Representation of Real Number in Binary

- Any real number $x$ in the interval $[0,1)$ can be represented in binary as $0.b_1b_2...$ where $b_i$ is a bit.
Real-to-Binary Conversion Algorithm

L := 0; R := 1; i := 1

while x > L *
    if x < (L+R)/2 then b_i := 0 ; R := (L+R)/2;
    if x ≥ (L+R)/2 then b_i := 1 ; L := (L+R)/2;
    i := i + 1
end{while}

b_j := 0 for all j ≥ i

* Invariant: x is always in the interval [L,R)
Arithmetic Coding

- Basic idea in arithmetic coding (Shannon-Fano-Elias):
  - Represent each string $x$ of length $n$ by a unique interval $[L,R)$ in $[0,1)$.
  - The width $r-l$ of the interval $[L,R)$ represents the probability of $x$ occurring.
  - The interval $[L,R)$ can itself be represented by any number, called a tag, within the half open interval.
  - The $k$ significant bits of the tag $\cdot t_1t_2t_3...$ is the code of $x$. That is, $\cdot t_1t_2t_3...t_k000...$ is in the interval $[L,R)$. 
Example of Arithmetic Coding

1. tag must be in the half open interval.
2. tag can be chosen to be \((L+R)/2\).
3. code is the significant bits of the tag.

\[
\begin{align*}
\text{tag} &= 17/27 = .101000010...
\text{code} &= 101
\end{align*}
\]
Some Tags are better than others

Using tag = (L+R)/2

tag = 13/27 = .011110110...
code = 0111

Alternative tag = 14/27 = .100001001...
code = 1
Examples

- \( P(a) = \frac{1}{3}, \ P(b) = \frac{2}{3}. \)

<table>
<thead>
<tr>
<th>Tag</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>aaa</td>
</tr>
<tr>
<td>000001001...</td>
<td>0 aab</td>
</tr>
<tr>
<td>000100110...</td>
<td>0001 aba</td>
</tr>
<tr>
<td>010000101...</td>
<td>01 abb</td>
</tr>
<tr>
<td>010111110...</td>
<td>01011 baa</td>
</tr>
<tr>
<td>011110111...</td>
<td>0111 bab</td>
</tr>
<tr>
<td>101000010...</td>
<td>101 bba</td>
</tr>
<tr>
<td>110110100...</td>
<td>11 bbb</td>
</tr>
<tr>
<td>0.95 bits/symbol</td>
<td></td>
</tr>
<tr>
<td>0.92 entropy lower bound</td>
<td></td>
</tr>
</tbody>
</table>
Code Generation from Tags

- If binary tag is \( t_1t_2t_3\ldots = (L+R)/2 \) in \([L,R]\) then we want to choose \( k \) to form the code \( t_1t_2\ldots t_k \).

- **Short code:**
  - choose \( k \) to be as small as possible so that 
    \[ L \leq t_1t_2\ldots t_k000\ldots < R. \]

- **Guaranteed code:**
  - choose \( k = \lceil \log_2 (1/(R-L)) \rceil + 1 \)
  - \( L \leq t_1t_2\ldots t_kb_1b_2b_3\ldots < R \) for any bits \( b_1b_2b_3\ldots \)
  - for fixed length strings provides a good prefix code.
  - example: \([.000000000\ldots, .000010010\ldots]\), tag = .000001001\ldots
    Short code: 0
    Guaranteed code: 000001
Guaranteed Code Example

- $P(a) = 1/3$, $P(b) = 2/3$.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Short Code</th>
<th>Prefix Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/27</td>
<td>0.000001001...</td>
<td>0</td>
</tr>
<tr>
<td>1/27</td>
<td>0.000100110...</td>
<td>0001</td>
</tr>
<tr>
<td>3/27</td>
<td>0.001001100...</td>
<td>001</td>
</tr>
<tr>
<td>5/27</td>
<td>0.010000101...</td>
<td>01</td>
</tr>
<tr>
<td>9/27</td>
<td>0.010111110...</td>
<td>01011</td>
</tr>
<tr>
<td>11/27</td>
<td>0.011110111...</td>
<td>0111</td>
</tr>
<tr>
<td>15/27</td>
<td>0.101000010...</td>
<td>101</td>
</tr>
<tr>
<td>19/27</td>
<td>0.110110100...</td>
<td>11</td>
</tr>
<tr>
<td>27/27</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Arithmetic Coding Algorithm

\[ C(x_i) = P(x_0) + P(x_1) + \ldots + P(x_i) \]

