符号

単語を定義していきますが、情報理論で使われている単語はあまり統一されていないので注意してください。
集合に関する単語は知っているとして、数学に寄せた言い回しを使っています。

情報理論 (information theory) は情報源符号化 (source coding) と通信路符号化 (channel coding) の 2 つで構成されています。これらを扱う分野は符号理論 (coding theory) と呼ばれるので、情報理論の大部分は符号理論です。

情報源符号化と通信路符号化が何か簡単に言っておきます。例えば、目的地までの交差点を曲がる順番が紙に右左左右と書いてあるなら、その通りに動くことで目的地にたどり着きます。これが migihidarihidarimigi と書いてあっても、ひらがなとローマ文字表記との対応を知っていれば、目的にたどり着けます。このように、元々の文章に使われている文字を別の文字や記号に変換したとき、どのような変換ならもとの文章を効率よく再現できるのかを調べるのが情報源符号化です。

相手に指示が書いてる紙を渡すときに邪魔が入り、migimihaaihidarimigi と書き換えられたとします。これを受け取ったとしても、migi と hidari しか書かれていないと知っているなら、mihaai を hidari と正しく修正でき、もとの文章を再現できます。このように文章を送る間に邪魔が入りもとの文章を維持できなくなる場合に、どれだけ効率よく相手に正しい内容を伝えられるかを調べるのが通信路符号化です。

情報源符号化、通信路符号化の両方とも確率を使って定式化しているので、情報理論では確率の知識が必要になります。

基本的な単語の定義を与えていきます。

- アルファベット

  あ, い, う,…, んや A,B,C,…,Z のように文章を作るための要素を記号 (symbol) や文字 (letter) と言い、その記号の集合をアルファベットと呼んでいます (A,B,C,…,Z によるアルファベット、あ, い, う,…, んによるアルファベット)。特に、0,1 を記号とするアルファベットをビット (bit, binary digits) と言います。

  アルファベットは記号を元とする集合です。なので、例えばローマ文字を元とするアルファベットは $\{\mathrm{A}, \mathrm{B}, \mathrm{C}, \ldots, \mathrm{Z}\}$ として、集合の表記で書かれます。抽象的には $\{a_1, a_2, \ldots, a_n\}$ のように書かれます。また、今の定義でのビットは集合 $\{0, 1\}$ を指しますが、情報量の単位でも同じビットが使われているので混同しないように注意してください (「エントロピー」参照)。

  ここではアルファベットを集合 $\mathcal{A}$ とすれば、その記号は $\mathcal{A}$ の元と呼んでいきます。

- 文字列

  アルファベットの元を並べたものを文字列（string, word）と呼びます。他にもアルファベットの元の系列 (sequence) や message と言われたりもします。

  ローマ文字による AFTGCB のようなものが文字列です。文字列はアルファベットの元 $a_1, a_2, \ldots, a_n$ を使って

  $$a_1 a_2 \cdots a_n$$

  と表記するので、積と混同しないようにしてください。n は文字列の長さ (length of string) と呼ばれます。例えば、アルファベットとして集合 $\{0, 1\}$ を使えば、$n = 2$ での文字列は

  $$00\ ,\ 01\ ,\ 10\ ,\ 11 \tag{1}$$

  として作れます。

  文字列を使っているときは、アルファベットの元と言うより文字列の文字と言った方が分かりやすそうなので、紛らわしくないときは文字と言ってしまいます。

長さが l の文字列の集合を $\mathcal{S}^l$ と表記し ((1) では $\mathcal{S}^2 = \{00, 01, 10, 11\}$)、異なる長さの文字列を全て含んだ集合を $\mathcal{S}^*$ と表記します。集合の記号で書くなら

$$\mathcal{S}^* = \mathcal{S}^0 \cup \mathcal{S}^1 \cup \mathcal{S}^2 \cup \cdots$$

$\mathcal{S}^0$ は文字列の長さが 0 の集合です。

ここでは $\mathcal{A}, \mathcal{B}$ をアルファベットとし、$\mathcal{A}, \mathcal{B}$ の元による長さ l の文字列の集合を $\mathcal{S}_A^l, \mathcal{S}_B^l$、$\mathcal{A}, \mathcal{B}$ の元による異なる長さの文字列を全て含んだ集合を $\mathcal{S}_A^*, \mathcal{S}_B^*$ と表記していきます。

