Vol.2017-AL-162 No.1
2017/3/13

高階圧縮における連続パターンのコンパクトな表現法

古谷 勇[1]　喜田 拓也[2]

概要：高階圧縮とは，入力データを表すラムダ式を構築し，それを符号化することで行う圧縮法である．本論文では，同じパターンが連続する部分に対して，コンパクトなラムダ式を生成する手法を提案する．提案手法では，パターンの連続回数を表す自然数（連長）を超冪の線形式に分解し，その式に対応するラムダ式を直接出力する．自然数 n に対して，出力されるラムダ式のサイズが $O(\log n)$ であることを証明した．また，実験の結果，連長をラムダ式のバイナリ表現で置き換える既存手法に比べ，ラムダ式のサイズを平均して約 21% 低減することを確認した．

1. はじめに

データ圧縮は，入力データ系列のモデル化と構築されたモデルの符号化の二つの処理からなる．例えば，ランレングス法 [5] では，入力データ系列中の記号とそれが連続する長さ（**連長**）でモデル化し，記号と数字の組をそれぞれに適当な符号で符号化を行う．Huffman 符号 [2] では，入力データ系列を記憶のない情報源としてモデル化し，各記号の頻度に従った符号化を行う．

入力データに対するより良いモデル化は，入力データをより良く圧縮する．入力データのモデルが既知であるならば，それを最適に圧縮することは容易である．しかし現実には，入力データのモデルが既知であるような場合は稀である．実際には，未知の情報源に対して漸近的に最適となるユニバーサル符号 [5] を用いるか，適当なモデルを仮定してデータに内在する構造を表現することになる．したがって，有限長の入力データを効率よく圧縮するためには，簡潔かつ十分な表現力を持つモデル化の方法が求められる．

この問題に対し，小林ら [4] は高階プログラムをモデルとするデータ圧縮を提案している．彼らの手法では，入力データに対し，それを生成する高階プログラムを構築することでデータ圧縮を行う．ここで高階プログラムとしては，**ラムダ式** [1] を用いている．これにより，複雑な構造を持つデータに対して詳細なモデル化を行うことができる．単純な例としては，$a\overbrace{2^{\cdots^2}}^{n}c$ のように，繰り返しが超冪をなす構造を

$$(\lambda f.\overbrace{f\ f\ \cdots\ f}^{n}\ a\ c)(\lambda g.\lambda x.g(g(x)))$$

のように，冪の深さの線形長のラムダ式で表現できる．他の手法，例えば文脈自由文法をモデルとする文法圧縮 [3] で同じ文字列を表現すると，$\overbrace{2^{\cdots^2}}^{n-1}$ に比例した長さが必要となる．これ以外にも，系列としての見かけは違うが同じ差分で増減するデータや，ほとんど同じだが一部が異なるパターンなども柔軟に表現することができる．

高階プログラムによる圧縮の課題点は，効率よい圧縮処理と符号化である．近年，矢口ら [6] は，高階プログラムをモデルとするデータ圧縮法を**高階圧縮**と呼び，その効率よい圧縮処理と符号化を提案している．彼らの手法では，効率よい符号化と圧縮処理のために**単純型付き**のラムダ式を用いている．圧縮処理の方針は，まずデータを単に直線的なラムダ式として表現し，そのラムダ式に対応する構文木を考える．そして，構文木内で共通して頻出する部分木をパターンとして抽出し，それらをまとめるようにラムダ式を変形する．この変形操作は，ラムダ計算における β 簡約の逆の処理に相当する．実験の結果より，矢口らの手法は，既存の手法と同程度以上の優れた圧縮率を達成することが示されている．

本論文では，連続するパターンに対して，よりコンパクトなラムダ式を構築する手法を提案する．提案手法は，連長を非負整数の超冪を用いた式に分解し，その式に対応するラムダ式へと置き換えることでデータを圧縮する．分解アルゴリズムを示すとともに，この表現方法によるラムダ式のサイズが連長 n に対して $O(\log n)$ であることを証明する．提案手法は，矢口らの圧縮手法に容易に組み込むことができる．

[1] 北海道大学工学部
Hokkaido University, School of Engineering,
flare@eis.hokudai.ac.jp
[2] 北海道大学大学院情報科学研究科
Hokkaido University, Graduate School of Information Science and Technology,
kida@ist.hokudai.ac.jp

既存のラムダ式による連長の表現手法としては，Kobayashi ら [4] がバイナリ表現に対応するものを示している．提案手法は，ほとんどの場合において，このバイナリ表現よりも小さなラムダ式を生成する．実験の結果，提案手法は，バイナリ表現より平均して約 21% 圧縮率が向上することを確認した．

2. 準備

2.1 ラムダ式

定義 1 (ラムダ式) 圧縮対象の入力データに含まれる文字の集合を A とし，以降では $a \in A$ を**終端文字**と呼ぶ．いま，終端文字を含まない有限アルファベット Σ を考える．ただし，Σ は，ラムダ式で用いる特殊な文字 $\{\lambda, . , (,) \}$ を含まないものとする．以降，文字 $x \in \Sigma$ を特に**変数**と呼ぶ．このとき，ラムダ式は次のように再帰的に定義される．すなわち，$a \in A$, $x \in \Sigma$, ラムダ式 M, N に対して，

$$\text{(i) } x \qquad \text{(ii) } (\lambda x.M) \qquad \text{(iii) } (M\ N) \qquad \text{(iv) } a$$

はすべてラムダ式である． □

上のラムダ式の定義において，特に (ii) の形をしたラムダ式を**ラムダ抽象**と呼び，(iii) の形をしたラムダ式を**関数適用**と呼ぶ．ラムダ抽象 $(\lambda x.M)$ に関して，変数 x はこのラムダ式の中で**束縛されている**という．ラムダ抽象 $(\lambda x.M)$ の中に束縛された形で現れる変数を $(\lambda x.M)$ の**束縛変数**，束縛されない形で現れる変数を $(\lambda x.M)$ の**自由変数**という．

ラムダ式が複雑になると，括弧が頻出して煩雑になる．以降では，特に混乱が起こらない限り，ラムダ式の括弧を省略して表記する．括弧を省略したラムダ式における結合の解釈は以下のものを用いる．

- 関数適用は左の結合を優先する．
- 関数適用とラムダ抽象では関数適用を優先する．

さらに，ラムダ抽象 $\lambda x.N$ の N がラムダ抽象 $\lambda y.M$ である場合，すなわち $\lambda x.(\lambda y.M)$ のような形のラムダ抽象について，これを $\lambda xy.M$ と略記する．これにより，$\lambda x_1.\lambda x_2.\cdots\lambda x_n.M$ は $\lambda x_1 x_2 \cdots x_n.M$ と表される．

次にラムダ式の書き換え規則を定義する．α **変換**は，変数の名前を付け替える規則で，次のようなものである．

$$\lambda x.M \to^{\alpha} \lambda y.M'$$

ここで，M' はラムダ式 M 中のすべての x を，M 中に自由変数として表れていない文字 $y \in \Sigma$ に置換したものである．α 変換によって得られるラムダ式はすべて同一のものと見なす．

β **簡約**は関数適用を書き換える規則で，次のようになる．

$$(\lambda x.M)\ N \to^{\beta} M'$$

ここで，M' は M 中に出現するすべての x を N に置換したラムダ式である．

β 簡約において，$(\lambda x.M)\ N$ 中の $\lambda x.M$ と N は，それぞれ β 簡約後のラムダ式を返す関数およびその引数と見なすことができる．このことから，$(\lambda x.M)\ N$ における N をしばしば**引数**と呼ぶ．また，ラムダ式を用いた計算とは，上述の書き換え規則にしたがってラムダ式を順次書き換えていき，それ以上 β 簡約が不可能な形を目指すことをいう．

2.2 高階圧縮の考え方

高階圧縮では，最初に，入力データを終端文字の関数適用が繰り返されたラムダ式として扱う．例えば，$abcde$ という文字列は，$a\ (b\ (c\ (d\ e)))$ というラムダ式と見なされる[*1]．次にこのラムダ式を，ラムダ計算後の形が同一になるように保ったまま，よりサイズの小さいラムダ式へと変換する．ここで，ラムダ式のサイズとは，次のようにして定義される．

定義 2 (ラムダ式のサイズ（小林ら [4]）) ラムダ式 M の**サイズ**を次のように再帰的に定義し，$\#M$ と記述する．すなわち，$a \in A$, $x \in \Sigma$, ラムダ式 M, N に対して，

$$\begin{aligned} \#x &= \#a = 1, \\ \#(\lambda x.M) &= \#M + 1, \\ \#(M\ N) &= \#M + \#N + 1. \end{aligned}$$

と定義する． □

このようにサイズ $\#M$ を定義すると，ラムダ式を構文木で表現した場合の木のノード数に等しくなることが知られている [6]．今回，構文木上での圧縮手法については触れないため，構文木の定義は省略する．

2.3 チャーチ数と連続パターン

定義 3 (チャーチ数) 非負整数 n に対して，

$$n \Leftrightarrow \lambda f x.\overbrace{f\ (f \cdots (f}^{n}\ x) \cdots).$$

とラムダ式を対応付ける．この n に対応づけられたラムダ式を**チャーチ数**と呼び，$\Lambda(n)$ と記述する． □

例えば，0 と 1 に対しては，それぞれ $\Lambda(0) = \lambda f x.x$ と $\Lambda(1) = \lambda f x.f\ x$ となる．

チャーチ数に対してそれらの加算，乗算，冪乗を表すラムダ式が次のように表せることが知られている．

定理 (チャーチ数の演算) 自然数 p, q に対して，以下の式が成り立つ．

$$\begin{aligned} &\text{（加算）} \quad \Lambda(p+q) = (\lambda mnfx.m\ f\ (n\ f\ x))\ \Lambda(p)\ \Lambda(q), \\ &\text{（乗算）} \quad \Lambda(p \cdot q) = (\lambda mnfx.m\ (n\ f)\ x)\ \Lambda(p)\ \Lambda(q), \\ &\text{（冪乗）} \quad \Lambda(p^q) = (\lambda mnfx.n\ m\ f\ x)\ \Lambda(p)\ \Lambda(q). \end{aligned}$$

[*1] 入力文字列に対して関数適用の順序は様々に考えられる．どのような順序で括弧付けしてもラムダ式のサイズに違いはない．圧縮処理の都合上，ここでは後ろから括っていくもののみを考える．

