

Instantaneously Decodable Codes

A coding scheme is said to be *instantaneously decodable* if encoded text using it can be decoded into the original text by scanning the encoded text from left to right (no back-tracking), picking off the decoded symbols the moment a symbol code is scanned.

The *prefix property* — no code for a symbol is a prefix of the code for another symbol — is a necessary and sufficient condition for instantaneous decoding.

The remarkable thing is that there is a simple arithmetical condition on the length of the codes for the scheme to be instantaneously decodable. This is the *Kraft Inequality*.

The Kraft Inequality

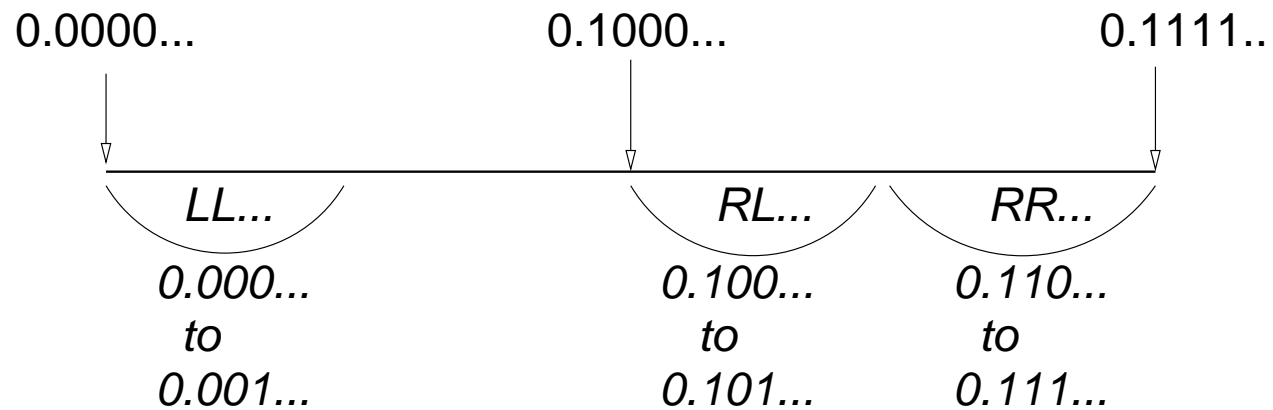
Suppose there are n symbols s_1, \dots, s_n . Let them be encoded into codes c_1, \dots, c_n . By $|c|$ we mean the length of code c .

$$\sum_{i=1}^{i=n} 2^{-|c_i|} \leq 1 \quad (1)$$

This is known as the *Kraft Inequality*.

Theorem: A coding scheme is instantaneously decodable implies the code lengths satisfy the Kraft Inequality. Conversely, if the Kraft Inequality holds, there exists an instantaneously decodable code of those lengths.