ここから、$\mathcal{A}$ の元による文字列と $\mathcal{B}$ の元による文字列との対応を見ていきます。対応を一般化させるために、$\mathcal{A}$ の元と $\mathcal{B}$ の元とを直接対応させるのでなく、$\mathcal{A}$ の元と $\mathcal{B}$ の元から作られる文字列とを対応させます。例えば、$\mathcal{B} = \{0, 1\}$ として、$a \in \mathcal{A}$ を $\mathcal{B}$ の元による文字列 00 に対応させるというようにします。

このような対応の有名な例がモールス信号 (モールス符号、Morse code) です。モールス信号はローマ文字 26 個を短点「・」と長点「-」で表現します。例えば SOS は、S と O が

$$\mathrm{S} \Rightarrow \cdots, \quad \mathrm{O} \Rightarrow ---$$

と表されることから、「･･･ - - - ･･･」です。このような変換に対する単語を定義していきます。

- 符号化関数、符号語

  $\mathcal{A}$ の元から $\mathcal{B}$ の元による文字列への写像 ϕ を定義し、単射になっているとします。ϕ を符号化関数 (code, encoding) と呼ぶことにします (後に出てくる符号にも code が使われるので注意)。符号化関数は $\mathcal{A}$ から $\mathcal{B}$ への写像でなく、$\mathcal{A}$ から $\mathcal{S}_B^*$ への写像であることに注意してください。符号化関数による変換は符号化 (coding) と呼ばれます。単射なので、$a, a' \in A,\ \phi(a), \phi(a') \in \mathcal{S}_B^*$ に対して $a \neq a'$ なら $\phi(a) \neq \phi(a')$、もしくは $\phi(a) = \phi(a')$ なら $a = a'$ です。単射であることを正則 (nonsingular) と言ったりもします。

  $\mathcal{A}$ の元 a から符号化関数 ϕ によって作られた $\phi(a) \in \mathcal{S}_B^*$ のことを符号語 (codeword) と言い、符号語 $\phi(a)$ の集合 $\mathcal{C} = \{\phi(a) |\ a \in \mathcal{A}\}$ を符号 (code) と言います。符号語を作るアルファベットは符号アルファベット (code alphabet) と呼ばれ、今の場合では $\mathcal{B}$ が符号アルファベットです。符号アルファベット $\mathcal{B}$ の元の数が r のとき r-ary 符号と言われ、$\mathcal{B} = \{0, 1\}$ での符号はバイナリ (binary)、$\{0, 1, 2\}$ では ternary と呼ばれます。また、符号語の長さを符号長と言いますが、符号語の長さと言っていきます。

  英語では符号化関数と符号に同じ code の単語を当てているようですが、あまり実害はないです。日本語でも符号をどっちの意味で使っているのかはっきりしない場合があったりします。

  符号化関数を単射としているのは、$\phi(a) = \phi(a')$ となるのは $a = a'$ のときとすることで異なる $\mathcal{A}$ の元に同じ符号語が使われないようにするためです。

  $\mathcal{S}_A^*$ の文字列から $\mathcal{S}_B^*$ の文字列への変換の写像 ϕ' (英語では ϕ の extended と言われる) は

  $$\phi'(a_1 a_2 \cdots a_n) = \phi(a_1)\phi(a_2)\cdots\phi(a_n)$$

  として、個別に ϕ によって変換したものと等しいとします。このようにしているので、ϕ' は ϕ と書いてしまいます。

  例えば、$\mathcal{A} = \{x, y, z\},\ \mathcal{B} = \{0, 1\}$ とし、符号語を $\phi(x) = 0,\ \phi(y) = 10,\ \phi(z) = 11$ と与えると、符号は $\mathcal{C} = \{0, 10, 11\}$ となります。この変換を文字列にも適用すれば、$xyxz \in \mathcal{S}_A^*$ は $010011 \in \mathcal{S}_B^*$ となります
  モールス信号は $\mathcal{A} = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \ldots, \mathrm{Z}\}$ を「・」、「-」による文字列へと符号化していて、S,O と SOS は

  $$c(\mathrm{S}) = \cdots,\ c(\mathrm{O}) = ---$$

  $$c(\mathrm{SOS}) = c(\mathrm{S})c(\mathrm{O})c(\mathrm{S}) = \cdots --- \cdots$$

と変換されます。

簡単に言えば、符号理論で扱っているのは、日常的な言語を適切な符号に変換するにはどうすればいいのか、という問題です。適切な符号が指している内容の 1 つが、モールス信号のように、日常的な言語による文字列とそれを符号化した文字列とが一意的に対応するというものです。このことを見ていきます。まず、一意復号可能と prefix-free を定義します。

- 一意復号可能
  符号化した文字列をもとに戻すことを復号 (decoding) と言います。文字列に対する符号化関数 ϕ が単射 ($\mathcal{S}_A^*$ から $\mathcal{S}_B^*$ への写像が単射) になっているとき、ϕ は一意復号可能 (uniquely decodable) と呼ばれます。ϕ による符号化で作られた $\mathcal{S}_B^*$ の文字列が $\mathcal{S}_A^*$ の 1 つの文字列に対応するなら、ϕ は一意復号可能ということです。もっと簡単に言えば、符号化した文字列が必ずもとの文字列だけに対応するなら一意復号可能です。
  テキトーに設定したのでは一意的に戻せないことは、簡単な例から分かります。$A = \{x, y, z\},\ B = \{0, 1\}$ として

$$\phi(x) = 0,\ \phi(y) = 01,\ \phi(z) = 10 \tag{2}$$

  と与えたとします。$\mathcal{S}_B^*$ から 010100 を取り出して $\mathcal{S}_A^*$ の文字列に戻そうとしたとき、例えば

$$010100 = \phi(x)\phi(z)\phi(z)\phi(x) \ \Rightarrow\ xzzx\ ,\ 010100 = \phi(y)\phi(x)\phi(z)\phi(x) \ \Rightarrow\ yxzx \tag{3}$$

  として、2 つの場合が作れます (他にも作れる)。

- prefix-free
  復号の基本的な手順は文字列の先頭から $\phi(a)$ との対応を見ていくという流れになっています。このときに、全ての対応が一意的になっているなら、復号による文字列が 1 つに決まります。しかし、(2),(3) の例では先頭の 0 から x と見えますが、010 となっているために 0 と 10 から xz、01 と 0 から yx のどちらでもよくなってしまっています。これは $\phi(y)$ の先頭に $\phi(x)$ がいるために起きています。
  というわけで、このような状況にならないためには、異なる $\mathcal{A}$ の元 a, a' による $\phi(a) = c$ と $\phi(a') = c'$ $(c, c'q' \in \mathcal{S}_B^*)$ に対して

$$c' \neq cs \quad (s \in \mathcal{S}_B^*)$$

  となっている必要があります。これは各符号語が他の符号語を先頭に含んでいないという条件になっており、そのような符号語を作る符号化関数の性質を prefix-free や instantaneous と呼びます (prefix は文法用語での接頭辞)。例えば

$$\phi(a_1) = 00,\ \phi(a_2) = 01\ ,\ \phi(a_3) = 01011$$

  となっているなら、$\phi(a_3)$ の頭の 2 文字が $\phi(a_2)$ なので prefix-free ではないです。
  prefix-free の符号化関数は prefix-free code ですが、free を外して prefix code と呼んでいることもあります。また、日本語では語頭符号や接頭符号などいろいろな呼び方があり、あまり統一されていないです。
  おそらく code を符号化関数と符号語の集合のどちらにも使っているせいだと思いますが、prefix-free が符号化関数の性質としている場合と符号語の性質としている場合とがあります。どちらでも言っている内容は同じなので困らないですが、文脈が微妙に異なってくるので多少気にしておくといいです。ここでは prefix-free は符号化関数の性質として使い、符号語や符号では prefix-free の符号語のように言っていきます。

分かりやすい prefix-free の符号の例を 1 つ示しておきます。$B = \{0, 1\}$ から符号を

$$C = \{1,\ 01,\ 001, \ldots, 00\cdots01\} \tag{4}$$

と作ると、見て分かるように prefix-free の符号になり、単進符号 (アルファ符号、unary code) と呼ばれます。単進符号は同じものを並べて最後に別のものを加えた符号のことなので、0 と 1 をひっくり返して

$$C = \{0,\ 10,\ 110,\ \ldots, 11\cdots10\}$$

としたものも単進符号です。(4) は 0 を並べていき最後に 1 を置くとして作られており、その 1 がコンマのように見えます (符号語の終わりを 1 が与えている)。このように、符号語の最後に特定の文字列が置かれている符号はコンマ符号 (comma code) と呼ばれます。

prefix-free なら一意復号可能なことを示します。ϕ が prefix-free なら、符号語による 1 つの文字列がもとの 1 つの文字列に復号できることを示せばいいです。なので、$\mathcal{A}$ の元による文字列を 2 つ用意し、それらを

$$x_1x_2\cdots x_n\ ,\ y_1y_2\cdots y_m \quad (x_i, y_j \in \mathcal{A}\ ,\ n \neq m)$$