□

チャーチ数 $\Lambda(n)$ の引数として終端文字 a と終端文字 c を取ると，ラムダ計算によって文字列 $a^n c$ が得られる．例えば，$\Lambda(3)$ に引数として a, c を取ったものを計算すると，

$$(\lambda fx.f(f(f\ x)))\ a\ c \to a\ (a\ (a\ c))$$

となり，文字列 $aaac$ が得られる．ここで，チャーチ数の引数として終端文字の代わりにそれぞれ終端文字列，すなわちパターンを取ったラムダ式を計算しても，同様にひとつめのパターンがチャーチ数の回数だけ連続する文字列が得られる．終端文字が連続する場合を考えることで，それを直ちにパターンが連続する場合にも応用することができる．

2.4 連続パターンに対するバイナリ表現を用いた方法

連続パターンに対応するラムダ式の表現手法として，Kobayashi ら [4] はバイナリ表現を示している．例えば，$a^{57}c$ という文字列に対応するラムダ式を次のようにして表現する．

$$b_1(b_0(b_0(b_1(b_1(b_1(\Lambda(0)))))))\ a\ c.$$

ただし，b_0 と b_1 はそれぞれ，

$$b_0 = \lambda fpq.f\ p\ (f\ p\ q),$$
$$b_1 = \lambda fpq.p\ (f\ p\ (f\ p\ q))$$

である．ここで，最初の式における $b_1b_0b_0b_1b_1b_1$ という部分は，57 のバイナリ表現である 111001 を逆順に並べたものになっている．

3. 提案手法

3.1 基本アイデア

チャーチ数の表す数が大きい場合，チャーチ数をそのまま書くよりも，計算結果がそのチャーチ数となる演算の形に分解して書いたほうが，ラムダ式のサイズが小さくなる場合がある．例えば，ラムダ式

$$(\lambda mnfx.m(n\ f)x)\ \Lambda(6)\ \Lambda(5) \qquad (1)$$

は $6\cdot5$ を表すが，これを計算することで $\Lambda(30)$ が得られる．$\#\Lambda(30) = 63$ であるのに対し，(1) のサイズは 41 である．

$a^{30}c$ を表すラムダ式 $\overbrace{a\ (a \cdots (a}^{30}\ c) \cdots)$ のサイズは 61 である．一方，式 (1) の後ろに引数として a, c を取ったラムダ式のサイズは 45 である．このように，$\Lambda(30)$ の代わりに式 (1) を使うことで，よりコンパクトに連続パターンを表現することができる．

3.2 チャーチ数の圧縮表現

次の例のように，加算，乗算，冪乗のラムダ式を組み合わせて，複雑な演算式をラムダ式で表現することができる．

例 1

$$(\lambda mnpfx.(m\ (n\ f)\ f)\ (p\ f\ x))\ \Lambda(2)\ \Lambda(3)\ \Lambda(6) \qquad (2)$$
$$\cdots\ 2\cdot3+6 \text{ を表すラムダ式,}$$

$$(\lambda mnfx.m\ (n\ f\ (n\ f))\ x)\ \Lambda(3)\ \Lambda(2) \qquad (3)$$
$$\cdots\ 3\cdot(2+2) \text{ を表すラムダ式.}$$

以降では，このように自然数 n を分解した演算式を

(ラムダ抽象) (チャーチ数) $\cdots$ (チャーチ数)

という形のラムダ式で表現したものを n の**チャーチ演算表現**と呼ぶ．また，チャーチ演算表現における前半の「(ラムダ抽象)」を**関数部**，後半の「(チャーチ数) $\cdots$ (チャーチ数)」を**引数部**と呼ぶ．自然数 n の分解式 $F[n]$ に対するチャーチ演算表現を $\mathcal{C}(F[n])$ と書く．n を分解して得られる式は複数存在するので $\mathcal{C}(F[n])$ も複数存在する．例えば，式 (2) と (3) は共に 12 を表すチャーチ演算表現である．

$\#\mathcal{C}(F[n])$ は可能な限り小さくなるのが望ましいが，一般にサイズが最小となる $\mathcal{C}(F[n])$ を求めるのは難しい．そこで，引数部のチャーチ数を一つに限定し，その中でサイズが最小となる $\mathcal{C}(F[n])$ を求めることを考える．

定義 4 (剰余内包チャーチ演算表現) 自然数 n に対して，φ を n 未満のある自然数とする．ここで，$r = n \bmod \varphi$，$\bar{n} = n - r$ とおく．$\bar{n}$ を φ のみの加算，乗算，冪乗からなる式に分解したものを $F_\varphi[\bar{n}]$ とし，この分解に対応するチャーチ演算表現 $\mathcal{C}(F_\varphi[\bar{n}])$ の関数部における末尾の束縛変数 x を $\overbrace{f\ (f \cdots (f}^{r}\ x) \cdots)$ で置き換えると，$F_\varphi[\bar{n}] + r$ に対応するチャーチ演算表現が得られる．これを n の**剰余内包チャーチ演算表現**と呼び，$\overline{\mathcal{C}}(F_\varphi[\bar{n}], r)$ で表す． □

チャーチ演算表現の場合と同様に，剰余内包チャーチ演算表現 $\overline{\mathcal{C}}(F_\varphi[\bar{n}], r)$ も計算することでチャーチ数 $\Lambda(n)$ となる．

ここで，$\overline{\mathcal{C}}(F_\varphi[\bar{n}], r)$ のサイズに関して，次の補題が成り立つ．

補題 1 自然数 n と φ $(\varphi < n)$ について，$r = n \bmod \varphi$，$\bar{n} = n - r$ とおく．このとき，$F_\varphi[\bar{n}]$ 中に出現する加算，乗算，冪乗の回数をそれぞれ N_p，N_m，N_e とすると，

$$\#\overline{\mathcal{C}}(F_\varphi[\bar{n}], r) = 2((2N_p + N_m + N_e) + \varphi + 6 + r)$$

が成り立つ．

証明 引数部にチャーチ数 $\Lambda(\varphi)$ をもつ剰余内包チャーチ演算表現 $\overline{\mathcal{C}}(F_\varphi[\bar{n}], r)$ の構造は，

$$\overline{\mathcal{C}}(F_\varphi[\bar{n}], r) = (\lambda nfx.M)\ \Lambda(\varphi)$$

である．チャーチ数の定義より $\#\Lambda(\varphi) = 2\varphi + 3$ である．したがってこのラムダ式のサイズは，

$$\begin{aligned}\#\overline{\mathcal{C}}(F_\varphi[\bar{n}],r) &= 3+\#M+(\#\Lambda(\varphi)+1)\\ &= \#M+2\varphi+7 \qquad (4)\end{aligned}$$

となる．次に M について考える．M はまず次のような構造をもつ．

$$M = p\,f\,X \qquad (5)$$

ここで，X は $\overbrace{f\,(f\cdots(f}^{r=n \bmod \varphi}\ x)\cdots)$ というラムダ式である．よって，M のサイズは $2r+5$ である．p は演算される数としての $\Lambda(\varphi)$ に対応する束縛変数である．ここに演算する数としての $\Lambda(\varphi)$ に対応する束縛変数として p をとった加算の表現を加えたものは $p\,f\,(p\,f\,X)$ となり，式 (5) と比べるとラムダ式のサイズが 4 だけ大きい（括弧はラムダ式のサイズには無関係であることに注意する）．同様に乗算と冪乗を考えると，それぞれ $p\,(p\,f)\,X$ と $p\,p\,f\,X$ となり，いずれもラムダ式のサイズは 2 だけ増える．したがって，

$$\#M = 5+4N_p+2N_m+2N_e+2r$$

である．式 (4) とあわせると，

$$\#\overline{\mathcal{C}}(F_\varphi[\bar{n}],r) = 2((2N_p+N_m+N_e)+\varphi+6+r)$$

となり，命題が成り立つ． □

補題 2 任意の自然数 $n \le 8$ に対して，$\#\Lambda(n) \le \#\overline{\mathcal{C}}(F_\varphi[\bar{n}],r)$ が成り立つ．

証明 補題 1 より，

$$\begin{aligned}&\#\overline{\mathcal{C}}(F_\varphi[\bar{n}],r) =\\ &\quad 2((2N_p+N_m+N_e)+\varphi+6+(n \bmod \varphi))\\ &\quad > 2(1+\varphi+6)\\ &\quad \ge 2(1+2+6) = 18\end{aligned}$$

である．$n \le 7$ のとき，$\#\Lambda(n) = 2n+3 < 18$ である．$n = 8$ のときも，$\#\mathcal{C}_2(n) = 20$，$\#\mathcal{C}_3(n) = 26$，$\#\mathcal{C}_4(n) = 24$ となり，いずれも $\#\Lambda(8) = 19$ より大きい．したがって $n \le 8$ に対して，$\#\Lambda(n) > \#\overline{\mathcal{C}}(F_\varphi[\bar{n}],r)$ となるような分解表現 $\overline{\mathcal{C}}(F_\varphi[\bar{n}],r)$ は存在しない． □

3.3 チャーチ数の超冪分解表現

$\overbrace{\varphi^{\varphi^{\cdots^{\varphi}}}}^{i}$ のような演算を超冪といい，ここでは ${}^i\varphi$ と表記する．冪乗は右結合で解釈される．例えば，${}^1 2 = 2, {}^2 2 = 4, {}^3 2 = 16, {}^4 2 = 65536$ である．便宜上，${}^0\varphi = 1$ とする．

自然数 n を，自然数 φ（$\varphi < n$）の超冪 ${}^i\varphi$ の線形和に分解することを考える．すなわち，k を ${}^k\varphi \le n$ を満たす最大の整数としたとき，p_i（$0 \le i \le k$）を非負整数の係数として，

$$n = p_k\ {}^k\varphi + p_{k-1}\ {}^{k-1}\varphi + \cdots + p_1\ {}^1\varphi + p_0\ {}^0\varphi$$

のように分解する．次に，この分解式の各項について，$r_i = p_i \bmod \varphi$ とおくと，

$$p_i\ {}^i\varphi = {}^i\varphi\,(p_i - r_i) + (\overbrace{{}^i\varphi + \cdots + {}^i\varphi}^{r_i})$$

と変形できる．ここで $\bar{p_i} = p_i - r_i$ とおき，さらに $\bar{p_i}$ を φ で同様にして分解する．この操作を再帰的に行うと，n を φ のみの加算，乗算，冪乗の式に分解することができる．

定義 5 (φ による超冪分解表現) n, φ を自然数とし，$r = n \bmod \varphi$ とおく．k を ${}^k\varphi \le n$ を満たす最大の整数としたとき，n を次のように再帰的に分解して表すことを φ による超冪分解表現といい，その式を $T_\varphi[n]$ で表す．

$$T_\varphi[n] = \begin{cases} n & (n \le \varphi \text{ の場合}),\\ {}^k\varphi T_\varphi[p_k - (p_k \bmod \varphi)] + (\overbrace{{}^k\varphi + \cdots + {}^k\varphi}^{p_k \bmod \varphi}) & \\ \quad + \cdots & \\ \quad + {}^1\varphi T_\varphi[p_1 - (p_1 \bmod \varphi)] + (\overbrace{{}^1\varphi + \cdots + {}^1\varphi}^{p_1 \bmod \varphi}) + r & \\ & (\text{それ以外の場合}). \end{cases}$$

ただし，p_i（$0 \le i \le k$）は，$0 \le p_i < {}^i\varphi$ をみたす最大の整数とする． □

定義 5 において，係数 $(p_i - (p_i \bmod \varphi))$ は必ず φ の倍数である．よって，係数を超冪に分解する際の剰余は必ず 0 となり，項として出現しない．したがって，$T_\varphi[n]$ は末尾の項 r 以外がすべて φ の加算，乗算，冪乗のみで表現される．以降，$T_\varphi[n]$ の末尾の項 r を取り除いた式を $\overline{T}_\varphi[n]$ と書く．与えられた φ に対して，上式にしたがって超冪分解される式 $T_\varphi[n]$ は一つに定まる．この $T_\varphi[n]$ に対応するチャーチ演算表現は，剰余内包チャーチ演算表現の一つであるので，これを $\overline{\mathcal{C}}(\overline{T}_\varphi[n],r)$ と書く．$\#\overline{\mathcal{C}}(\overline{T}_\varphi[n],r)$ が最小となるような φ を φ^* とし，文脈から φ^* に誤解がない限り，その時の $\overline{T}_{\varphi^*}[n]$ を単に $T^*[n]$ と記述する．Algorithm 1 に，$T^*[n]$ を求めるアルゴリズムを示す．

補題 3 任意の自然数 n に対し，φ^* は $[2, \sqrt{n}]$ に存在する．

証明 $T_\varphi[n]$ に含まれる φ の加算，乗算，冪乗の回数をそれぞれ $N_p(T_\varphi[n])$，$N_m(T_\varphi[n])$，$N_e(T_\varphi[n])$ とする．n を φ^* を用いた演算の形に表せるのは $\varphi^* \le n/2$ のときのみであることに注意する．そこで，φ_x を $\sqrt{n}+1 < \varphi_x \le n/2$ なる整数とする．このとき，$T_{\varphi_x}[n]$ は，乗算および冪乗を含まず，加算を 1 回以上含む表現となる．よって，$N_p(T_{\varphi_x}[n]) \ge 1, N_m(T_{\varphi_x}[n]) = N_e(T_{\varphi_x}[n]) = 0$ である．このとき，補題 1 より，次式が成り立つ．

$$\#\overline{\mathcal{C}}(\overline{T}_{\varphi_x}[n],r_x) = 4N_p(T_{\varphi_x}[n]) + 2\varphi_x + 12 + 2(n \bmod \varphi_x).$$

Algorithm 1 $T^*[n]$ を求めるアルゴリズム

Input: n
Output: $T^*[n]$
1: **for** $\varphi = 2$ to $\sqrt{n}$ **do**
2: $r \leftarrow n \bmod \varphi$;
3: $\overline{T}_\varphi[n] \leftarrow$ FACTORIZEBYTETR$((n-r), \varphi)$;
4: **end for**
5: $T^*[n] \leftarrow \#\overline{\mathcal{C}}(\overline{T}_\varphi[n], r)$ が最小となる $\overline{T}_\varphi[n]$.
6:
7: **function** FACTORIZEBYTETR(n, φ)
8: $k \leftarrow {}^k\varphi \le n$ をみたす最大の k;
9: **for** $i = k$ to 1 **do**
10: $p_i \leftarrow (p\ {}^i\varphi \le n$ をみたす最大の $p)$;
11: $p' \leftarrow p_i - (p_i \bmod \varphi)$;
12: **if** $p' > 1$ **then** FACTORIZEBYTETR(p', φ);
13: $n \leftarrow n - p_k\ {}^i\varphi$;
14: **end for**
15: **end function**

次に，$T_{\varphi_x-1}[n]$ の場合を考える．このときも同様に，$N_p(T_{\varphi_x-1}[n]) \ge 1, N_m(T_{\varphi_x-1}[n]) = N_e(T_{\varphi_x-1}[n]) = 0$ であるが，$T_{\varphi_x}[n]$ の場合に比べて，φ_x が $\varphi_x - 1$ に減少し，加算の回数と余りの部分がそれぞれ $\lfloor (N_p(T_{\varphi_x}[n]) + 1 + (n \bmod \varphi))/\varphi_x \rfloor$ および $N_p(T_{\varphi_x}[n]) + 1 + (n \bmod \varphi)$ だけ増加する．したがって，

$$\begin{aligned}
&\#\overline{\mathcal{C}}(\overline{T}_{\varphi_x-1}[n], r_{x-1}) \\
&\quad = 4N_p(T_{\varphi_x-1}[n]) + 2(\varphi_x - 1) \\
&\qquad\qquad + 12 + 2(n \bmod \varphi_x - 1) \\
&\quad = \#\overline{\mathcal{C}}(\overline{T}_{\varphi_x}[n], r_x) - 2 \\
&\qquad + 4\lfloor (N_p(T_{\varphi_x}[n]) + 1 + (n \bmod \varphi))/\varphi_x \rfloor \quad (6) \\
&\qquad + 2(N_p(T_{\varphi_x}[n]) + 1 + (n \bmod \varphi)) \quad (7)
\end{aligned}$$