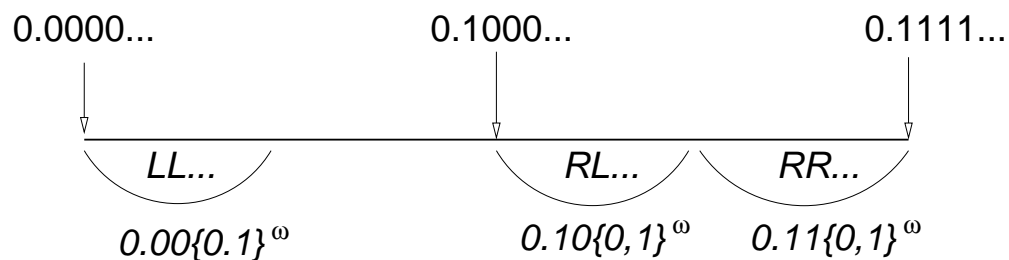
The 0-1 Interval and 0-1 Codes I



0.101101... interpret as: RLRRLR...
 where 0 = Left half; 1 = Right half

The 0-1 Interval and 0-1 Codes II

A CONCISE REPRESENTATION FOR INFINITE STRINGS



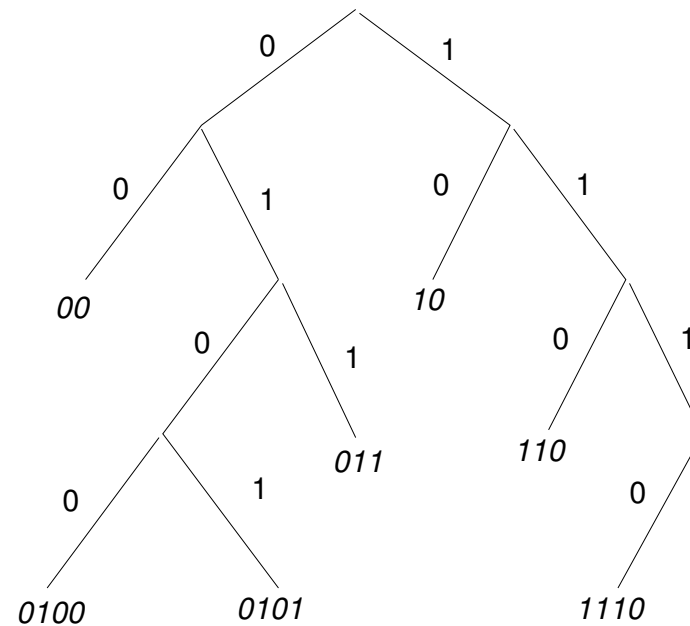
0.101101... interpret as: RLRRLR...

where 0 = Left half; 1 = Right half

$\{a,b\}^\omega$ means an infinite string of a's and b's, e.g.

aababbbaababbbaaab....

0-1 Prefix Codes and Trees

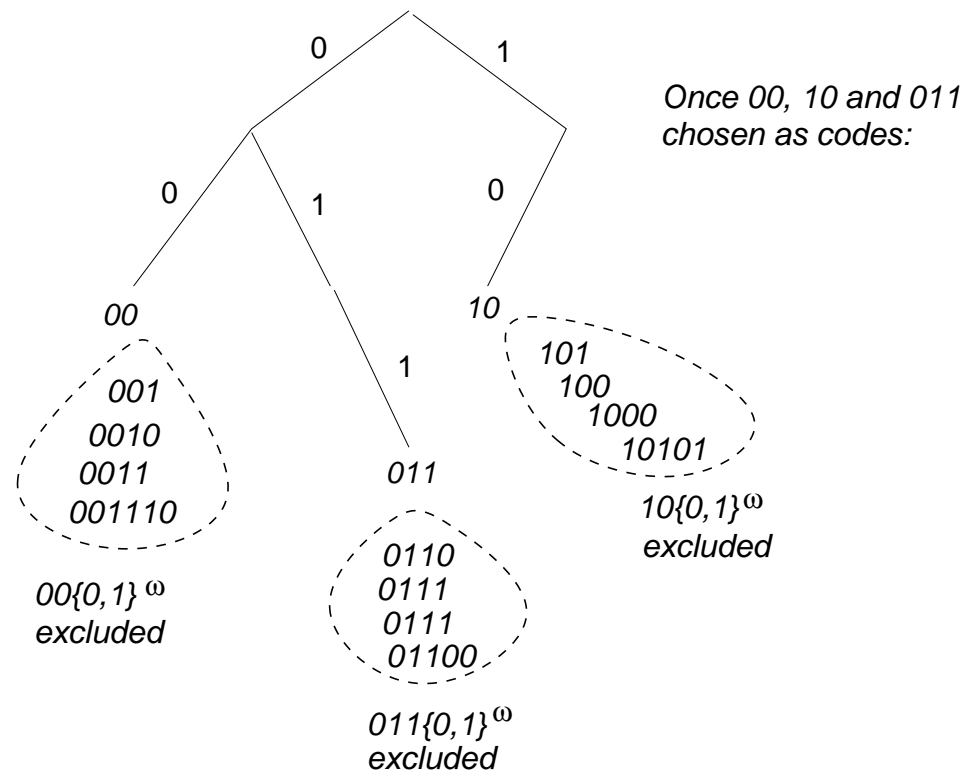


Real-time decodable code = can read codes off from the left w/o back-tracking

Implies no code is the prefix of another -- Prefix Codes

Prefix code iff all codes are in leaves of code assignment tree

Prefix Trees and Excluded Sub-trees



Each leaf of a prefix tree is a "root" of an implicit exclusion subtree!
 Say that the code there "covers" the excluded codes in its subtree

Excluded Strings

Lemma A

If $a_1 a_2 \dots a_k$ is a chosen prefix code, then all strings of the form $a_1 a_2 \dots a_k \{0, 1\}^+$ are excluded as codes.

Corollary B

In particular, if this is so, then no string whose initial segment is in the *sub-interval* $[0.a_1 a_2 \dots a_k 0^\omega, 0.a_1 a_2 \dots a_k 1^\omega]$ can be used as another code.

As the difference between the ends of this sub-interval is $0.0^k 1^\omega$, its *length* is 2^{-k} .

Uniqueness of Exclusion

Lemma C

$a_1a_2 \dots a_k$ excludes $c_1c_2 \dots c_l$ if and only if $a_1a_2 \dots a_k$ is a *prefix* of $c_1c_2 \dots c_l$, i.e., $a_1 = c_1, a_2 = c_2, \dots, a_k = c_k$ where $k < l$.

Corollary D

No two distinct prefix codes can exclude the same code.

Proof

Suppose distinct prefix codes $a_1a_2 \dots a_k$ and $b_1b_2 \dots b_j$ both exclude $c_1c_2 \dots c_l$. Then by the lemma, both $a_1a_2 \dots a_k$ and $b_1b_2 \dots b_j$ are prefixes of $c_1c_2 \dots c_l$. Thus if $k \leq j$, $a_i = c_i = b_i$ for $1 \leq i \leq k$, and therefore $a_1a_2 \dots a_k$ is a prefix of $b_1b_2 \dots b_j$, a contradiction. The other case $l \leq k$ is similar.

Total Covered Length

Lemma E

If s_1, s_2, \dots, s_n are prefix 0 – 1 codes, then they exclude codes whose prefixes occupy disjoint sub-intervals of lengths $|s_1|, |s_2|, \dots, |s_n|$.

Proof

From the previous lemmas.

Corollary F — one half of the Kraft Inequality

If s_1, s_2, \dots, s_n are prefix codes, then

$$\sum_{i=1}^{i=n} 2^{-|s_i|} \leq 1 \quad (2)$$

Existence of Prefix Codes

We now argue that if there are numbers x_1, x_2, \dots, x_n such that

$$\sum_{i=1}^{i=n} 2^{-|x_i|} \leq 1 \quad (3)$$

then there is a set of prefix codes s_1, s_2, \dots, s_n such that $|s_i| = x_i$ for $1 \leq i \leq n$. Without loss of generality we may assume that $x_1 \leq x_2 \leq \dots \leq x_n$.

Desired Code Lengths

Suppose we want codes of increasing lengths $j_1, j_2 \dots j_k$. Let $N(j_1), N(j_2) \dots N(j_k)$ be the number of codes corresponding to those lengths. Then the Kraft Inequality can be re-written

