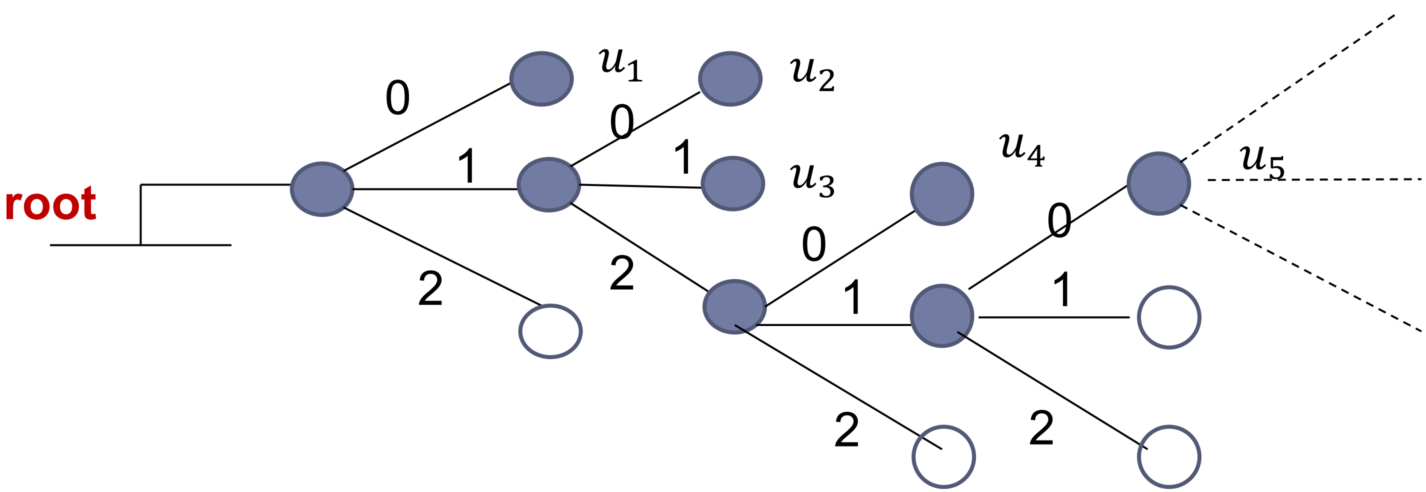
There are only D^n nodes in the tree. Thus,

$$D^{n-l_1} + D^{n-l_2} + \ldots + D^{n-l_k} \leq D^n$$

Divide by D^n and you get Kraft's inequality !!

Kraft inequality – *D*-ary trees

- Alphabet can be ternary or *D*-ary in general !
- **Example** : Construct **ternary** prefix-free code with 5 words of lengths l_1 =1, l_2 =2, l_3 =2, l_4 =3, l_5 =4.
- Then, $\sum_{i=1}^{5} 3^{-l_i} = \frac{1}{3} + \frac{1}{9} + \frac{1}{9} + \frac{1}{27} + \frac{1}{81} = \frac{49}{81} \leq 1$



u_1= 0 ; u_2= 10 ;u_3= 11; u_4= 120 ; u_5= 1210

Kraft – McMillan theorem

- Kraft inequality was about prefix-free coding and the property related to the codewords' length.
- If we consider a broader class of uniquely decodable codes then we have the following result:dekodirati?

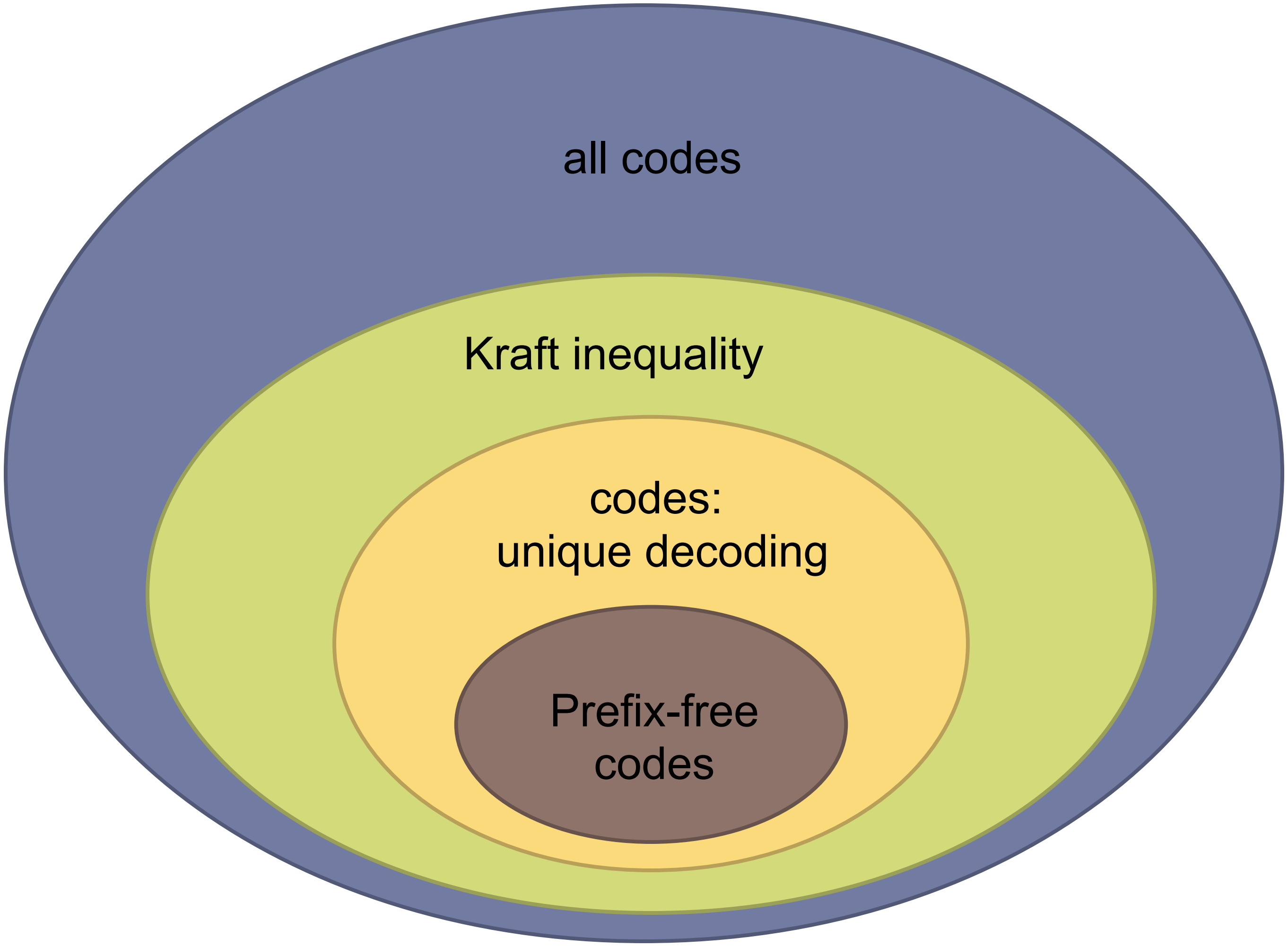
Kraft-McMillan inequality

The lengths $\ell_1, \ldots, \ell_m$ of any (binary) code, which is uniquely decodable, satisfy the following inequality:

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 1.$$

Conversely: If we have the lengths ℓ_i that satisfy the above inequality then there exists a (prefix-free) uniquely decodable whose codewords have these lengths.

Kraft – McMillanov theorem and coding



Optimal symbol coding

- What is optimal ?

- Symbol coding:
  - Shannon coding
  - Shannon-Fano coding
  - Huffman coding

Codewords and probability

- Let $\ell_1, \ldots, \ell_m$ be the lengths of (binary) codewords of a uniquely decodable code $C : \mathcal{X} \to \{0,1\}^*$. The Kraft-McMillan result gives:

$$c = \sum_{i=1}^{m} 2^{-\ell_i} \leq 1.$$

- Let us define probability mass distribution $p : \mathcal{X} \to [0,1]$ as:

$$p_i = \frac{2^{-\ell_i}}{c} \qquad \Leftrightarrow \ell_i = \log_2 \frac{1}{p_i c}.$$

Function p is indeed a valid probability distribution as:

- **non-negativity**: $p(x) \geq 0$ for any $x \in \mathcal{X}$
- sum of all probabilities equals 1

$$\sum_{x \in \mathcal{X}} p(x) = \sum_{i=1}^{m} = \frac{2^{-\ell_i}}{c} = 1$$

Length of codewords and probability

If we have a code whose codewords (lengths) **satisfy Kraft with equality**, thus $c = 1$, then we can compute average length (using $p_i = 2^{-\ell_i}$) as:

$$
\begin{aligned}
E[\ell(X)] &= \sum_{i=1}^{m} 2^{-\ell_i} \ell_i \\
&\sum_{i=1}^{m} p_i \log_2 \frac{1}{p_i} = H(X).
\end{aligned}
$$

This is a **lower bound** on the average codeword length !