が成り立つ．ここで，式 (6) と (7) のうち，少なくともどちらか一方は正の値を取る．すなわち，式 (6) + 式 (7) > 2 であることから，

$$\begin{aligned}
&\#\overline{\mathcal{C}}(\overline{T}_{\varphi_x-1}[n], r_{x-1}) - \#\overline{\mathcal{C}}(\overline{T}_{\varphi_x}[n], r_x) \\
&\quad = 4\lfloor (N_p(T_{\varphi_x}[n]) + 1 + (n \bmod \varphi))/\varphi_x \rfloor \\
&\qquad + 2(N_p(T_{\varphi_x}[n]) + 1 + (n \bmod \varphi)) - 2 \\
&\quad > 2 - 2 = 0
\end{aligned}$$

が成り立つ．よって帰納的に，$\#\mathcal{C}_{\varphi_x}(T_{\varphi_x}[n])$ が最小となるのは φ_x が $\varphi_x \le N/2$ で最大のときで，

$$\begin{aligned}
\min(\#\overline{\mathcal{C}}(\overline{T}_{\varphi_x}[n], r_x)) &\le 4 + 2\varphi_x + 12 \\
&= 2\varphi_x + 16 \quad (8)
\end{aligned}$$

とわかる．最小値は $\varphi_x = n/2$ のときに得られる．φ_y を $\varphi_y \le \sqrt{n}$ をみたす最大の整数とする．このとき，$N_m(T_{\varphi_y}[n]) = 1$，$N_p(T_{\varphi_y}[n]) = N_e(T_{\varphi_y}[n]) = 0$，$n \bmod \varphi_y = n - \varphi_y^2$ である．よって補題 1 より，以下が成り立つ．

n	φ^*	$\lfloor\sqrt{n}\rfloor$	n	φ^*	$\lfloor\sqrt{n}\rfloor$
9	3	3	23	2	4
10	3	3	24	2	4
11	3	3	25	5	5
12	3	3	26	5	5
13	3	3	27	3	5
14	3	3	28	3	5
15	3	3	29	3	5
16	2	4	30	3	5
17	2	4	31	3	5
18	2	4	32	2	5
19	2	4	33	2	5
20	2	4	34	2	5
21	2	4	35	2	5
22	2	4	36	3	6

表 1 $9 \le n \le 36$ に対する φ^* と $\lfloor\sqrt{n}\rfloor$

$$\begin{aligned}
&\#\overline{\mathcal{C}}(\overline{T}_{\varphi_y}[n], r_y) \\
&\quad = 2N_m(T_{\varphi_y}[n]) + 2\varphi_y + 12 + 2(n \bmod \varphi_y) \\
&\quad = 2 + 2\varphi_y + 12 + 2(n - \varphi_y^2) \\
&\quad < 2\varphi_y + 14 + 2((\varphi_y + 1)^2 - \varphi_y^2) \\
&\quad = 2\varphi_y + 14 + 2(2\varphi_y + 1) \\
&\quad = 6\varphi_y + 16. \quad (9)
\end{aligned}$$

以上より，次のような不等式が得られる．

$$\begin{aligned}
\text{式 (8)} - \text{式 (9)} &= (2\varphi_x + 16) - (6\varphi_y + 16) \\
&= 2\varphi_x - 6\varphi_y = 2 \cdot n/2 - 6\varphi_y = n - 6\varphi_y \\
&> n - 6\sqrt{n}. \quad (10)
\end{aligned}$$

(i) $n > 36$ **のとき** 式 (10) は正となるから，式 (8) $>$ 式 (9)．すなわち，$\varphi > \sqrt{n}$ で $\varphi \le \sqrt{n}$ のときよりも $\#\overline{\mathcal{C}}(\overline{T}_\varphi[n], r)$ が小さくなることはない．

(ii) $9 \le n \le 36$ **のとき** n に対する φ^* および $\lfloor\sqrt{n}\rfloor$ は表 1 のようになる．このことから，φ^* はいずれも $\sqrt{n}$ 以下であることがわかる．

よって以上 (i)(ii) より，補題 3 が証明された．なお，$n \le 8$ の場合に関しては，補題 2 より，チャーチ数よりもサイズが小さくなる剰余内包チャーチ演算表現は存在しないので，ここでは考慮しない． □

定理 1 Algorithm 1 の時間計算量は $O(\sqrt{n}(\mathrm{slog}_\varphi n)^2)$ である．ここで，slog_φ は超対数，すなわち，超冪 ${}^i\varphi$ に対して $i = \mathrm{slog}_\varphi {}^i\varphi$ となる関係を表す．