とし、これらを符号化したとき同じ文字列になるとします。符号化した $\mathcal{S}_B^*$ での文字列は

$$\phi(x_1x_2\cdots x_n) = \phi(y_1y_2\cdots y_m)$$
$$\phi(x_1)\phi(x_2)\cdots\phi(x_n) = \phi(y_1)\phi(y_2)\cdots\phi(y_m)$$

ϕ が prefix-free のとき

$$x_1x_2\cdots x_n = y_1y_2\cdots y_n \quad (n = m)$$

となるなら、$\mathcal{A}$ の元による 1 つの文字列に対応するので、一意的に復号されたことになります（文字列の写像として単射になっている)。

こうなることは単純に分かります。異なる $\mathcal{A}$ の元を変換しているので、一般的には $\phi(x_1) \neq \phi(y_1)$ です。しかし、prefix-free なら、文字列の最初に現れる $\phi(x_1), \phi(y_1)$ の先頭には他の符号語が含まれていません。このため、$\phi(x_1) = \phi(y_1)$ と分かり、$\mathcal{A}$ から $\mathcal{S}_B^*$ へは単射なので $x_1 = y_1$ となります。そして、$\phi(x_2), \phi(y_2)$ 以降も同様なので、

$$x_1x_2\cdots x_n = y_1y_2\cdots y_n \quad (n = m)$$

となり、ϕ は一意復号可能の定義を満たします。

英語で prefix-free を instantaneous とも言うのは、prefix-free では一意復号可能で、その符号語による文字列は左から読んでいけばすぐに復号できるからです (符号語の先頭に他の符号語が現れないために文字列を符号語に区切るのが簡単)。このため、prefix-free な符号化関数を日本語では瞬時復号可能と言ったりします。

ここから一意復号可能なら prefix-free となるための条件を求めます。条件を求めるのは一意復号可能なら prefix-free ではないからです。例えば、$00, 01$ は prefix-free ではないですが、一意復号可能です ((左からでなく右から読んでいくと判別できるから)。

まず、prefix-free となる符号化関数が存在するための条件から求めます。元の個数に関する表記から与えていきます。$\mathcal{A}$ の元から符号語を 0, 10, 110, 111 と作ったとします。これらの符号語の文字列の長さ l には $l = 1, 2, 3$ があります。1 個の符号語には 1 個の $\mathcal{A}$ の元が対応するので (単射だから)、長さが 1 での $\mathcal{A}$ の元の個数は 1 個、2 での個数は 1 個、3 での個数は 2 個です。このように符号語の長さ l に対応する $\mathcal{A}$ の元の個数を λ_l と表記します。今の例では $\lambda_1 = 1, \lambda_2 = 1, \lambda_3 = 2$ です。数学的に書けば、$\phi(a) \in \mathcal{S}_B^l$ $(a \in \mathcal{A})$ となる a の個数を λ_l とするということです。$\mathcal{A}$ の元の個数としていますが、符号語の個数と言っても同じです。

$\mathcal{B}$ の元の個数が β なら、作れる長さ l の符号語の個数は β^l 個です ($\mathcal{S}_B^l$ の元の総数)。これは単純に、例えば $\mathcal{B} = \{0, 1\}$ なら

$$00\ ,\ 01\ ,\ 10\ ,\ 11\ \Leftarrow\ 2^2 = 4$$

$$000\ ,\ 001\ ,\ 010\ ,\ 100\ ,\ 011\ ,\ 110\ ,\ 101\ ,\ 111\ \Leftarrow\ 2^3 = 8$$

となるからです。単射としているので、長さ l の符号語を作る $\mathcal{A}$ の元の数 λ_l が長さ l の $\mathcal{S}_B$ $(\mathcal{S}_B^l)$ の元の数 β_l を超えることはないために $\lambda_l \leq \beta_l$ です。

まずは $\mathcal{B} = \{0, 1\}$ として、prefix-free になるように符号語を作ってみます。$l = 1$ での符号語は $0, 1$ なので、$\lambda_1 = 1$ とし、$\phi(a_1) = 0$ としたとします。次に $l = 2$ の符号語を作ろうとすると、prefix-free にしたいので頭が 0 の符号語は使えません。なので、$l = 2$ では 1 から始まるようにしか作れなく、$10, 11$ のみなので、$\phi(a_2) = 10$ と選びます。これ以降も同様にしていくことで prefix-free として作れます。