Initialize \( L = 0 \) and \( R = 1 \);
For \( i = 1 \) to \( n \) do
  \[ W := R - L; \]
  \[ L := L + W \times C(x_{i-1}); \]
  \[ R := L + W \times C(x_i); \]
  \[ T := (L + R)/2; \]
Choose code for the tag
Example

\[ P(A) = \frac{1}{4}, \ P(b) = \frac{1}{2}, \ P(c) = \frac{1}{4} \]
\[ C(a) = \frac{1}{4}, \ C(b) = \frac{3}{4}, \ C(c) = 1 \]

\text{abca}

<table>
<thead>
<tr>
<th>symbol</th>
<th>W</th>
<th>L</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1/4</td>
</tr>
<tr>
<td>W := R - L;</td>
<td>a</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>L := L + W C(x);</td>
<td>b</td>
<td>1/4</td>
<td>1/16</td>
</tr>
<tr>
<td>R := L + W P(x)</td>
<td>c</td>
<td>1/8</td>
<td>5/32</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>1/32</td>
<td>5/32</td>
</tr>
</tbody>
</table>

tag = \(\frac{5/32 + 21/128}{2} = \frac{41}{256} = 0.001010010\ldots\)
\[ L = 0.001010000\ldots \]
\[ R = 0.001010100\ldots \]
\text{code} = 00101
\text{prefix code} = 00101001
Example

\[ P(A) = \frac{1}{4}, \ P(b) = \frac{1}{2}, \ P(c) = \frac{1}{4} \]

\[ C(a) = \frac{1}{4}, \ C(b) = \frac{3}{4}, \ C(c) = 1 \]

\[ bbbbb \]

<table>
<thead>
<tr>
<th>symbol</th>
<th>W</th>
<th>L</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ W := R - L; \]
\[ L := L + W \ C(x); \]
\[ R := L + W \ P(x) \]

tag =
L =
R =
code =
prefix code =
Decoding

- Assume the length is known to be 3.
- 0001 which converts to the tag .0001000...

```
   .0001000...          output a
     ---------------
     a
     b
     1
```
Decoding

- Assume the length is known to be 3.
- 0001 which converts to the tag .0001000...

```plaintext
.0001000...
  aa
  a
  ab
  b
  1

output a
```
Decoding

- Assume the length is known to be 3.
- 0001 which converts to the tag .0001000...
Arithmetic Decoding Algorithm

\[ C(x_i) = P(x_0) + P(x_1) + \ldots + P(x_i) \]

Decode \(b_1b_2...b_m\), the number of symbols in \(n\)

Initialize \(L := 0\) and \(R := 1\);
\(t := .b_1b_2...b_m\)
For \(i = 1\) to \(n\) do
  \(W := R-L;\)
  Find \(j\) such that \(L + W*C(x_{j-1}) \leq t < L + W*C(x_j)\)
  Output \(x_j;\)
  \(L := L + W*C(x_{j-1});\)
  \(R := L + W*C(x_j);\)
Decoding Example

\[ P(a) = \frac{1}{4}, \ P(b) = \frac{1}{2}, \ P(c) = \frac{1}{4} \]
\[ C(a) = 0, \ C(b) = \frac{1}{4}, \ C(c) = \frac{3}{4} \]

- 00101

<table>
<thead>
<tr>
<th>W</th>
<th>L</th>
<th>R</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1/4</td>
<td>a</td>
</tr>
<tr>
<td>1/4</td>
<td>1/16</td>
<td>3/16</td>
<td>b</td>
</tr>
<tr>
<td>1/8</td>
<td>5/32</td>
<td>6/32</td>
<td>c</td>
</tr>
<tr>
<td>1/32</td>
<td>5/32</td>
<td>21/128</td>
<td>a</td>
</tr>
</tbody>
</table>

\[ \text{tag} = .00101000... = 5/32 \]
Decoding Issues

- There are two ways for the decoder to know when to stop decoding.
  1. Transmit the length of the string
  2. Transmit a unique end of string symbol
Practical Arithmetic Coding

- Scaling:
  - By scaling we can keep L and R in a reasonable range of values so that $W = R - L$ does not underflow.
  - The code can be produced progressively, not at the end.
  - Complicates decoding some.

- Integer arithmetic coding avoids floating point altogether.
Uniqueness and Efficiency of Arithmetic Code

- Uniqueness:
  Proof:

- Efficiency:
  Proof: