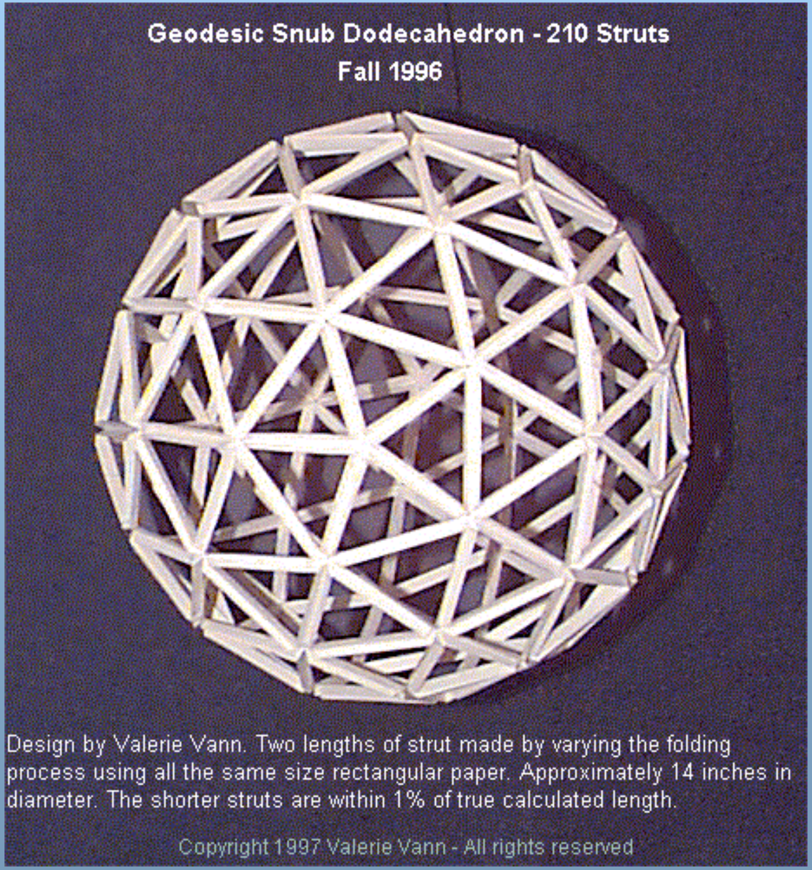
The Revenge of a Student - Symbol Codes

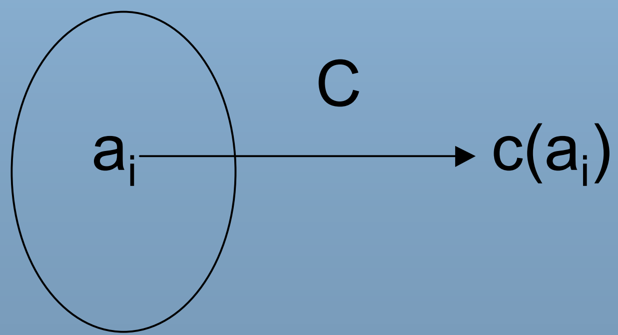


Symbol codes

- Notation: $\{0,1\}^+ = \{0,1,00,01,10,11,000,\ldots\}$
- A symbol code *C* is a mapping from $\mathcal{A}_x$ to $\{0,1\}^+$

$$c^+(x_1 x_2 x_3 \ldots x_N) = c(x_1)c(x_2)c(x_3)\ldots c(x_N)$$

$\mathcal{A}_x$



$$l(x) = |x|$$

Decoding of symbol codes

- A code C(X) is uniquely decodable if

$$\forall \mathbf{x}, \mathbf{y} \in A_X^+, \mathbf{x} \neq \mathbf{y} \Rightarrow c^+(\mathbf{x}) \neq c^+(\mathbf{y})$$

- A code C(X) is a prefix code if no codeword is a prefix of any other codeword
- The expected length L(C,X) of a symbol code C for ensemble X is

$$L(C, X) = \sum_{x \in A_X} P(x) l(x)$$

Example

$\mathcal{A}_X$ = {1,2,3,4}, P_X = {1/2,1/4,1/8,1/8}

C: c(1) = 0, c(2) = 10, c(3) = 110, c(4) = 111

The entropy of X is 1.75 bits: L(C,X) is also 1.75 bits

Obs!

$$l_i = \log_2(1/p_i), p_i = 2^{-l_i}$$

Kraft inequality

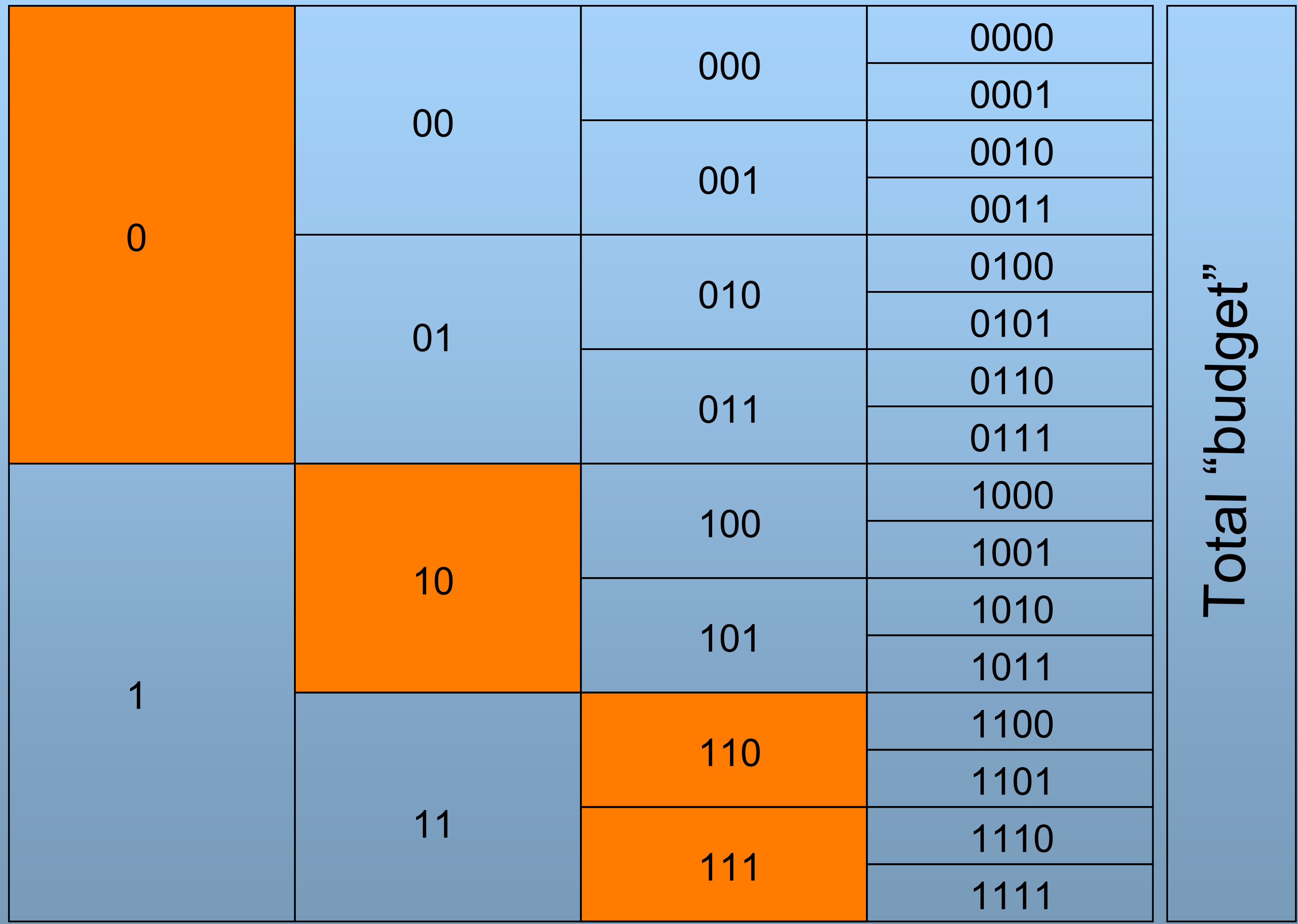
- Given a list of integer $\{l_i\}$, does there exist a uniquely decodable code with $\{l_i\}$?
- "Market model": total budget 1; cost per codeword of length *l* is 2^{-l}.

Kraft inequality: For any uniquely decodeable code *C* over the binary alphabet {0,1}, the codeword lengths must satisfy:

$$\sum_i 2^{-l_i} \leq 1$$

Conversely, given a set of codeword lengths that satisfythis inequality, there exists a uniquely decodable prefix code with these codelengths.

Limits of unique decodeability



What can we hope for?

Lower bound on expected length: The expected length L(C,X) of a uniquely decodable code is bounded below by H(X).

Compression limit of symbol codes: For an ensemble X there exists a prefix code

$$H(X) \leq L(C,X) < H(X) + 1.$$

"Proof-map" of the lower bound

Define $q_i \equiv 2^{-l_i} / z$, where $z = \sum_{i'} 2^{-l_{i'}}$

By the definition of log

Thus $l_i = \log 1/q_i - \log z$

Substitution

$$L(C,X) = \sum_i p_i l_i = \sum_i p_i \log 1/q_i - \log z$$

Kraft inequality

$$\geq 0$$

$$\geq \sum_i p_i \log 1/p_i - \log z$$

Gibbs inequality

$$\geq H(X)$$

Proof of Gibbs' inequality

- Jensen's inequality: f(E(x)) ≤ E(f(x))

$$\Rightarrow \int p(x) \log \frac{p(x)}{q(x)} \geq -\log \int p(x) \frac{q(x)}{p(x)}$$

$$\Rightarrow \int p(x) \log \frac{p(x)}{q(x)} \geq 0$$

$$\Rightarrow -\int p(x) \log q(x) \geq -\int p(x) \log p(x),$$

- Alternative proofs: see e.g. Wikipedia

(What happens if we use the "wrong" code?)

Assume the "true probability distribution" is $\{p_i\}$. If we use a complete code with lengths l_i, they define a probabilistic model $q_i = 2^{-l_i}$. The average length is

$$L(C,X) = H(X) + \sum_i p_i \log p_i / q_i$$

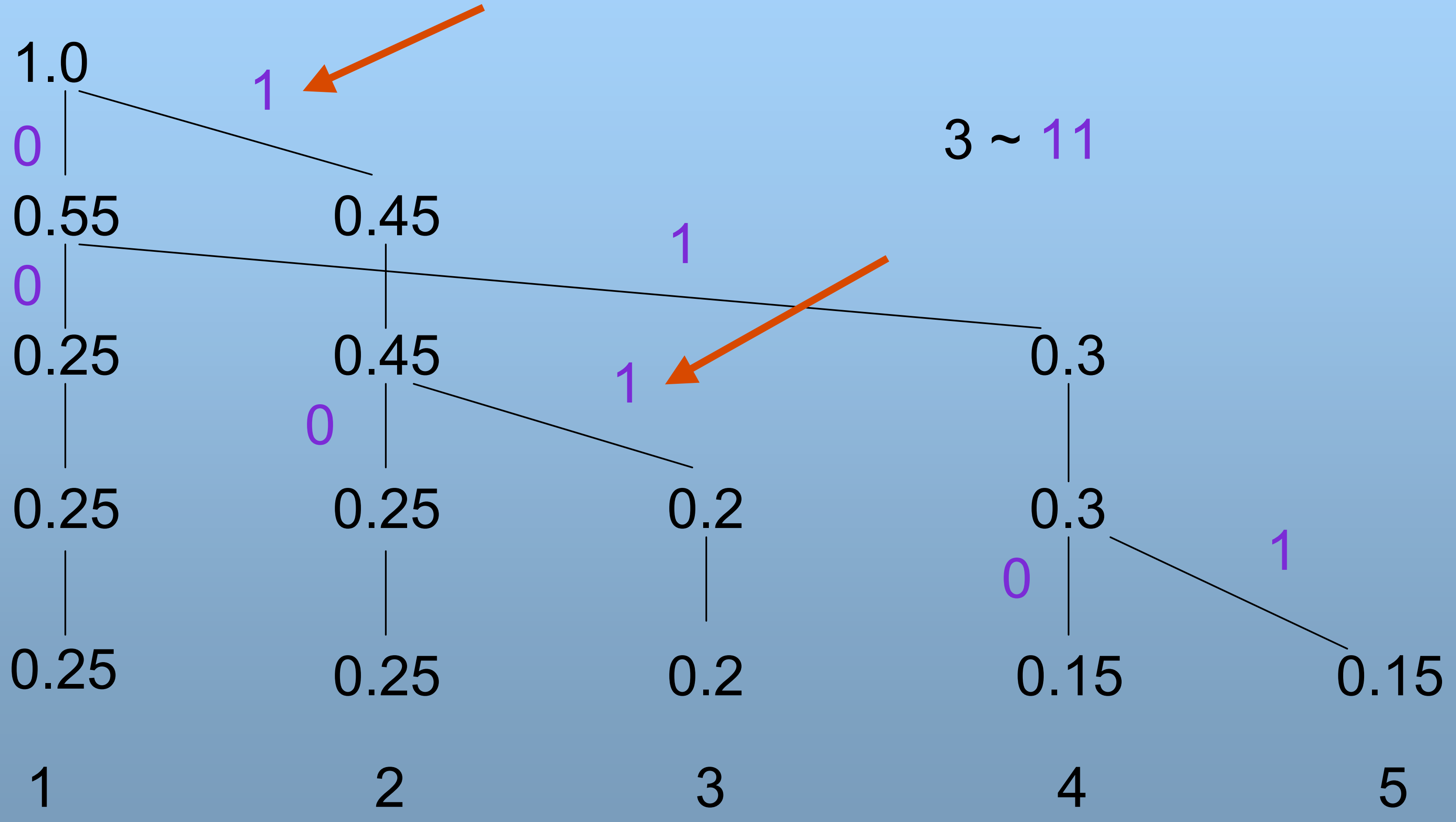
Kullback-Leibler divergence $D_{KL}(p||q)$

NB: The expected code length reaches the minimum H(X) when $l_i = \log(1/p_i)$
(in other words: when p=q and K-L divergence is zero)

Optimal symbol code: Huffman coding

- Take two least probable symbols in the alphabet as defined by $\{p_i\}$.
- Combine these symbols into a single symbol, $p_{new} = p_1 + p_2$. Repeat (until one symbol)

Huffman in practice



Huffman for the Linux manual

L(C,X) = 4.15 bits

H(X) = 4.11 bits

a_i	p_i	$\log_2 \frac{1}{p_i}$	l_i	$c(a_i)$
a	0.0575	4.1	4	0000
b	0.0128	6.3	6	001000
c	0.0263	5.2	5	00101
d	0.0285	5.1	5	10000
e	0.0913	3.5	4	1100
f	0.0173	5.9	6	111000
g	0.0133	6.2	6	001001
h	0.0313	5.0	5	10001
i	0.0599	4.1	4	1001
j	0.0006	10.7	10	1101000000
k	0.0084	6.9	7	1010000
l	0.0335	4.9	5	11101
m	0.0235	5.4	6	110101
n	0.0596	4.1	4	0001
o	0.0689	3.9	4	1011
p	0.0192	5.7	6	111001
q	0.0008	10.3	9	110100001
r	0.0508	4.3	5	11011
s	0.0567	4.1	4	0011
t	0.0706	3.8	4	1111
u	0.0334	4.9	5	10101
v	0.0069	7.2	8	11010001
w	0.0119	6.4	7	1101001
x	0.0073	7.1	7	1010001
y	0.0164	5.9	6	101001
z	0.0007	10.4	10	1101000001
-	0.1928	2.4	2	01

Figure 3.3. Huffman code for the English language ensemble introduced in figure 1.16.

Why is this not the end of the story?

- Adaptation: what if the ensemble X changes? (as it does…)
  - ✓ calculate probabilities in one pass
  - ✓ communicate code + the Huffman-coded message
- "The extra bit": what if H(X) ~1 bit?
  - ✓ Group symbols to blocks and design a "Huffman block code"

IEEE Information Society Golden Award: Stream codes

The guessing game

THERE-IS-NO-GROUP-LIKE-COSCO-GROUP

2115112113111121111132111111121111

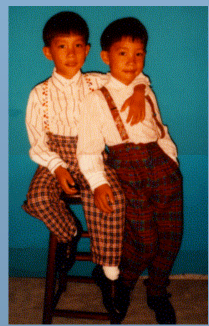
"A new alphabet"

The number of guesses before the character was identified

Encode: use the number of guesses

2115112113111121111132111111121111

Decode: let the twin guess and stop after the communicated number of guesses



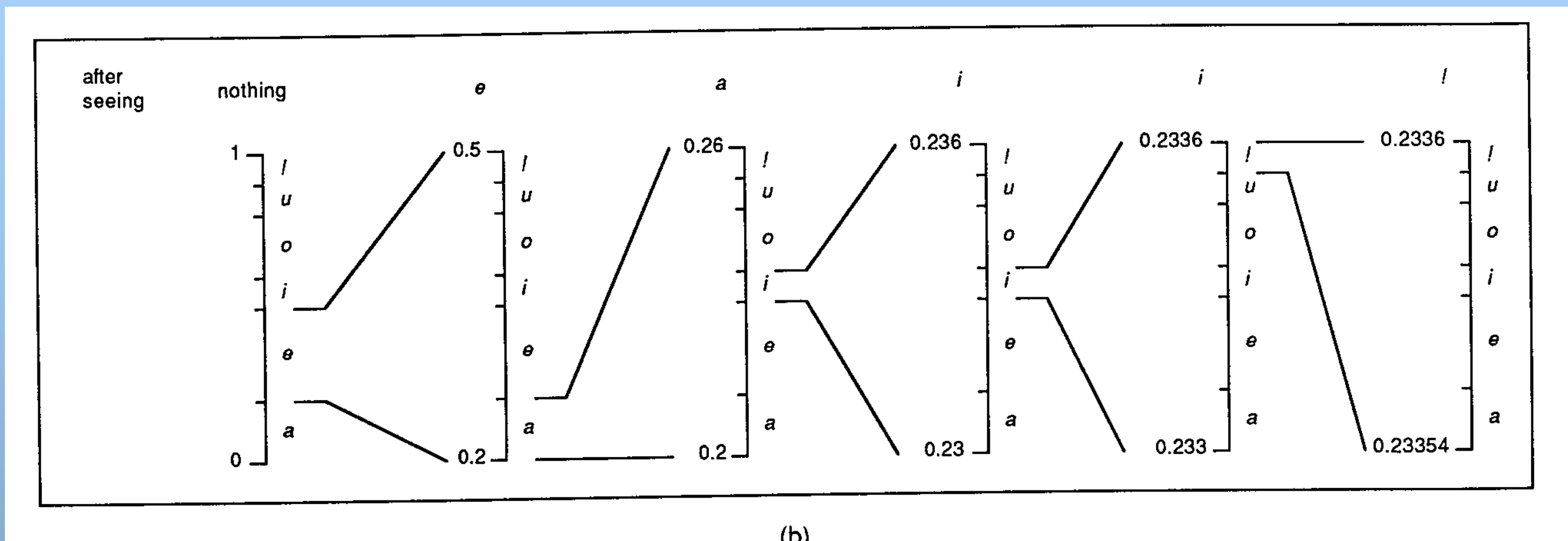
History of arithmetic coding

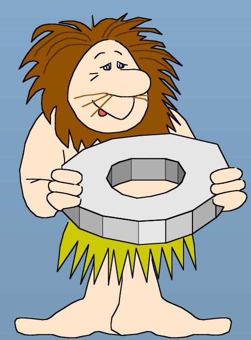
- Does not require that the symbols translate into integral number of bits
- Shannon 1948 discussed binary fractions
- First code of this type discovered by Elias
- 1976 Pasco and Rissanen (independently)
- Rissanen & Langdon 1979 described hardware implementation

An example fixed model

Symbol	Probability	Range
a	0.2	[0,0.2)
e	0.3	[0.2,0.5)
i	0.1	[0.5,0.6)
o	0.2	[0.6,0.8)
u	0.1	[0.8,0.9)
!	0.1	[0.9,1.0)

The idea


after seeing
nothing
e
a
i
i
!
1
0
0.5
0.2
0.26
0.2
0.236
0.23
0.2336
0.233
0.2336
0.23354
!
u
o
i
e
a
(b)



Arithmetic coding

- with every new symbol produced by the source, the probabilistic model provides a predictive distribution over all possible values of the next symbol
- encoder uses the model predictions to create a binary string
- dynamic model (chain rule):

$$P(e,a,i,i,!)=P(e)P(a|e)P(i|e,a)P(i|e,a,i)P(!|e,a,i,i)$$

Basics

- Source alphabet $\mathcal{A}_X = \{a_1,\ldots,a_I\}$
- Source stream $x_1, x_2, \ldots$
- Model M:

$$P(x_n = a_i \mid x_1,\ldots,x_{n-1})$$

- A binary transmission is viewed defining an interval within the real line from 0 to 1

01101 ⟶ [0.01101,0.01110)

Basics continued

- [0,1) can be divided into I intervals according to $P(x_1{=}a_i)$

$$[0,P(x_1 = a_1)),[P(x_1 = a_1),P(x_1 = a_2)),\ldots$$

- Repeat the same procedure with interval a_i to get $a_i\, a_1, \ldots, a_i\, a_I$ so that the length of $a_i\, a_j$ is proportional to

$$P(x_2 = a_j \mid x_1 = a_i)$$

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

Encoding example

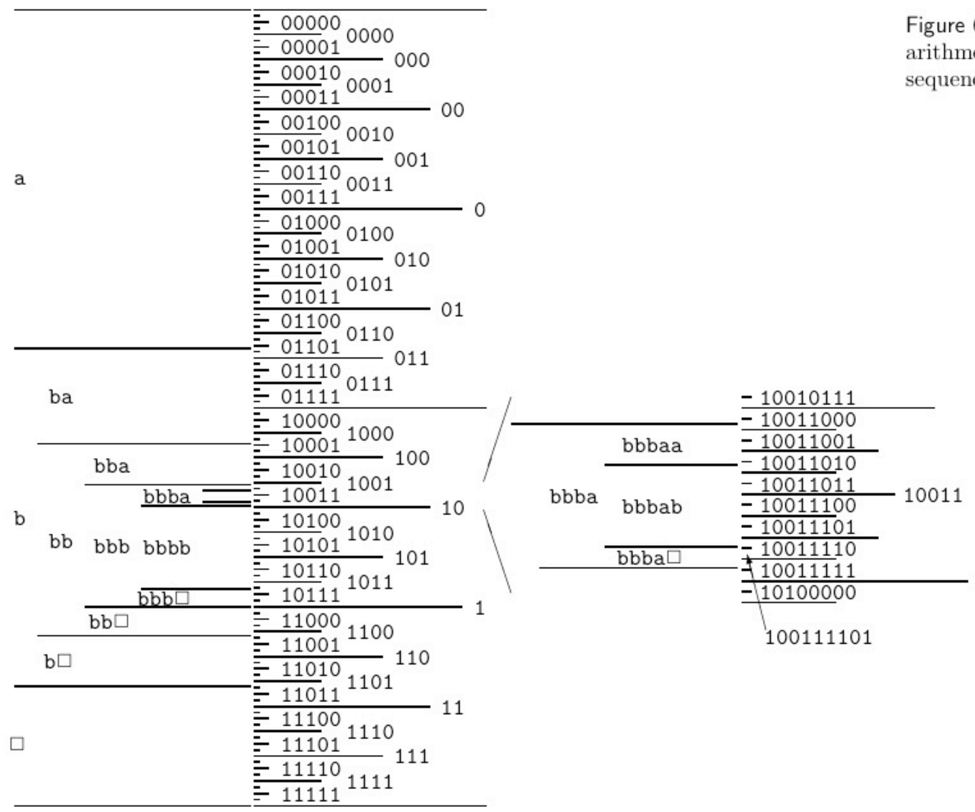


Figure 6.4. Illustration of the arithmetic coding process as the sequence bbba□ is transmitted.

Decoding example

10011101

?

Decoder

Calculate the initial P(a), P(b) and P(!) [duplicate the encoder!] and deduce the intervals “a”, “b” and “!”

10 → Deduce that the first symbol was “b”

Calculate P(a|b), P(b|b) and P(!|b) and deduce the intervals “ba”, “bb” and “b!”

1001 → Deduce that the second symbol was “b” Etc.

Lempel-Ziv coding

- simple to implement, asymptotic rate approaches the entropy
- widely used (gzip, compress,...)
- basic idea: replace a substring with a pointer to an earlier occurrence of the substring
- Example:
  - ✓ String: 1011010100010...
  - ✓ Substrings: 1, 0, 11, 01, 010, 00, 10,...
  - ✓ Replace 010 with a pointer to "01" + "0"

Various codes: the big picture

- fixed length block codes: mappings from a fixed number of course symbols to a fixed length binary message
- symbol codes
  - ✓ variable length code for each symbol in the alphabet
  - ✓ code lengths integers
  - ✓ Huffmann code (expectation) optimal

…big picture continued

- stream codes
  - ✓ not constrained to emit at least one bit for every symbol in the source stream
  - ✓ arithmetic codes use a probabilistic model that identifies each string with a sub-interval of [0,1). "Good compression requires intelligence"
  - ✓ Lempel-Ziv codes memorize strings that have already occurred. "No prior assumptions on the world"