この手順を一般化します。$l = 1$ のときに λ_1 個の符号語を作ったとすれば ($\mathcal{A}$ の λ_1 個の元から $l = 1$ の符号語を作る)、$l = 2$ での符号語を prefix-free にするためには最初の文字を $l = 1$ のときに選ばれなかった $\beta - \lambda_1$ 個の中から取り出すことになります。そして、$\mathcal{B}$ $(\mathcal{S}_B^1)$ の元は β 個なので、2 文字目に β 個の元から 1 個を選んで加えることで $l = 2$ の文字列になります。このため、$l = 2$ で prefix-free の符号語として使える文字列は $\beta(\beta - \lambda_1)$ 個あります。$l = 2$ では符号語を λ_2 個選んだとすれば、prefix-free にするためには $l = 3$ での文字列の最初の 2 文字は $l = 2$ で選ばれなかった $\beta(\beta - \lambda_1) - \lambda_2$ 個の中から選ぶことになります。そうすると、$l = 3$ での prefix-free の符号語は $\beta(\beta(\beta - \lambda_1) - \lambda_2)$ 個あります。

これが続くので、長さ l の符号語を λ_l 個選んだときに残る長さ l の文字列の個数 (長さ $l + 1$ の文字列の l 文字目までに使える文字列の個数) は

$$\begin{aligned}
l = 1\ ,\ \lambda_1 :&\ \beta - \lambda_1 \\
l = 2\ ,\ \lambda_2 :&\ \beta^2 - \beta\lambda_1 - \lambda_2 \\
l = 3\ ,\ \lambda_3 :&\ \beta^3 - \beta^2\lambda_1 - \beta\lambda_2 - \lambda_3 \\
&\vdots \\
l = i - 1\ ,\ \lambda_{i-1} :&\ F_{i-1} \\
l = i\ ,\ \lambda_i :&\ \beta F_{i-1} - \lambda_i \\
&\vdots
\end{aligned}$$

と書けます。F_{i-1} は

$$F_{i-1} = \beta^{i-1} - \beta^{i-2}\lambda_1 - \beta^{i-3}\lambda_2 - \cdots - \lambda_{i-1}$$

そして、$\lambda_i \leq \beta F_{i-1}$ なので

$$\lambda_i \leq \beta(\beta^{i-1} - \beta^{i-2}\lambda_1 - \beta^{i-3}\lambda_2 - \cdots - \lambda_{i-1})$$

この不等式は prefix-free になるように符号語を作ったら出てきたものなので、これを満たしているなら prefix-free になっている言えます。不等式は $l = L$ を長さの最大とすれば ($\phi(a)$ $(a \in \mathcal{A})$ の長さが L 以下)

$$\begin{aligned}
\lambda_L &\leq \beta^L - \beta^{L-1}\lambda_1 - \beta^{L-2}\lambda_2 - \cdots - \beta\lambda_{L-1} \\
\frac{\lambda_L}{\beta^L} &\leq 1 - \frac{\lambda_1}{\beta} - \frac{\lambda_2}{\beta^2} - \cdots - \frac{\lambda_{L-1}}{\beta^{L-1}} \\
\frac{\lambda_1}{\beta} + \frac{\lambda_2}{\beta^2} + \cdots + \frac{\lambda_L}{\beta^L} &\leq 1 \qquad (5)
\end{aligned}$$

と変形できます。これをクラフト (Kraft) の不等式と言います。というわけで、クラフトの不等式が成立しているなら、prefix-free な符号化関数が存在します。

クラフトの不等式は左辺を別の形にしていることが多いので、それに書き換えます。(5) の左辺を

$$K = \frac{\lambda_1}{\beta} + \frac{\lambda_2}{\beta^2} + \cdots + \frac{\lambda_L}{\beta^L} = \sum_{l=1}^{L} \frac{\lambda_l}{\beta^l} \qquad (6)$$

と表記します。λ_l を全て足したものを

$$\alpha = \lambda_1 + \lambda_2 + \cdots + \lambda_L$$

λ_l は長さ l の符号語の数なので、α は符号語の総数です ($\mathcal{A}$ の元の総数)。なので、$\mathcal{S}_B^*$ での各符号語を $\phi_1, \phi_2, \ldots, \phi_\alpha$ とすれば、それぞれの長さを $l_1, l_2, \ldots, l_\alpha$ と与えられます。このように設定すると、例えば長さ 3 の符号語が ϕ_2, ϕ_5 の 2 個 ($\lambda_3 = 2$)、長さ 5 の符号語が $\phi_5, \phi_{10}, \phi_{12}$ の 3 個 ($\lambda_5 = 3$) であるなら

$$\begin{aligned}
\frac{\lambda_3}{\beta^3} &= \frac{1}{\beta^{l_2}} + \frac{1}{\beta^{l_5}} = \frac{1}{\beta^3} + \frac{1}{\beta^3} = \frac{2}{\beta^3} \\
\frac{\lambda_5}{\beta^5} &= \frac{1}{\beta^{l_5}} + \frac{1}{\beta^{l_{10}}} + \frac{1}{\beta^{l_{12}}} = \frac{1}{\beta^5} + \frac{1}{\beta^5} + \frac{1}{\beta^5} = \frac{3}{\beta^5}
\end{aligned}$$

このようになっていることから

$$K = \sum_{i=1}^{\alpha} \frac{1}{\beta^{l_i}}$$

となり、クラフトの不等式は

$$\sum_{i=1}^{\alpha} \frac{1}{\beta^{l_i}} \leq 1 \qquad (7)$$

と書けます。

クラフトの不等式が成立するなら prefix-free な符号化関数が存在することが分かったので、次に文字列の個数から一意復号可能となる条件を求めます。

文字列の個数を表す表記を作ります。$\mathcal{A}$ の元から符号化関数 ϕ によって符号語が与えられており、各長さの符号語に対して $\mathcal{A}$ の元の個数が $\lambda_1, \lambda_2, \ldots, \lambda_L$ となっているとします。$\mathcal{A}$ の元から作られる文字列 ($\mathcal{S}_A^*$ の元) の長さを r、$\mathcal{A}$ の元から作られる符号語の最大の長さを L とすれば、符号語の取れる最小の長さは 1 であることから、$\mathcal{S}_A^*$ の長さ r の文字列の符号化は $\mathcal{S}_B^*$ での長さ m $(1 \leq m \leq rL)$ の文字列を作ります。符号化によって長さ m の文字列になる $\mathcal{S}_A^*$ の長さ r の文字列の個数を $q_r(m)$ とします。$r = 1$ のときは $\mathcal{A}$ の元なので、$q_1(m)$ は長さ m の符号語の個数 λ_m です。

一意復号可能は、符号化された文字列を 1 つのもとの文字列に戻せることです。この対応関係は、もとの文字列の個数が符号化された文字列の個数を上回っていると成立しません (1 つの符号化された文字列が複数の文字列に対応してしまう)。これによる条件を作ります。

今の話から、一意復号可能なら $\mathcal{S}_A^*$ の文字列は $\mathcal{S}_B^*$ に含まれる文字列にそれぞれが対応しないといけないので、$q_r(m)$ は $\mathcal{S}_B^m$ に含まれる文字列の個数を超えることはないです。超えてしまっていたら、いくつかの文字列の対応が重複してしまい、一意的な復号ができなくなります。$\mathcal{S}_B^m$ に含まれる文字列の個数は $\mathcal{B}$ の元の個数 β から β^m なので、$q_r(m) \leq \beta^m$ です。この不等式を (5) に関連付けます。

$q_1(m) = \lambda_m$ なので

$$K = \frac{\lambda_1}{\beta} + \frac{\lambda_2}{\beta^2} + \cdots + \frac{\lambda_L}{\beta^L} = q_1(1)\frac{1}{\beta} + q_1(2)\frac{1}{\beta^2} + \cdots + q_1(L)\frac{1}{\beta^L}$$

と書けます。これは多項式になっているので

$$Q_1(x) = q_1(1)x + q_1(2)x^2 + \cdots + q_1(L)x^L$$

として、関数 Q_1 を作ります。これを $q_r(m)$ に拡張して

$$Q_r(x) = q_r(1)x + q_r(2)x^2 + \cdots + q_r(rL)x^{rL}$$

rL までにしているのは $1 \leq m \leq rL$ だからです。Q_r は下の補足で示しているように

$$Q_r(x) = Q_1^r(x) \tag{8}$$

という関係を持っています。

Q_r を

$$Q_r(\frac{1}{\beta}) = q_r(1)\frac{1}{\beta} + q_r(2)\frac{1}{\beta^2} + \cdots + q_r(rL)\frac{1}{\beta^{rL}}$$

とすると、$q_r(m) \leq \beta^m$ から各項は 1 以下です。項の数は rL で、各項が 1 以下になっていることから

$$Q_r(\frac{1}{\beta}) \leq rL$$

$r = 1$ のとき $q_1(m) = \lambda_m$ なので

$$Q_1(\frac{1}{\beta}) = \frac{\lambda_1}{\beta} + \frac{\lambda_2}{\beta^2} + \cdots + \frac{\lambda_L}{\beta^L}$$

(8) を使うと

$$Q_r(\frac{1}{\beta}) = Q_1^r(\frac{1}{\beta}) = (\frac{\lambda_1}{\beta} + \frac{\lambda_2}{\beta^2} + \cdots + \frac{\lambda_L}{\beta^L})^r = K^r \leq rL \tag{9}$$

となっているのが分かります。というわけで、一意復号可能であるためには、文字列の個数による不等式 $K^r \leq rL$ を満たす必要があります。

この不等式は K の不等式に変えられます。二項定理から

$$(1+y)^r = 1 + ry + \frac{1}{2}r(r-1)y^2 + \cdots$$

となっていることを利用します。$y > 0$ ならこれの各項は正なので、第 2 項だけを取り出すと

$$(1+y)^r > \frac{1}{2}r(r-1)y^2$$
$$\frac{(1+y)^r}{r} > \frac{1}{2}(r-1)y^2$$

これは $1 + y = K$ なら $K/r \leq L$ に対応させられます。対応させるために

$$K = 1 + y > 1 \quad (y > 0)$$

として、条件 (9) と合わせると

$$\frac{1}{2}(r-1)y^2 < \frac{K^r}{r} \leq LK = 1 + y > 1 \quad (y > 0)$$

しかし、A の元による文字列の長さ r には制限がないので、$(r-1)x^2/2$ が L を超えることができてしまいます。なので、$K > 1$ では不等式が成立しなくなるので、一意復号可能であるなら $K \leq 1$ でないといけないことが分かります。というわけで、$\lambda_1, \lambda_2, \ldots, \lambda_L$ (もしくは $l_1, l_2, \ldots, l_\alpha$) の符号語を持つ一意復号可能な符号関数が存在するなら $K \leq 1$ が成立する、となります。

$K \leq 1$ が一意復号可能と prefix-free の両方で出てきているので合わせられます。これらは、一意復号可能な符号化関数が存在するなら $K \leq 1$、$K \leq 1$ なら prefix-free な符号化関数が存在する、となっていて、$K \leq 1$ が 2 つの間を繋いでいます。なので、$\lambda_1, \lambda_2, \ldots, \lambda_L$ $(l_1, l_2, \ldots, l_\alpha)$ の符号語を持つ一意復号可能な符号化関数が存在すれば、同じ $\lambda_1, \lambda_2, \ldots, \lambda_L$ $(l_1, l_2, \ldots, l_\alpha)$ での prefix-free な符号化関数が存在する、となります。そして、prefix-free なら一意復号可能でもあるので必要十分条件になっています。

まとめると、$K \leq 1$ を満たすようにして prefix-free な符号化関数を作れば、それがそのまま一意復号可能になるということです。一意復号可能なら $K \leq 1$ はマクミラン (McMillan) が示したので、$K \leq 1$ をクラフト-マクミランの不等式と言うこともあります。

例として、$B = \{0, 1\}$ とし、長さが $1, 2, 3, 4, 5$ の符号語がそれぞれ 1 個ずつあるとします。このときのクラフトの不等式は

$$\frac{1}{2}+\frac{1}{2^2}+\frac{1}{2^3}+\frac{1}{2^4}+\frac{1}{2^5}=\frac{31}{32}<1$$

として成立するので、prefix-free な符号化関数が存在します。実際に、5 個の符号語は単進符号

$$\mathcal{C}=\{0,\ 10,\ 110,\ 1110,\ 11110\}$$

として作れます。

・補足

$Q_r(x)$ を

$$Q_r(x)=q_r(1)x+q_r(2)x^2+\cdots+q_r(rL)x^{rL}$$

と定義します。$r=1$ のとき

$$Q_1(x)=q_1(1)x+q_1(2)x^2+\cdots+q_1(L)x^L=\lambda_1x+\lambda_2x^2+\cdots+\lambda_Lx^L$$

これの 2 乗は

$$\begin{aligned}Q_1^2(x)&=(\lambda_1x+\lambda_2x^2+\cdots+\lambda_Lx^L)^2\\&=\lambda_1^2x^2+(\lambda_1\lambda_2+\lambda_2\lambda_1)x^3+(\lambda_2^2+\lambda_1\lambda_3+\lambda_3\lambda_1)x^4+(\lambda_1\lambda_4+\lambda_4\lambda_1+\lambda_2\lambda_3+\lambda_3\lambda_2)x^5+\cdots+\lambda_L^2x^{2L}\\&=\sum_{(i+j=2)}\lambda_i\lambda_jx^2+\sum_{(i+j=3)}\lambda_i\lambda_jx^3+\sum_{(i+j=4)}\lambda_i\lambda_jx^4+\sum_{(i+j=5)}\lambda_i\lambda_jx^5+\cdots+\sum_{(i+j=2L)}\lambda_i\lambda_jx^{2L}\end{aligned}\tag{10}$$

$(i+j=2)$ のように書いているのは、$i+j=2$ になる i,j の組み合わせに対して和を取り、$i=j$ では二重に和を取らないという意味にしています。

何を知りたいのかをはっきりさせるために、$r=2$ の場合から見ていきます。$\mathcal{A}$ の元による文字列を a_1a_2 と作り、これを $\phi(a_1)\phi(a_2)$ に変換したとします。$\mathcal{A}$ の元による $\phi(a)$ の長さは 1 以上としていることから、$\phi(a_1)\phi(a_2)$ の長さが 1 になることはないので、$q_2(1)=0$ です。$q_2(2)$ は $\phi(a_1),\phi(a_2)$ がどちらも長さ 1 になる a_1,a_2 による組み合わせの数なので、長さ 1 の符号語を作る $\mathcal{A}$ の元の個数 λ_1 から、$q_2(2)=\lambda_1^2$ です (a_1 として選べる個数が λ_1 個、a_2 では λ_1 個)。$q_2(3)$ では $\phi(a_1),\phi(a_2)$ のどちらかの長さが 1 か 2 になる a_1,a_2 による組み合わせの数なので、長さ 2 の符号語を作る A の元の個数 λ_2 から $q_2(3)=\lambda_1\lambda_2+\lambda_2\lambda_1=2\lambda_1\lambda_2$ です。これらから、$q_2(l)$ は A の元から長さ l になる 2 個の組み合わせを取り出す個数になっているのが分かります。なので、$i+j=l$ $(i,j=1,2,\ldots,L)$ として、(10) と同じ表記を使えば

$$q_2(l)=\sum_{(i+j=l)}\lambda_i\lambda_j$$

そうすると、(10) と

$$Q_2(x)=q_2(2)x^2+q_2(3)x^3+q_2(4)x^4+\cdots+q_2(2L)x^{2L}$$

を見比べることで、x^l の係数が一致しているのが分かります。よって

$$Q_2(x) = Q_1^2(x)$$

という関係になっています。
　今の話を

$$Q_1^r(x) = (\lambda_1 x + \lambda_2 x^2 + \cdots + \lambda_L x^L)^r$$

に一般化します。これを展開すると各項は r 個の積になるので、x^M の項は

$$\Big(\sum_{(i_1+i_2+\cdots+i_r=M)} \lambda_{i_1}\lambda_{i_2}\cdots\lambda_{i_r} \Big) x^{i_1+i_2+\cdots+i_r} \quad (i_1+i_2+\cdots+i_r = M)$$

となっています (より正確な形は多項定理として与えられている)。(10) での表記と同じように、$(i_1+i_2+\cdots+i_r = M)$ も $i_1 = i_2 = \cdots = i_r$ のときは 2 重に和を取らないとしています。一方で、$q_r(m)$ は $a_1a_2\cdots a_r$ から長さ m の $\phi(a_1)\phi(a_2)\cdots\phi(a_r)$ になる $a_1, a_2, \ldots, a_r$ の組み合わせの個数なので、符号語の長さが $i_1, i_2, \ldots, i_r$ $(i_1+i_2+\cdots+i_r = m)$ になる $\mathcal{A}$ の元の個数 $\lambda_{i_1}, \lambda_{i_2}, \ldots, \lambda_{i_r}$ から

$$q_r(m) = \sum_{(i_1+i_2+\cdots+i_r=m)} \lambda_{i_1}\lambda_{i_2}\cdots\lambda_{i_r}$$

よって、$Q_1^r(x)$ と

$$Q_r(x) = q_r(1)x + q_r(2)x^2 + \cdots + q_r(rL)x^{rL}$$

での x^l の係数は等しいことが分かり

$$Q_r(x) = Q_1^r(x)$$

となります。