$$\sum_{i=1}^{i=k} N(j_i) 2^{-j_i} \leq 1 \quad (4)$$

Algorithm for Trie Construction

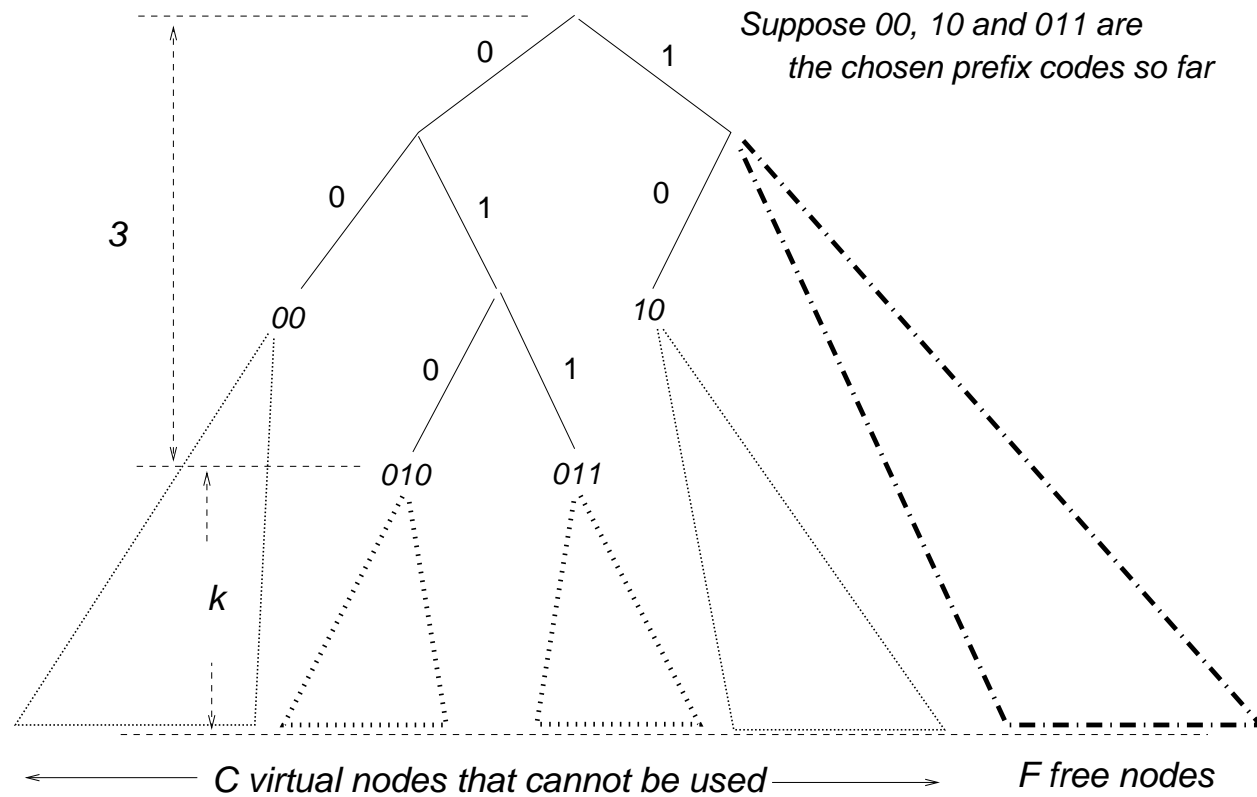
We construct a trie for the codes as follows.

Start with a (binary) tree of depth j_1 . To the leaves of this tree, assign j of them to the shortest $N(j_1)$ desired codes. Then extend the unassigned leaves as follows: make them the roots of subtrees that extend the original tree to the next depth j_2 . Assign $N(j_2)$ of those new leaves to the next shortest codes.

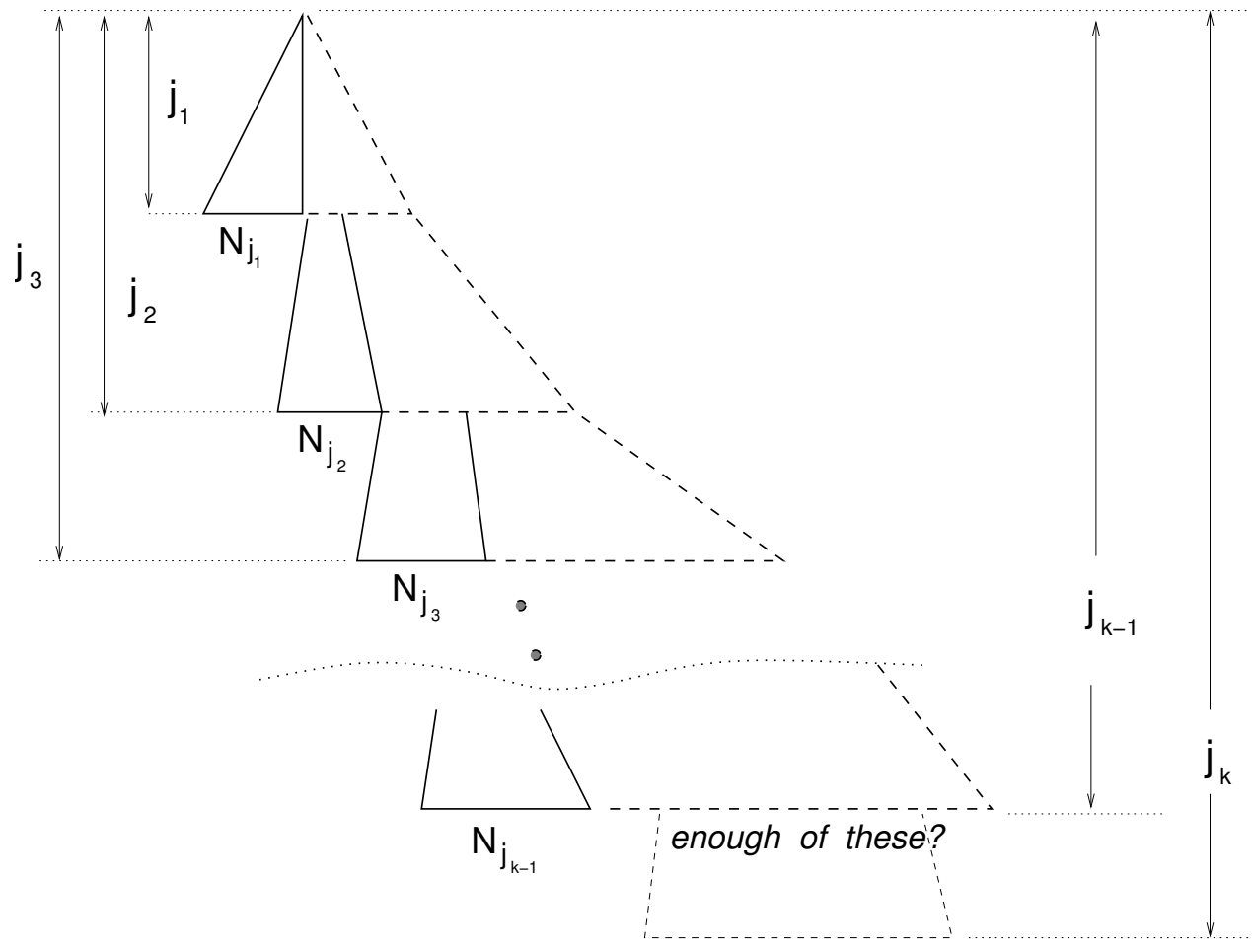
After the i -th stage of the algorithm the tree has been extended to a depth of j_i and at that new frontier $N(j_i)$ of its leaves have been assigned.

The feasibility (correctness) of the algorithm is proved if we can show that at any stage *it is the case that enough leaves are left over to extend for the codes of the next higher length to be assigned.*

Code Construction — A Sketch



Enough Free Nodes?



Code Construction — A Sketch

At level j_k there are 2^{j_k} virtual nodes. These are the total number of nodes available for codes *if no ancestors were “used up” at higher levels.*

At level j_1 , N_{j_1} are used up by assigning codes to them. Therefore the number of “lost” virtual nodes at level j_k is $N_{j_1} \cdot 2^{j_k - j_1}$.

Generally, the number of lost virtual nodes at level j_k due to N_{j_h} assignments at higher level j_h ($j_h < j_k$) is $N_{j_h} \cdot 2^{j_k - j_h}$.

Hence the number $F(k)$ of *free* nodes at level j_k is

$$2^{j_k} - 2^{j_k} (N_{j_1} \cdot 2^{-j_1} + \dots + N_{j_{k-1}} \cdot 2^{-j_{k-1}}).$$

Thus, there is space for the N_{j_k} nodes at level j_k if $N_{j_k} \leq F(k)$, i.e.

$$N_{j_k} \leq 2^{j_k} (1 - N_{j_1} \cdot 2^{-j_1} + \dots + N_{j_{k-1}} \cdot 2^{-j_{k-1}}).$$

But the last expression is precisely the Kraft Inequality!