Lower bound on average codeword length for uniquely decodable codes

$$E[\ell(X)] \geq H(X).$$

Proof for the lower bound on average length

Spodnja meja povprečne dolžine kod je entropija vira.

$$E[\ell(X)] \geq H(X).$$

Proof:

$$\begin{aligned}
E[\ell(X)] - H(X) &= \sum_{x\in\mathcal{X}} p(x)\,\ell(x) - \sum_{x\in\mathcal{X}} p(x)\,\log_2 \frac{1}{p(x)} \\
&= \sum_{x\in\mathcal{X}} p(x)\,\log_2 \frac{1}{2^{-\ell_x}} - \sum_{x\in\mathcal{X}} p(x)\,\log_2 \frac{1}{p(x)} \\
&= \sum_{x\in\mathcal{X}} p(x)\,\log_2 \frac{p(x)}{2^{-\ell_x}} \\
&= \sum_{x\in\mathcal{X}} p(x)\left[\log_2 \frac{p(x)}{q(x)} + \log_2 \frac{1}{c}\right] \\
&= D(p\,\|\,q) + \log_2 \frac{1}{c} \geq 0\ .
\end{aligned}$$

$$\boxed{q(x) = \frac{2^{-\ell(x)}}{c}}$$

Lower bound for source coding

What have we learnt so far ? For coding of symbols with unique decoding, we have:

Summary

- $E[\ell(X)] - H(X) = D(p||q) + \log_2 \frac{1}{c}$, where $q(x) = \frac{2^{-\ell(x)}}{c}$.
- $E[\ell(X)] \geq H(X)$.
- When $\ell(x) = \log_2 \frac{1}{p(x)}$, then $E[\ell(X)] = H(X)$- **optimal coding**.

Also, when we have I.I.D. random variables $X_1, X_2, \ldots, X_n$ then

$$E[\ell(X_1, \ldots, X_n)] = E[\sum_{i=1}^{n} \ell(X_i)] = \sum_{i=1}^{n} E[\ell(X_i)] = nH(X).$$

Shannon's result states that this is optimal, not only for coding of symbols.

Shannon coding

The problem with $\ell(x) = \log_2 \frac{1}{p(x)}$ is that $\ell(x)$ is not an integer in general (e.g. if $\ell(x) = 2.45$ we cannot assign this length). **Solution is simply** $\ell(x) = \left\lceil \log_2 \frac{1}{p(x)} \right\rceil$.

Shannon coding

Given the probability distribution $p(X)$ we assign:

$$\ell(x) = \left\lceil \log_2 \frac{1}{p(x)} \right\rceil, \quad \forall x \in \mathcal{X}.$$

Shannon's coding: example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$
a	0.0644	3.9	4
b	0.0108	6.5	7
c	0.0178	5.8	6
d	0.0359	4.7	5
e	0.0991	3.3	4
f	0.0147	6.0	7
g	0.0184	5.7	6
h	0.0535	4.2	5
i	0.0551	4.1	5
j	0.0011	9.8	10
k	0.0083	6.8	7
l	0.0343	4.8	5
	⋮		
y	0.0165	5.9	6
z	0.0005	10.7	11
	0.2111	2.2	3

$$H(X) = 4.03$$

- Shannon (1948)
  - Sort probabilities from largest to the smallest.

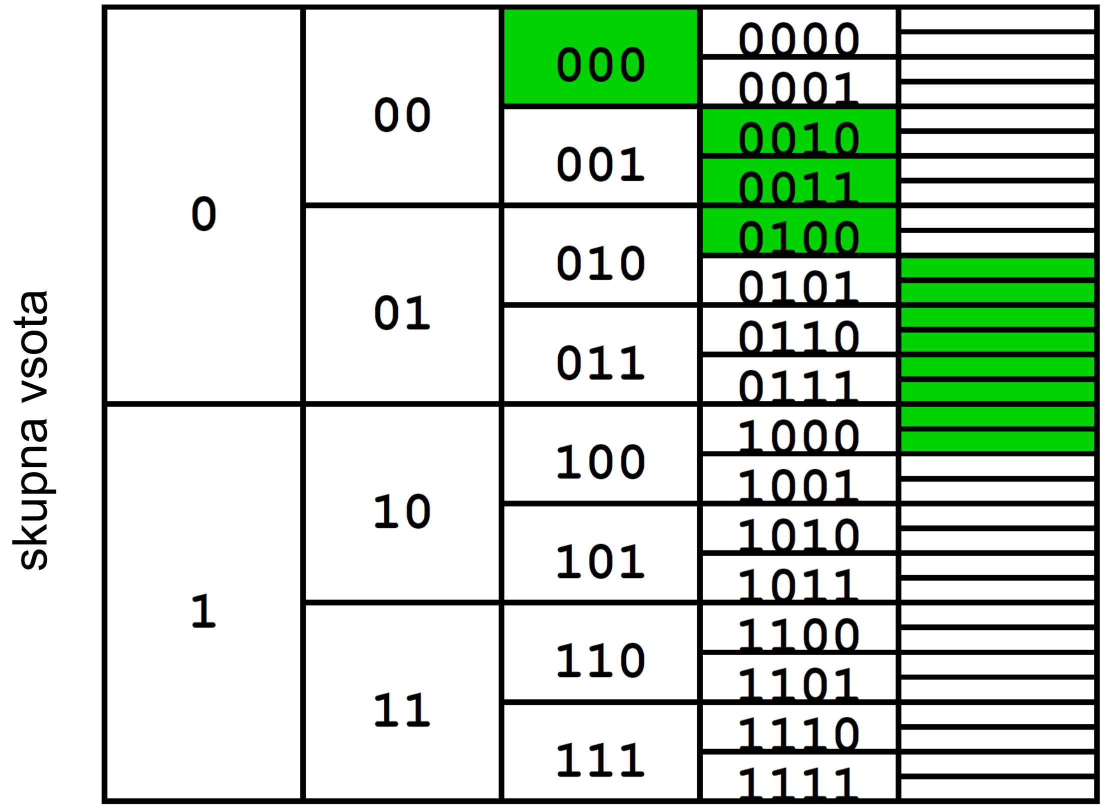
Shannon's coding: example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$
	0.2111	2.2	3
e	0.0991	3.3	4
t	0.0781	3.6	4
a	0.0644	3.9	4
o	0.0598	4.0	5
i	0.0551	4.1	5
h	0.0535	4.2	5
n	0.0516	4.2	5
s	0.0475	4.3	5
r	0.0401	4.6	5
d	0.0359	4.7	5
l	0.0343	4.8	5
	⋮		
x	0.0011	9.8	10
j	0.0011	9.8	10
z	0.0005	10.7	11

$$H(X) = 4.03$$

- Shannon (1948)
  - Sort probabilities from largest to the smallest.
  - Select codewords so that we get a prefix-free code. (Kraft table)

Shannon‘s coding



Codewords length (3, 4, 4, 4, 5, 5, 5, 5, ... , 10, 10, 11)

Shannonovo kodiranje: summary

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
	0.2111	2.2	3	000
e	0.0991	3.3	4	0010
t	0.0781	3.6	4	0011
a	0.0644	3.9	4	0100
o	0.0598	4.0	5	01010
i	0.0551	4.1	5	01011
h	0.0535	4.2	5	01100
n	0.0516	4.2	5	01101
s	0.0475	4.3	5	01110
r	0.0401	4.6	5	01111
d	0.0359	4.7	5	10000
l	0.0343	4.8	5	10001
	⋮			
x	0.0011	9.8	10	1010111101
j	0.0011	9.8	10	1010111110
z	0.0005	10.7	11	10101111110

$$H(X) = 4.03$$
$$E[\ell(X)] = 4.60$$
$$E[\ell(X)] - H(X) = 0.57$$

Shannon coding

- Average length of the codewords is:

$$E\left[\ell(X)\right] = E\left[\left\lceil \log_2 \frac{1}{p(X)} \right\rceil\right] \leq E\left[\log_2 \frac{1}{p(X)} + 1\right] = H(X) + 1$$

- In our example:

$$E[\ell(X)] - H(X) = 4.60 - 4.03 = 0.57 \leq 1$$

Shannon-Fano coding: example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$
a	0.0644	3.9
b	0.0108	6.5
c	0.0178	5.8
d	0.0359	4.7
e	0.0991	3.3
f	0.0147	6.0
g	0.0184	5.7
h	0.0535	4.2
i	0.0551	4.1
j	0.0011	9.8
k	0.0083	6.8
l	0.0343	4.8
	⋮	
y	0.0165	5.9
z	0.0005	10.7
	0.2111	2.2

$$H(X) = 4.03$$

- Shannon-Fano coding
  - Sort probabilities from largest to smallest.

Shannon-Fano coding: example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$
	0.2111	2.2
e	0.0991	3.3
t	0.0781	3.6
a	0.0644	3.9
o	0.0598	4.0
i	0.0551	4.1
h	0.0535	4.2
n	0.0516	4.2
s	0.0475	4.3
r	0.0401	4.6
d	0.0359	4.7
l	0.0343	4.8
	⋮	
x	0.0011	9.8
j	0.0011	9.8
z	0.0005	10.7

- Shannon-Fano coding
  - Sort probabilities from largest to smallest
  - Split into two equal parts approximately enaka.
  - First group of symbols assigns ‘0’, to the other ‘1’.
  - Repeat recursively on each part (group of symbols).

Shannon-Fano coding: example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
	0.2111	2.2	2	0
e	0.0991	3.3	4	0
t	0.0781	3.6	4	0
a	0.0644	3.9	4	0
o	0.0598	4.0	4	0
i	0.0551	4.1	4	1
h	0.0535	4.2	4	1
n	0.0516	4.2	4	1
s	0.0475	4.3	5	1
r	0.0401	4.6	5	1
d	0.0359	4.7	5	1
l	0.0343	4.8	5	1
	⋮			
x	0.0011	9.8	10	1
j	0.0011	9.8	10	1
z	0.0005	10.7	10	1

Shannon-Fano coding: example

	X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
		0.2111	2.2	2	00
	e	0.0991	3.3	4	01
	t	0.0781	3.6	4	01
	a	0.0644	3.9	4	01
	o	0.0598	4.0	4	01
	i	0.0551	4.1	4	10
	h	0.0535	4.2	4	10
	n	0.0516	4.2	4	10
	s	0.0475	4.3	5	10
	r	0.0401	4.6	5	10
	d	0.0359	4.7	5	11
	l	0.0343	4.8	5	11
		⋮			
	x	0.0011	9.8	10	11
	j	0.0011	9.8	10	11
	z	0.0005	10.7	10	11

Shannon-Fano coding: example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
	0.2111	2.2	2	00
e	0.0991	3.3	4	010
t	0.0781	3.6	4	010
a	0.0644	3.9	4	011
o	0.0598	4.0	4	011
i	0.0551	4.1	4	100
h	0.0535	4.2	4	100
n	0.0516	4.2	4	101
s	0.0475	4.3	5	101
r	0.0401	4.6	5	101
d	0.0359	4.7	5	110
l	0.0343	4.8	5	110
	⋮			
x	0.0011	9.8	10	111
j	0.0011	9.8	10	111
z	0.0005	10.7	10	111

Shannon-Fano coding: example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
	0.2111	2.2	2	00
e	0.0991	3.3	4	0100
t	0.0781	3.6	4	0101
a	0.0644	3.9	4	0110
o	0.0598	4.0	4	0111
i	0.0551	4.1	4	1000
h	0.0535	4.2	4	1001
n	0.0516	4.2	4	1010
s	0.0475	4.3	5	1011
r	0.0401	4.6	5	1011
d	0.0359	4.7	5	1100
l	0.0343	4.8	5	1100
	⋮			
x	0.0011	9.8	10	1111
j	0.0011	9.8	10	1111
z	0.0005	10.7	10	1111

Shannon-Fano coding: example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
	0.2111	2.2	2	00
e	0.0991	3.3	4	0100
t	0.0781	3.6	4	0101
a	0.0644	3.9	4	0110
o	0.0598	4.0	4	0111
i	0.0551	4.1	4	1000
h	0.0535	4.2	4	1001
n	0.0516	4.2	4	1010
s	0.0475	4.3	5	10110
r	0.0401	4.6	5	10111
d	0.0359	4.7	5	11000
l	0.0343	4.8	5	11001
	⋮			
x	0.0011	9.8	10	11111
j	0.0011	9.8	10	11111
z	0.0005	10.7	10	11111

Shannon-Fano coding: example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
	0.2111	2.2	2	00
e	0.0991	3.3	4	0100
t	0.0781	3.6	4	0101
a	0.0644	3.9	4	0110
o	0.0598	4.0	4	0111
i	0.0551	4.1	4	1000
h	0.0535	4.2	4	1001
n	0.0516	4.2	4	1010
s	0.0475	4.3	5	10110
r	0.0401	4.6	5	10111
d	0.0359	4.7	5	11000
l	0.0343	4.8	5	11001
	⋮			
x	0.0011	9.8	10	111111
j	0.0011	9.8	10	111111
z	0.0005	10.7	10	111111

Shannon-Fano coding: example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
	0.2111	2.2	2	00
e	0.0991	3.3	4	0100
t	0.0781	3.6	4	0101
a	0.0644	3.9	4	0110
o	0.0598	4.0	4	0111
i	0.0551	4.1	4	1000
h	0.0535	4.2	4	1001
n	0.0516	4.2	4	1010
s	0.0475	4.3	5	10110
r	0.0401	4.6	5	10111
d	0.0359	4.7	5	11000
l	0.0343	4.8	5	11001
	⋮			
x	0.0011	9.8	10	1111111101
j	0.0011	9.8	10	1111111110
z	0.0005	10.7	10	1111111111

$$H(X) = 4.03$$

$$E[\ell(X)] = 4.07$$

$$E[\ell(X)] - H(X) = 0.04$$

Shannon - Fano coding

- Expected codeword lengths for Shannon-Fano coding is:

$$E[\ell(X)] \leq H(X) + 1 - p_{min},$$

where p_{min} is the smallest probability over alphabet letters.

- In our example:

$$E[\ell(X)] - H(X) = 4.06 - 4.03 = 0.03 \leq 1 - 0.0005.$$

- Is this optimal ? NO. Huffman coding.

Shannon result on coding of symbols

Shannon result on coding of symbols

For any alphabet $\mathcal{X}$, associated with a RV X, there exists prefix-free coding $C : \mathcal{X}^* \to \{0,1\}^*$ such that the average code length satisfies

$$H(X) \leq E[\ell(X)] \leq H(X) + 1.$$

Huffman coding

- Huffman coding algorithm:

1. Sort symbols w.r.t. Probability p_i

2. Merge two symbols with smallest probability, say *i* and *j*, and remove these from the set. Add a **new pseudosymbol** *z* whose probability is $p_i + p_j$.

3. If there are more than 1 symbols left, go to step 1.

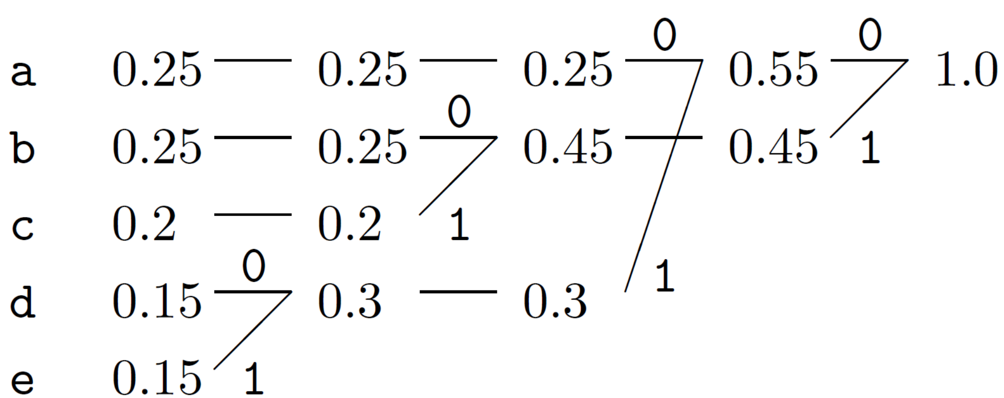
4. Result is a binary tree, which we use for coding.

Huffman coding : small example

- We have a set of symbols $\mathcal{X} = \{a, b, c, d, e\}$

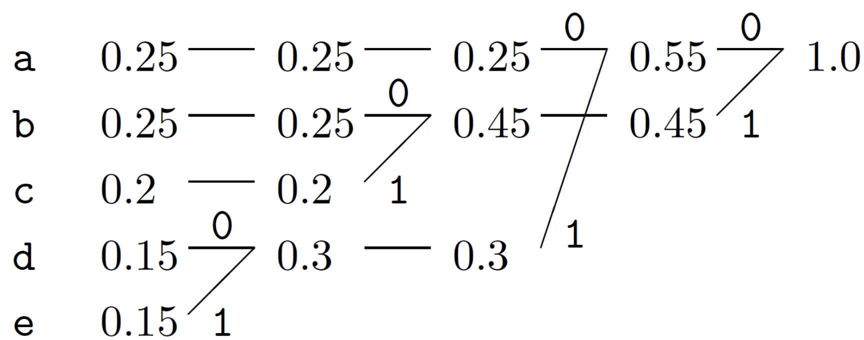
  with respective probabilities $p(x) = \{0.25, 0.25, 0.2, 0.15, 0.15\}$

- We apply Huffman coding.



Huffman coding : small example

- We apply Huffman coding.



- Codewords are the bits (from the root) in the binary tree

a_i	p_i	$h(p_i)$	l_i	$c(a_i)$
a	0.25	2.0	2	00
b	0.25	2.0	2	10
c	0.2	2.3	2	11
d	0.15	2.7	3	010
e	0.15	2.7	3	011

$$E[\ell(X)] = 2.30$$
$$- \quad H(X) = 2.2855$$
$$= \quad 0.0145$$

Huffman coding: coding of english alphabet

Demo:

http://www.cs.auckland.ac.nz/software/AlgAnim/huffman.html

Huffman coding: optimality

- What does optimality means ?
  - Shortest average code length among any possible coding of symbols.

- **Huffman coding is optimal.**
- <u>Proof (sketch):</u>
  - Step 2, to assign two symbols with smallest probability the same length of codeword (one ends with 0 and other with 1), leads to coding with shortest average code length. Alternatively, there is no other coding with shorter average codeword length.
  - Proof uses contradiction, MacKay page 105.

symbol	probability	Huffman code	alternative codes	Modifed alternative code with shorter length
a	p_a	$c_{\mathrm{H}}(a)$	$c_{\mathrm{R}}(a)$	$c_{\mathrm{R}}(c)$
b	p_b	$c_{\mathrm{H}}(b)$	$c_{\mathrm{R}}(b)$	$c_{\mathrm{R}}(b)$
c	p_c	$c_{\mathrm{H}}(c)$	$c_{\mathrm{R}}(c)$	$c_{\mathrm{R}}(a)$

Remaining problems with symbol coding

- We have shown that the average length of codewords satisfies the inequality $E[\ell(X)] \leq H(X) + 1$. Is it enough ?

- NO. There are 3 problems:

1. One extra bit, *H(X) + 1*.
    - Not a problem, if H(X) is large, BUT if H(X) is small (e.g. 1 bit) it is significant.

2. With Shannon-Fano and Huffman coding **one needs to know the probability distribution of alphabet letters**.
    - This distribution can be determined form the data we want to encode (not always).
    - What about decoding?

3. Assumption about independency about random variables and the same distribution is not realistic in most of the cases
    - Example: Natural languages

Apriori knowledge about probability distribution

- With Shannon-Fano and Huffman coding **one needs to know the probability distribution of alphabet letters**.
  - This distribution can be determined form the data we want to encode (not always).
  - What about decoding ?

- Solution:
  - After encoding source sybmols (database) we send (save) the used probability distribution of the symbols.
  - It increases the total length of compressed data, but if the amount of compressed data is large it does not affect much.

Solutions to 1st and 3rd problem

- 1. One extra bit, $H(X) + 1$.
- 3. Assumption about independency about random variables and the same distribution

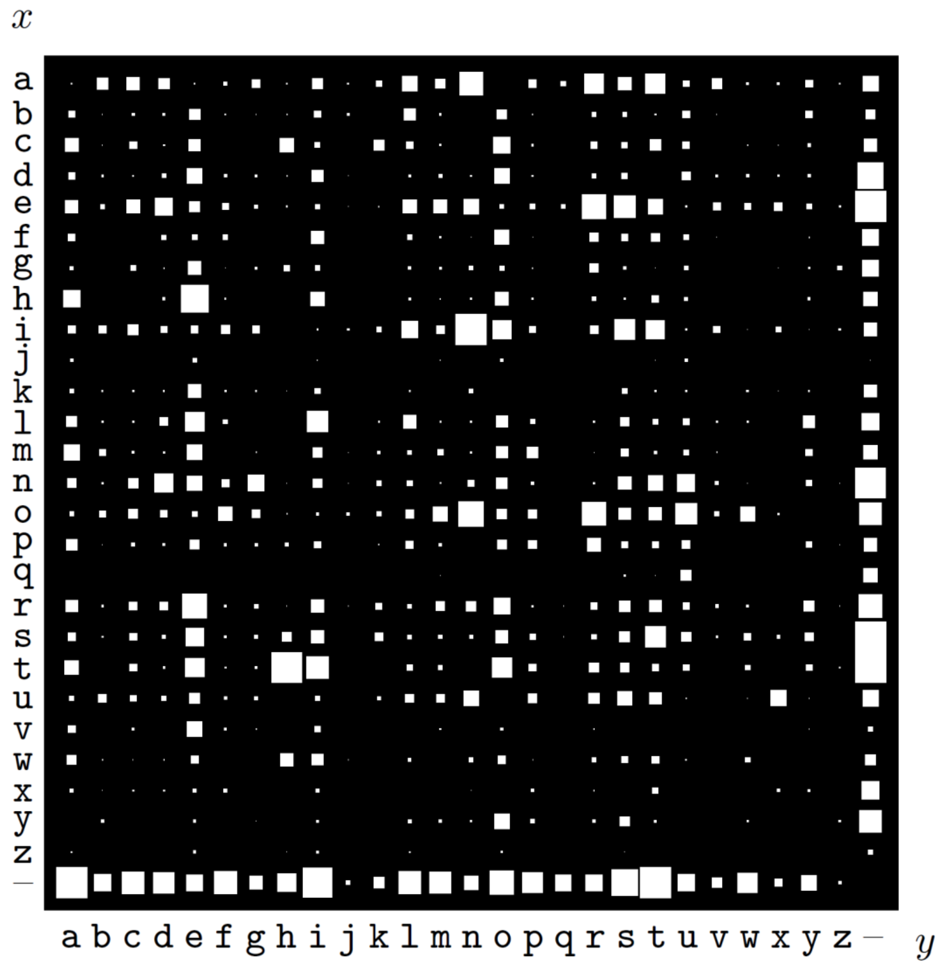
Solution:

- **Block codes**

  Merge (several) symbols in blocks and blocks become new symbols.

  - Entropy increases and one extra bit (at most) becomes insignificant.
  - We can model dependency between the symbols.

Example of block codes: bigram model for English alphabet



Another possibility: Arithmetic coding

- First one game:

- Guess unknown sentence by suggesting the missing letters.

_ .

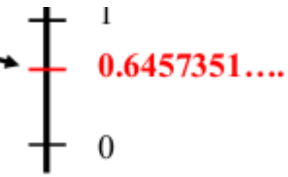
- **IDEA:** You maybe need 10 guesses for the (correct) first letter, maybe 6 for the second etc. If decoder guesses in the exactly same way there is a possibility of using such approach for **stream codes**

Arithmetic coding

- Coding resembles the previous game.
- Instead of „guesser“ we use probabilty distribution of the source alphabet we want to encode.
- **Basic idea**:
  Sequence of symbols is represented as subinterval [a, b) of the interval [0, 1), thus $[a, b) \subseteq [0, 1)$
- Since we cannot represent arbitrary real numbers *a, b* with finite number of bits, we take some $x \in [a, b)$ of shortest binary length
- This number x we represent as a binary string predstavimo, e.g. x = 0.0110100:
  - Large interval implies shorter binary string.
  - Small interval implies longer binary string.

Main idea of arithmetic coding

Sequence: $S_1\ S_2\ S_3\ S_4 \ldots$ Mapped to…

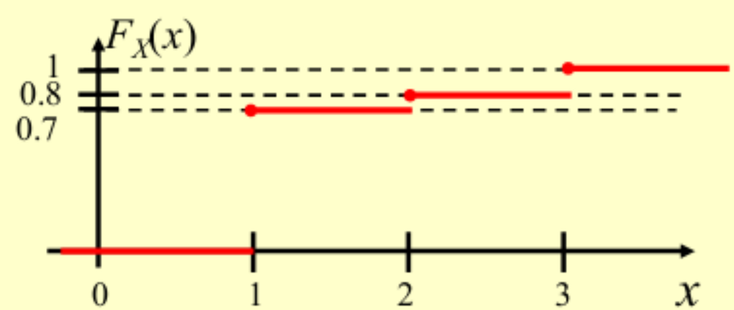


Each possible sequence gets mapped to a unique number in [0,1)

- The mapping depends on the prob. of the symbols
- You don't need to *a priori* determine all possible mappings
  - The Mapping is "built" as each new symbol arrives

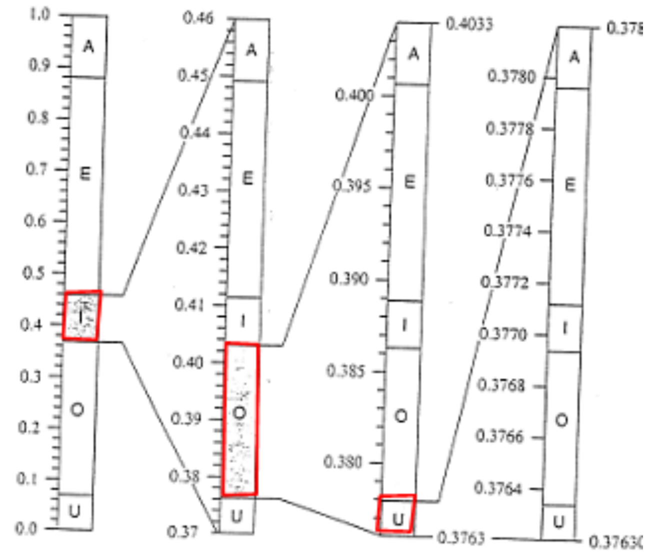
Recall **<u>CDF of an RV</u>**: symbols $\{a_1, a_2, a_3\}$ → RV X w/ values: {1, 2, 3}

Consider $P(X=1) = 0.7$ $P(X=2) = 0.1$ $P(X=1) = 0.2$



Example - Vowellish

Symbols	Probabilities	Optimal # Bits log2(1/Pi)
a	0.12	3.06
e	0.42	1.25
I	0.09	3.47
o	0.3	1.74
u	0.07	3.84



<u>To send "iou"</u>: Send any # C such that

$0.37630 \le C < 0.37819$

Using Binary Fraction of

0.011000001 (9 bits)

9 bits for Arithmetic

vs

10 bits for Huffman

As each symbol is processed find new $\begin{Bmatrix} \text{upper limit} \\ \text{lower limit} \end{Bmatrix}$ for interval

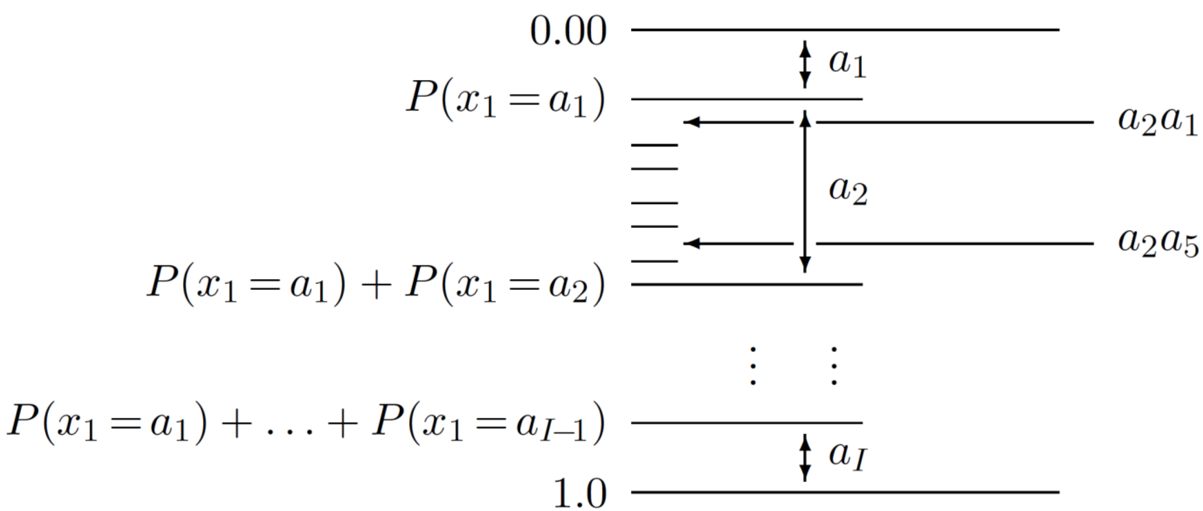
Arithmetic coding: assumptions

- We have an alphabet $\mathcal{A}_X = \{a_1, a_2, \dots, a_I\}$ where a_I is a special symbol which denotes *end of transmission*.
- We have a source, generating symbols $x_1, x_2, \dots, x_n, \dots$ from $\mathcal{A}_X$, not necessarily mutually independent.
- We assume that, both with encoding and decoding, we know the probability of generating $x_n = a_i$ given the knowledge of $x_1, \dots, x_{n-1}$, thus

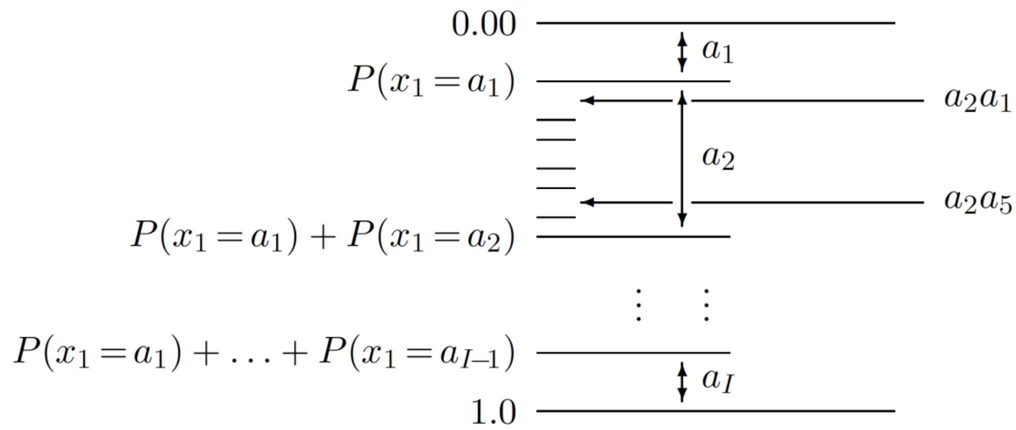
$$p(x_n = a_i | x_1, \dots, x_{n-1}).$$

Arithmetic coding

- The given probabilities a_i are represented as subintervals of interval [0,1) as follows: $P(x_1 = a_i)$
  - First probabilities



Arithmetic coding



- We then use the probabilities $P(x_2 = a_j | x_1 = a_i)$ to split each interval a_i into $a_ia_1, a_ia_2, \ldots, a_ia_I$ where the length of subintervals is directly proportional to $P(x_2 = a_j | x_1 = a_i)$. The length of a_ia_j equals to:

$$P(x_1 = a_i, x_2 = a_j) = P(x_1 = a_i)P(x_2 = a_j | x_1 = a_i).$$

- If we repeat this process we can split the interval $[0, 1)$ to correspond to any possible sequence $x_1, x_2, \ldots, x_n$. The interval length corresponds to the probability $P(x_1, x_2, \ldots, x_n)$.

Arith. coding – computing conditional (pogojnih) probabilities and new intervals

Postopek priredbe vezanih (pogojnih) verjetnosti intervalu

$u = 0.0$
$v = 1.0$
$p = v - u$
for $n = 1$ **to** N **do**
  Izračunamo kumulativne verjetnosti Q_n in R_n.
  $v = u + pR_n(x_n|x_1, \ldots, x_{n-1})$
  $u = v + pQ_n(x_n|x_1, \ldots, x_{n-1})$
  $p = v - u$
end for

$$Q_n(a_i \,|\, x_1, \ldots, x_{n-1}) \equiv \sum_{i'=1}^{i-1} P(x_n = a_{i'} \,|\, x_1, \ldots, x_{n-1}),$$

$$R_n(a_i \,|\, x_1, \ldots, x_{n-1}) \equiv \sum_{i'=1}^{i} P(x_n = a_{i'} \,|\, x_1, \ldots, x_{n-1}).$$

Programming interval updates

Consider a sequence of RVs $X = (x_1, x_2, x_3, \ldots, x_n)$ corresponding to the sequence of symbols $(S_1, S_2, S_3, \ldots, S_n)$

Ex.

$$\text{Alphabet} = \{a_1, a_2, a_3\} \rightarrow \text{RV Values } \{1, 2, 3\}$$

$$(S_1\ S_2\ S_3\ S_4) = (a_2\ a_3\ a_3\ a_1) \rightarrow (x_1\ x_2\ x_3\ x_4) = (2\ 3\ 3\ 1)$$

Initial Values:

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

Interval Update:

$$l^{(n)} = l^{(n-1)} + \left[u^{(n-1)} - l^{(n-1)}\right] F_X(x_n - 1)$$

$$u^{(n)} = l^{(n-1)} + \left[u^{(n-1)} - l^{(n-1)}\right] F_X(x_n)$$

From Prev Interval — From Prob Model

Some properties of the update

- What is the smallest $l^{(n)}$ can be?

$$l^{(n)} = l^{(n-1)} + \underbrace{\left[u^{(n-1)} - l^{(n-1)}\right]}_{>0} \underbrace{F_X(x_n - 1)}_{\geq 0} \Rightarrow l^{(n)} \geq l^{(n-1)}$$

- What is the largest $u^{(n)}$ can be?

$$u^{(n)} = l^{(n-1)} + \left[u^{(n-1)} - l^{(n-1)}\right] \underbrace{F_X(x_n)}_{\leq 1}$$

$$\leq l^{(n-1)} + \left[u^{(n-1)} - l^{(n-1)}\right] \Rightarrow u^{(n)} \leq u^{(n-1)}$$

These imply an important requirement for decoding:

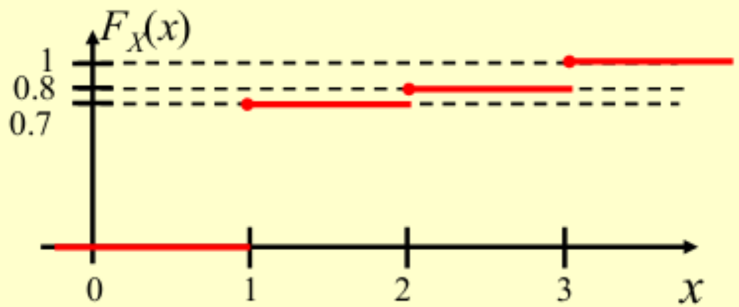
New Interval $\subseteq$ Old Interval

Example of interval update

Symbols $\{a_1, a_2, a_3\}$ ➔ RV X w/ values: {1, 2, 3}

Consider $P(X=1) = 0.7$ $P(X=2) = 0.1$ $P(X=1) = 0.2$

CDF for this alphabet/RV



Consider the sequence $(a_1\ a_3\ a_2) \rightarrow (1\ 3\ 2)$

To process the first symbol "1"

$F_X(0)$

$$l^{(1)} = l^{(0)} + \left[u^{(0)} - l^{(0)}\right] F_X(1-1) = 0 + [1-0] \times 0 \quad = 0$$

$$u^{(1)} = l^{(0)} + \left[u^{(0)} - l^{(0)}\right] F_X(1) \quad = 0 + [1-0] \times 0.7 = 0.7$$

$F_X(1)$

Example cont.

To process the 2nd symbol "3"

$$l^{(2)} = l^{(1)} + \left[u^{(1)} - l^{(1)}\right] F_X(3-1) = 0 + [0.7-0] \times 0.8 = 0.56$$

$$u^{(2)} = l^{(1)} + \left[u^{(1)} - l^{(1)}\right] F_X(3) \quad = 0 + [0.7-0] \times 1 \quad = 0.7$$

($0.8 = F_X(2)$, $1 = F_X(3)$)

To process the 3rd symbol "2"

1 0 1 0 1 0 1 0 0 0

$$l^{(3)} = l^{(2)} + \left[u^{(2)} - l^{(2)}\right] F_X(2-1) = 0.56 + [0.7-0.56] \times 0.7 = 0.658$$

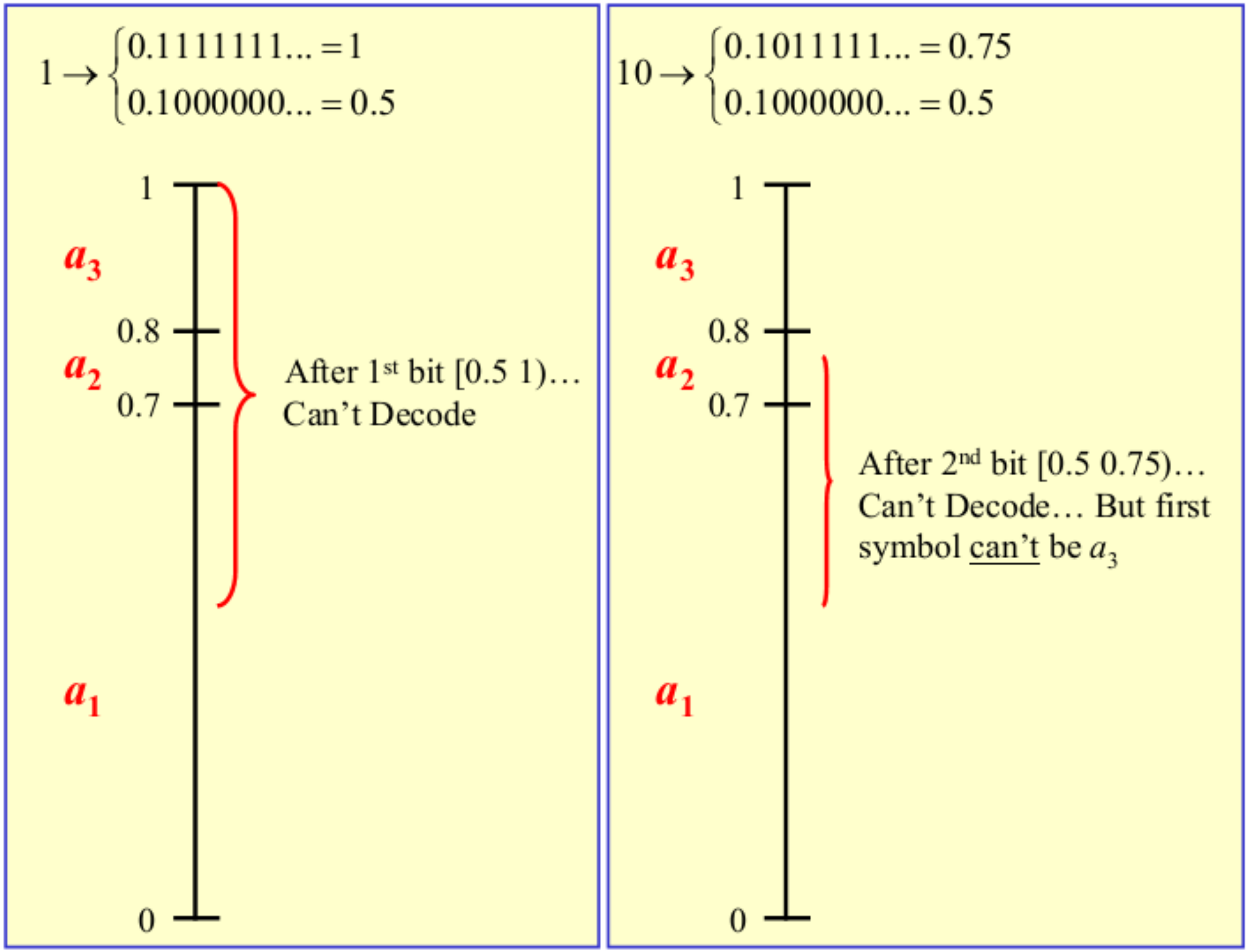
$$u^{(3)} = l^{(2)} + \left[u^{(2)} - l^{(2)}\right] F_X(2) \quad = 0.56 + [0.7-0.56] \times 0.8 = 0.672$$

($0.7 = F_X(1)$, $0.8 = F_X(2)$)

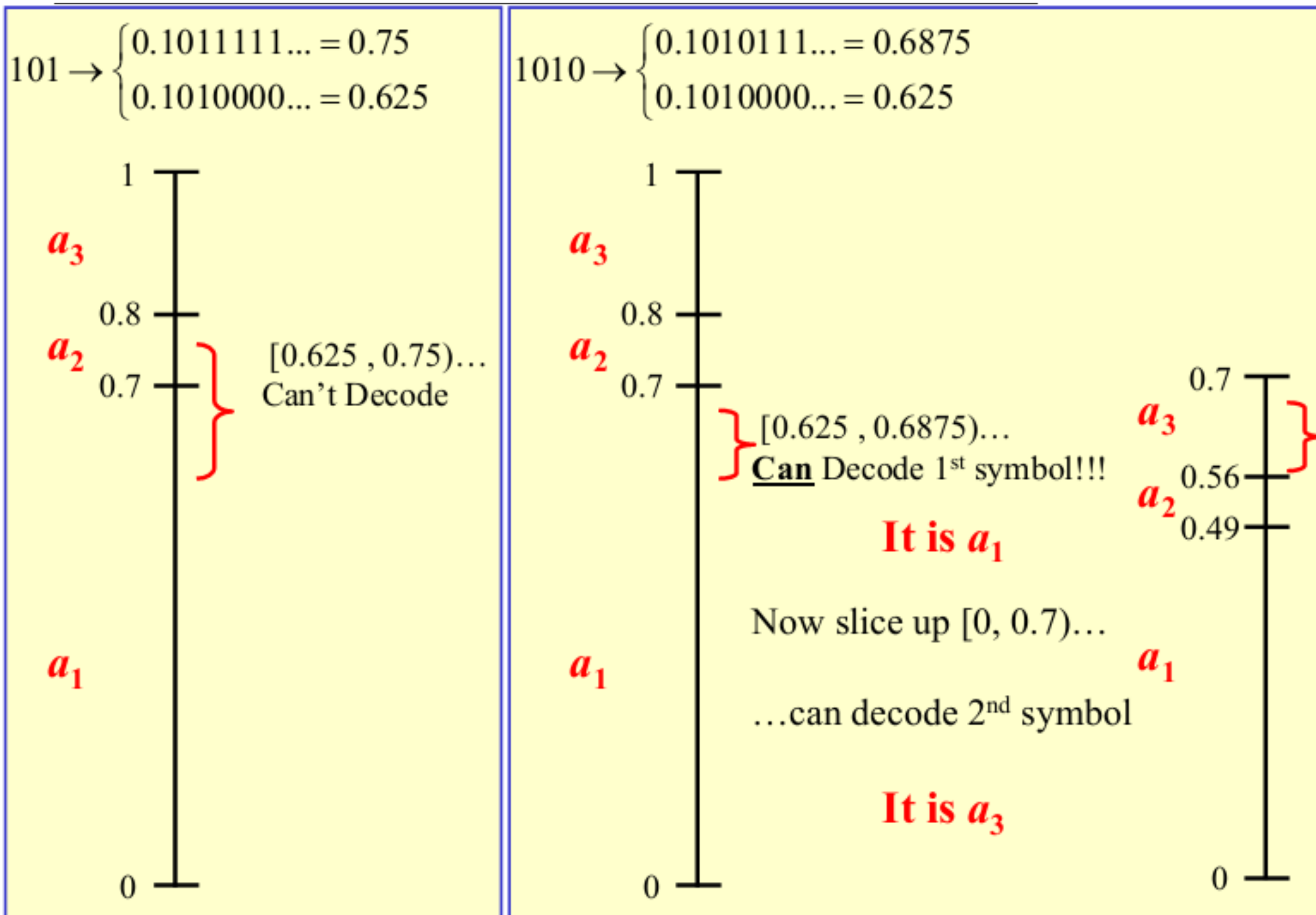
So… send a number in the interval [0.658,0.672) Pick 0.6640625

$0.6640625_{10} = 0.1010101_2$ **Code = 1 0 1 0 1 0 1**

Decoding the received codeword

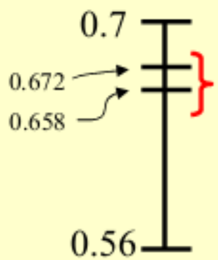


Decoding cont.



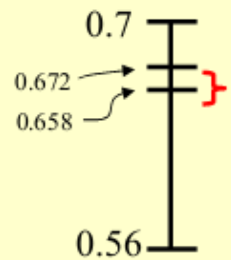
Decoding cont.

$$10101 \rightarrow \begin{cases} 0.1010111\ldots = 0.6875 \\ 0.1010100\ldots = 0.65625 \end{cases}$$



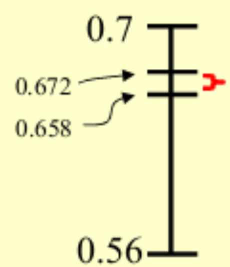
[0.65625 , 0.6875)…
Can't Decode

$$101010 \rightarrow \begin{cases} 0.10101011\ldots = 0.671875 \\ 0.10101000\ldots = 0.65625 \end{cases}$$



[0.65625, 0.671875)…
Can't Decode

$$1010101 \rightarrow \begin{cases} 0.1010101111\ldots = 0.671875 \\ 0.1010101000\ldots = 0.6640625 \end{cases}$$



[0.6640625, 0.671875)
Can Decode 3rd symbol!!!

It is a_2

In practice there are ways to handle termination issues!

Comparison to Huffman

1. A "binary tag" lying between $l^{(n)}$ and $u^{(n)}$ can be found
2. The tag can be truncated to a finite # of bits
3. The truncated tag still lies between $l^{(n)}$ and $u^{(n)}$
4. The truncated tag is Unique & Decodable
5. For IID sequence of length *m:*

$$H(S) \leq \bar{l}_{Arith} < H(S) + \frac{2}{m}$$

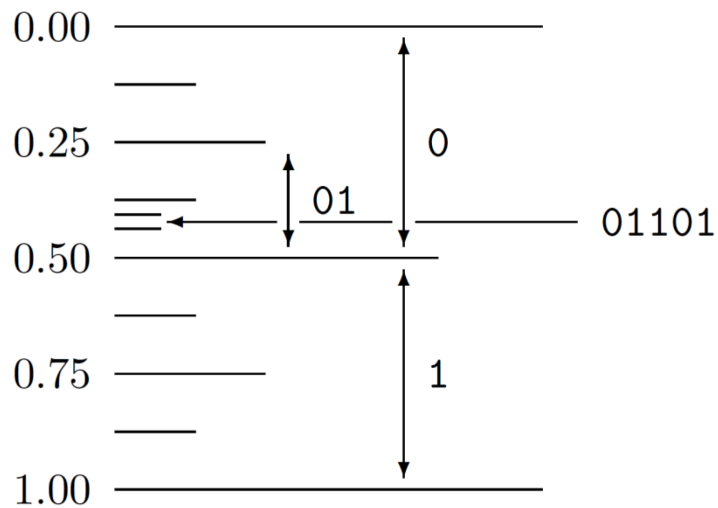
Compared to Huffman:

$$H(S) \leq \bar{l}_{Huff} < H(S) + \frac{1}{m}$$

Hey! AC is worse than Huffman??!! So why consider AC???!!!
Remember, this is for coding the entire length of *m* symbols…
You'd need 2^m codewords in Huffman… which is impractical!
But for AC is VERY practical!!!

For Huffman must be kept small but for AC it can be VERY large!

Arithmetic coding: representing real numbers in [0,1) with binary strings



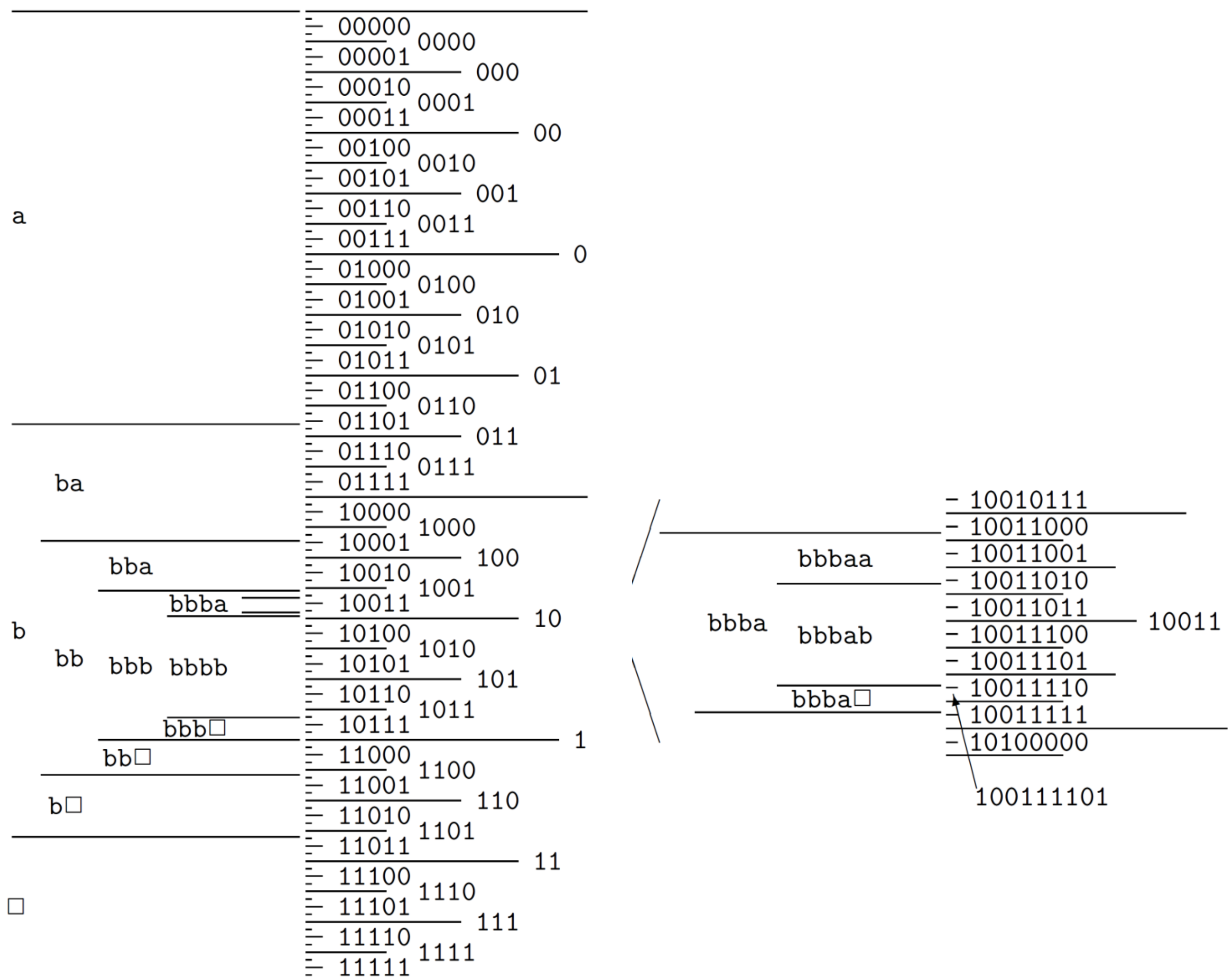
- **Example**: binary string 01 is the shortest representation of 0.01..., which lies in the interval [0.25, 0.50) of real numbers.

Arithmetic coding: example

- We want to encode sequence of outcomes when tossing unfair coin. There are 2 possibilities 'a' (tail) in 'b' (head), and the symbol to denote the end of experiment '□'.
- Let us encode 'bbba□'.
- **Coding is performed at the same time as symbols arrive.**
- We know conditional probabilities (assumption) of possible outcomes:

	$P(\mathtt{a}) = 0.425$	$P(\mathtt{b}) = 0.425$	$P(\square) = 0.15$
b	$P(\mathtt{a} \mid \mathtt{b}) = 0.28$	$P(\mathtt{b} \mid \mathtt{b}) = 0.57$	$P(\square \mid \mathtt{b}) = 0.15$
bb	$P(\mathtt{a} \mid \mathtt{bb}) = 0.21$	$P(\mathtt{b} \mid \mathtt{bb}) = 0.64$	$P(\square \mid \mathtt{bb}) = 0.15$
bbb	$P(\mathtt{a} \mid \mathtt{bbb}) = 0.17$	$P(\mathtt{b} \mid \mathtt{bbb}) = 0.68$	$P(\square \mid \mathtt{bbb}) = 0.15$
bbba	$P(\mathtt{a} \mid \mathtt{bbba}) = 0.28$	$P(\mathtt{b} \mid \mathtt{bbba}) = 0.57$	$P(\square \mid \mathtt{bbba}) = 0.15$

Arithmetic coding: example



Arithmetic coding: decoding

We decode the binary string '1001111101'. The bits are arriving continuously and we need to decode them in real-time. At decoding we know the probabilities, as for the encoding. From the probability model we first find $P(a)$, $P(b)$ and $P(\Box)$, thus knowing the intervals for 'a', 'b' and '$\Box$'.

After we've received the first bit '1' we do not know whether the symbol was 'b'' or '$\Box$'. After receiving '0' as the second bit we know that the first symbol was 'b''.

Decoder then reads off the probabilities $P(a|b$, $P(b|b)$ and $P(\Box|b)$ and compute the intervals 'ba', 'bb' and '$b\Box$'. Similarly as before, we decode next symbol 'b' when we receive '1001', the third symbol b when we receive '100111' etc. until we decode '$bba\Box$.

First order prob. models

Suppose you have three symbols and you have a 1st order conditional probability model for the source emitting these symbols…

For the first symbol in the sequence you have a std Prob Model

For subsequent symbols in the sequence you have 3 context models

$$P(a_1) = 0.2 \quad P(a_2) = 0.4 \quad P(a_3) = 0.4 \quad \sum_i P(a_i) = 1$$

$$P(a_1 | a_1) = 0.1 \quad P(a_2 | a_1) = 0.5 \quad P(a_3 | a_1) = 0.4 \quad \sum_i P(a_i | a_1) = 1$$

$$P(a_1 | a_2) = 0.95 \quad P(a_2 | a_2) = 0.01 \quad P(a_3 | a_2) = 0.04 \quad \sum_i P(a_i | a_2) = 1$$

$$P(a_1 | a_3) = 0.45 \quad P(a_2 | a_3) = 0.45 \quad P(a_3 | a_3) = 0.1 \quad \sum_i P(a_i | a_3) = 1$$

Now let's see how these are used to code the sequence $a_2\ a_1\ a_3$

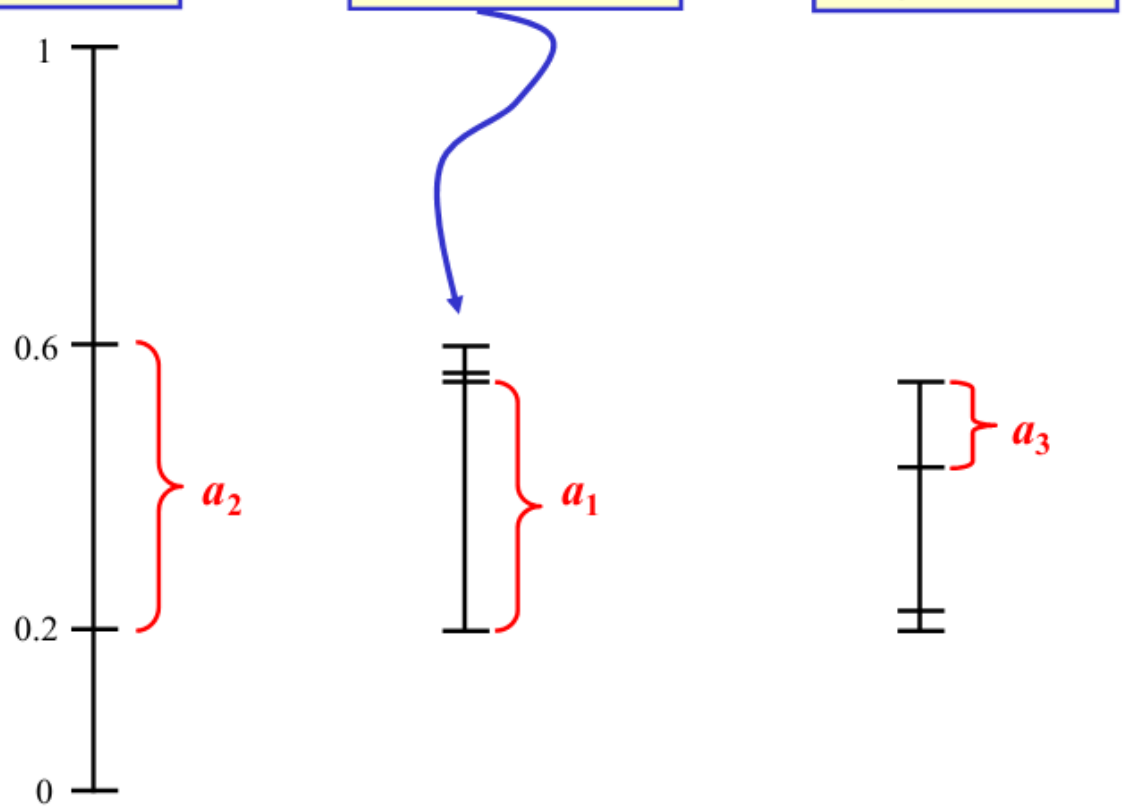
Note: Decoder needs to know these models

Example

$$P(a_1) = 0.2 \qquad P(a_2) = 0.4 \qquad P(a_3) = 0.4$$

$$P(a_1 | a_2) = 0.95 \qquad P(a_2 | a_2) = 0.01 \qquad P(a_3 | a_2) = 0.04$$

$$P(a_1 | a_1) = 0.1 \qquad P(a_2 | a_1) = 0.5 \qquad P(a_3 | a_1) = 0.4$$



Adaptive models

- Start with some *a priori* "prototype" prob model (could be cond.)
- Do coding with that for awhile as you observe the actual frequencies of occurrence of the symbols
  - Use these observations to update the probability models to better model the ***ACTUAL*** source you have!!
- Can continue to adapt these models as more symbols are observed
  - Enables tracking probabilities of source with changing probabilities
- Note: Because the decoder starts with the same prototype model and sees the same symbols the coder uses to adapt… it can automatically synchronize adaptation of its models to the coder!
  - As long as there are no transmission errors!!!

Arithmetic coding: determining conditional probabilities

- **One possibility: Bayes‘ model**

	$P(\mathtt{a}) = 0.425$	$P(\mathtt{b}) = 0.425$	$P(\square) = 0.15$
b	$P(\mathtt{a} \mid \mathtt{b}) = 0.28$	$P(\mathtt{b} \mid \mathtt{b}) = 0.57$	$P(\square \mid \mathtt{b}) = 0.15$
bb	$P(\mathtt{a} \mid \mathtt{bb}) = 0.21$	$P(\mathtt{b} \mid \mathtt{bb}) = 0.64$	$P(\square \mid \mathtt{bb}) = 0.15$
bbb	$P(\mathtt{a} \mid \mathtt{bbb}) = 0.17$	$P(\mathtt{b} \mid \mathtt{bbb}) = 0.68$	$P(\square \mid \mathtt{bbb}) = 0.15$
bbba	$P(\mathtt{a} \mid \mathtt{bbba}) = 0.28$	$P(\mathtt{b} \mid \mathtt{bbba}) = 0.57$	$P(\square \mid \mathtt{bbba}) = 0.15$

In our example we assumed that, the symbol ‘□’ has probability 0.15, and 0.85 we split between ‘a’ in ‘b’ using the following rule:

$$P_{\mathrm{L}}(\mathtt{a} \mid x_1, \ldots, x_{n-1}) = \frac{F_{\mathtt{a}} + 1}{F_{\mathtt{a}} + F_{\mathtt{b}} + 2}$$ Laplace rule

$F_{\mathtt{a}}(x_1, \ldots, x_{n-1})$ Number of *a*-s in sequence x_1 to x_{n-1}

$F_{\mathtt{b}}$ Number of *b*-s in sequence x_1 to x_{n-1}

Arithmetic coding: summary

- With arithmetic coding we encode a sequence and not separate symbols.

- Arithmetic coding is optimal. The larger is the sequence of symbols the closer we get to the entropy H(X).

- Probability distibution can be easily changed or adapted online.

- Real-time coding and decoding.