Reals in Binary

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

**First Conversion**

```
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 $(L,R)$

**Conversion using Scaling**

- Always scale the interval to unit size, but $x$ must be changed as part of the scaling.

**Binary Conversion with Scaling**

```
y := x; i := 0
while y > 0 *
    i := i + 1;
    if y < 1/2 then b_i := 0; y := 2y;
    if y > 1/2 then b_i := 1; y := 2y - 1;
end {while}
b_j := 0 for all j > i + 1
```

* Invariant: $x = \ldots b_ib_2... b_1 + y/2$

**Proof of the Invariant**

- Initially $x = 0 + y/2^0$
- Assume $x = b_ib_2... b_1 + y/2^i$
  - Case 1. $y < 1/2$. $b_i = 0$ and $y' = 2y$
    \[ b_ib_2... b_{i+1} + y'/2^{i+1} = b_ib_2... b_{i+1} + 2y2^{-i+1} \]
    \[ = x \]
  - Case 2. $y \geq 1/2$. $b_i = 1$ and $y' = 2y - 1$
    \[ b_ib_2... b_{i+1} + y'/2^{i+1} = b_ib_2... b_{i+1} + (2y-1)2^{i-1} \]
    \[ = b_ib_2... b_i + y/2 - 1/2 + 2y2^{i-1} - 1/2 \]
    \[ = b_ib_2... b_i + y/2 \]
    \[ = x \]
Example and Exercise

\[ x = \frac{1}{3} \quad x = \frac{17}{27} \]

\[
\begin{array}{c|c|c}
 y & i & b \\
\hline
 \frac{1}{3} & 1 & 0 \\
 2/3 & 2 & 1 \\
 \frac{1}{3} & 3 & 0 \\
 2/3 & 4 & 1 \\
 \vdots & \vdots & \vdots \\
\end{array}
\]

Arithmetic Coding

Basic idea in arithmetic coding:
- 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 \( t_1 t_2 \ldots t_k \) is the code of \( x \). That is, \( \ldots t_1 t_2 \ldots t_k 000 \ldots \) is in the interval \([L, R)\).
- It turns out that \( k \approx -\log_2(1/(R-L)) \).

Example of Arithmetic Coding (1)

\[
\begin{array}{c|c|c|c}
 & 0 & 1/3 & 2/3 \\
\hline
 a & 3/27 & \ldots & \ldots \\
 b & \ldots & 2/27 & 1/27 \\
 ba & \ldots & \ldots & 1/27 \\
 bab & \ldots & \ldots & \ldots \\
\hline
\end{array}
\]

Some Tags are Better than Others

\[
\begin{array}{c|c|c|c}
 & 0 & 1/3 & 2/3 \\
\hline
 a & 11/27 & \ldots & \ldots \\
 b & \ldots & 15/27 & \ldots \\
 ba & \ldots & \ldots & \ldots \\
 bab & \ldots & \ldots & \ldots \\
\hline
\end{array}
\]

Example of Codes

- \( P(a) = \frac{1}{3}, P(b) = \frac{2}{3} \).
- \( \text{tag} = (L+R)/2 \)
- \( \text{code} \)

- Short code:
  - Choose \( k \) to be as small as possible so that \( L \leq t_1 t_2 \ldots t_k 000 \ldots < R \).
- Guaranteed code:
  - Choose \( k = \lfloor \log_2 \left( (1/(R-L))+1 \right) \rfloor + 1 \)
  - \( L \leq t_1 t_2 \ldots b_1 b_2 b_3 \ldots < R \) for any bits \( b_1 b_2 b_3 \ldots \)
  - For fixed length strings provides a good prefix code.
- Example: \( \ldots \cdot 000000000 \ldots \), tag = \( .000001001 \ldots \)
- Short code: 0
- Guaranteed code: 000001
Guaranteed Code Example

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>W</th>
<th>L</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>1/4</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>0</td>
<td>1/4</td>
</tr>
<tr>
<td>a</td>
<td>1/2</td>
<td>1/16</td>
<td>3/16</td>
</tr>
<tr>
<td>b</td>
<td>1/8</td>
<td>5/32</td>
<td>6/32</td>
</tr>
<tr>
<td>b</td>
<td>1/32</td>
<td>5/32</td>
<td>21/128</td>
</tr>
</tbody>
</table>

$W := R - L$;
$L := L + W \cdot C(x)$;
$R := L + W \cdot P(x)$;

Decoder 

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

Arithmetic Coding Algorithm

- $P(a_1), P(a_2), \ldots, P(a_m)$
- $C(a_i) = P(a_1) + P(a_2) + \ldots + P(a_i)$
- Encode $x_1 x_2 \ldots x_n$

Arithmetic Coding Example

- $P(a) = 1/4$, $P(b) = 1/2$, $P(c) = 1/4$
- $C(a) = 0$, $C(b) = 1/4$, $C(c) = 3/4$

Arithmetic Coding Exercise

- $P(a) = 1/4$, $P(b) = 1/2$, $P(c) = 1/4$
- $C(a) = 0$, $C(b) = 1/4$, $C(c) = 3/4$
- bbbb

Decoding (1)

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

Decoding (2)

- Assume the length is known to be 3.
- 0001 which converts to the tag .0001000...
Decoding (3)
• Assume the length is known to be 3.
• 0001 which converts to the tag .0001000...

Arithmetic Decoding Algorithm
• $P(a_1), P(a_2), \ldots, P(a_m)$
• $C(a_i) = P(a_1) + P(a_2) + \ldots + P(a_{i-1})$
• Decode $b_1b_2\ldots b_n$, number of symbols is $n$.

```
Initialize L := 0 and R := 1;
t := .b_1b_2\ldots b_m000...
for i = 1 to n do
  W := R - L;
  find $j$ such that $L + W \cdot C(a_j) \leq t < L + W \cdot (C(a_j) + P(a_j))$
  output $a_j$;
  L := L + W \cdot C(a_j);
  R := L + W \cdot P(a_j);
```

Decoding Example
• $P(a) = 1/4$, $P(b) = 1/2$, $P(c) = 1/4$
• $C(a) = 0$, $C(b) = 1/4$, $C(c) = 3/4$
• 00101
tag = .00101000... = 5/32
```
<table>
<thead>
<tr>
<th>W</th>
<th>L</th>
<th>R</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>1/4</td>
<td>1/4</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>
```

Decoding Issues
• There are at least 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.

More Issues
• Context
• Adaptive
• Comparison with Huffman coding