証明 2 行目の k の計算にかかる時間は $O(\mathrm{slog}_\varphi n)$ である．このとき，k の最大値は $\mathrm{slog}_\varphi n$ なので，関数 FACTORIZEBYTETR 中の **for** ループ（9-14 行目）は最大で $\mathrm{slog}_\varphi n$ 回繰り返される．各繰り返し中の p_i の計算（5 行目）に関して，$0 \le p < {}^i\varphi$ より，これにかかる時間は $O(\mathrm{slog}_\varphi n)$ である．いま，FACTORIZEBYTETR(n, φ) の計算量を $O(F(n))$，最大の再帰回数を ρ と表すと，次が成り立つ．

$$
\begin{aligned}
&O(F(N))\\
&\quad = O(\mathrm{slog}_\varphi n + \mathrm{slog}_\varphi n(\mathrm{slog}_\varphi n + O(F(\mathrm{slog}_\varphi n))))\\
&\quad = O(\mathrm{slog}_\varphi n(\mathrm{slog}_\varphi n + O(F(\mathrm{slog}_\varphi n))))\\
&\quad = O((\mathrm{slog}_\varphi n)^2 + \mathrm{slog}_\varphi n\, O(F(\mathrm{slog}_\varphi n)))\\
&\quad = O((\mathrm{slog}_\varphi n)^2 + \mathrm{slog}_\varphi n(\mathrm{slog}_\varphi \mathrm{slog}_\varphi n)^2\\
&\qquad + \mathrm{slog}_\varphi n \mathrm{slog}_\varphi \mathrm{slog}_\varphi n(O(F(\mathrm{slog}_\varphi \mathrm{slog}_\varphi n))))\\
&\quad = O((\mathrm{slog}_\varphi n)^2 + \mathrm{slog}_\varphi n(\mathrm{slog}_\varphi \mathrm{slog}_\varphi n)^2\\
&\qquad + \mathrm{slog}_\varphi n \mathrm{slog}_\varphi \mathrm{slog}_\varphi n(\mathrm{slog}_\varphi \mathrm{slog}_\varphi \mathrm{slog}_\varphi n)^2\\
&\qquad + \cdots +\\
&\qquad + \mathrm{slog}_\varphi n \cdots \overbrace{\mathrm{slog}_\varphi \cdots \mathrm{slog}_\varphi}^{\rho-1} n(\overbrace{\mathrm{slog}_\varphi \cdots \mathrm{slog}_\varphi}^{\rho} n)^2\\
&\qquad + \mathrm{slog}_\varphi n \cdots \overbrace{\mathrm{slog}_\varphi \cdots \mathrm{slog}_\varphi}^{\rho} n).
\end{aligned}
$$

ここで，$\overbrace{\mathrm{slog}_\varphi \cdots \mathrm{slog}_\varphi}^{i-1} n < (\overbrace{\mathrm{slog}_\varphi \cdots \mathrm{slog}_\varphi}^{i} n)^2$ より，

$$O(F(n)) = O((\mathrm{slog}_\varphi n)^2)$$

である．さらに補題 3 より，φ の変化は 2 から $\sqrt{n}$ までであるので，1-4 行目の **for** ループは最大で $\sqrt{n}-1$ 回繰り返される．以上より，Algorithm 1 の時間計算量は $O(\sqrt{n}(\mathrm{slog}_\varphi n)^2)$ となる． □

$T^*[n]$ を求めるにあたり，φ^* は最大で $\sqrt{n}$ まで探索する必要がある．例えば，$n = 49$ に対する $T^*[49]$ は $7 \cdot 7$ であり，$\varphi^* = 7 = \sqrt{49}$ である．

補題 4　補題 2 の命題に対して裏が成り立つ．すなわち，$n > 8$ のとき，$\#\Lambda(n) > \#\overline{\mathcal{C}}(F_\varphi[\bar{n}], r)$ となるような n の分解式 $F_\varphi[\bar{n}] + r$ が存在する．

証明　$n > 8$ で $\#\Lambda(n) > \#\overline{\mathcal{C}}(T^*[n], r)$ となる $T^*[n]$ が存在することを示す．式 (9) より，

$$
\begin{aligned}
\#\overline{\mathcal{C}}(\overline{T}_{\varphi_y}[n], r_y) &< 6\varphi_y + 16\\
&\leq 6\sqrt{n} + 16. \qquad (11)
\end{aligned}
$$

(i) $n > 19$ **のとき**　$\#\Lambda(n) = 2n + 3 > 6\sqrt{n} + 16$ より式 (11) とあわせて，$\#\Lambda(n) > \#\overline{\mathcal{C}}(\overline{T}_{\varphi_y}[n], r_y) \geq \#\overline{\mathcal{C}}(T^*[n], r)$ が言える．

(ii) $8 < n \leq 19$ **のとき**　表 2 より，$\#\Lambda(n) = 2n + 3 > \#\overline{\mathcal{C}}(T^*[n], r)$ とわかる．

以上 (i)(ii) より，補題 4 は成り立つ． □

補題 4 より，いま $\overline{\mathcal{C}}(T^*[n], r)$ において $\varphi^* > 8$ であったとき，$\overline{\mathcal{C}}(T^*[n], r)$ の引数部のチャーチ数 $\Lambda(\varphi^*)$ を，さらに超冪分解した分解式の剰余内包チャーチ演算表現に書き換えることで，よりサイズの小さいラムダ式を得ることができる．この操作を $\varphi^* \leq 8$ となる $\overline{\mathcal{C}}(T^*[n], r)$ が現れるまで再帰的に行った剰余内包チャーチ演算表現を $\mathcal{C}^*[n]$ と表す．$\mathcal{C}^*[n]$ を求めるアルゴリズムを Algorithm 2 に示す．

n	$\#\Lambda(n) = 2n+3$	$\#\overline{\mathcal{C}}(T^*[n], r)$
9	21	20
10	23	22
11	25	24
12	27	24
13	29	26
14	31	28
15	33	28
16	35	20
17	37	22
18	39	24
19	41	26

表 2　$9 \leq n \leq 19$ に対する $\Lambda(n) = 2n + 3$ と $\#\overline{\mathcal{C}}(T^*[n], r)$ の値

Algorithm 2 $\mathcal{C}^*[n]$ を求めるアルゴリズム

Input: n
Output: $\mathcal{C}^*[n]$

```
1: function compressChurch(n)
2:     if n ≤ 8 then
3:         return Λ(n)
4:     else
5:         for φ = 2 to √n do
6:             r_φ ← n mod φ;
7:             T_φ[n] ← factorizeByTetr((n − r_φ), φ);
8:         end for
9:         φ* ← #C̄(T̄_φ[n], r_φ) が最小となるときの φ.
10:        T*[n] ← T̄_φ*[n]
11:        r ← r_φ*
12:        F ← C̄(T*[n], r_φ*) の関数部のラムダ式;
13:        G ← compressChurch(φ*);
14:        return (F G)
15:    end if
16: end function
17: C*[n] ← compressChurch(n);
```

定理 2　$\#\mathcal{C}^*[n]$ の最大値は $O(\log n)$ である．

証明　式 (9) より，$\#\overline{\mathcal{C}}(\overline{T}_{\varphi_y}[n], r_y) < 6\varphi^*_y + 16$ であるから，$\varphi^*_{y1} > 8$ に対して $\#\mathcal{C}^*[n] < 6\#\overline{\mathcal{C}}(\overline{T}_{\varphi^*_{y1}}[\varphi^*_y], r_{y1}) + 16$ が成り立つ．さらに，$\varphi^*_{y2} > 8$ に対して $\#\overline{\mathcal{C}}(\overline{T}_{\varphi^*_{y1}}[\varphi_y], r_{y1}) < 6\#\overline{\mathcal{C}}(\overline{T}_{\varphi^*_{y2}}[\varphi^*_{y1}], r_{y2}) + 16$ である．これは $\varphi^*_{yi} > 8$ に対して再帰的に成り立ち，再帰回数を ρ とすると，

$$
\begin{aligned}
\#\mathcal{C}^*(T^*[n]) &< 6\#\overline{\mathcal{C}}(\overline{T}_{\varphi^*_{y1}}[\varphi^*_1], r_{y1}) + 16\\
&< 6(6\#\overline{\mathcal{C}}(\overline{T}_{\varphi^*_{y2}}[\varphi^*_2], r_{y2}) + 16) + 16\\
&\quad \cdots\\
&< 6(6(\cdots(6\varphi^*_{y\rho} + 16)\cdots) + 16) + 16\\
&= 6^\rho \varphi^*_{y\rho} + \sum_{i=0}^{\rho-1}(6^i \cdot 16) \qquad (12)
\end{aligned}
$$

となる．ここで，$\varphi^*_{y\rho}$ は 8 以下の定数である．さらに，いま各 $\varphi^*_{y(i+1)}$ はそれぞれ $\sqrt{\varphi^*_{yi}}$ 以下に減少していくので，再帰回数 ρ は高々 $O(\log\log n)$ である．したがって，

$$O(\#\mathcal{C}^*[n]) = O(6^{\log\log n}) = O(\log n)$$

であるから，定理 2 が成り立つ． □

3.4 超冪分解表現を用いた連続パターンの圧縮

ここでは，チャーチ数の超冪分解表現の関数部に引数部のラムダ式を直接に記述することで，ラムダ式のサイズを小さくする手法について述べる．

例として，文字列 b, d について，b が 16 回連続したのちに d が続く文字列 $b^{16}d$ を考える．また，文字列 b, d をラムダ式に変換したものをそれぞれ B, D とする．このとき，文字列 $b^{16}d$ は，提案手法によって

$$\mathcal{C}^*[16]\ B\ D \tag{13}$$

のように変換される．ここで，

$$\mathcal{C}^*[16] = (\lambda pfx.p\ p\ p\ f\ x)\ (\lambda fx.f\ (f\ x)). \tag{14}$$

である．式 (14) は，計算すると，

$$\begin{aligned}\mathcal{C}^*[16] &\to \lambda fx.\overbrace{f\ (\cdots(f}^{16}\ x)\cdots) \\ &= \Lambda(16)\end{aligned}$$

となる．

算出されたチャーチ数 $\Lambda(16)$ の束縛変数 f および x は，元の $\mathcal{C}^*[16]$ の関数部における束縛変数 f, x に由来している．そこで，$\mathcal{C}^*[16]\ B\ D$ の引数部にある B と D を先に f と x に代入すると，

$$(\lambda p.p\ p\ p\ B\ D)\ (\lambda fx.f\ (f\ x)). \tag{15}$$

という式が得られる．ラムダ式 (13) と (15) のサイズを比較すると，

$$\begin{aligned}&\text{式 (13) の右辺のサイズ：}\quad 22 + \#B + \#D, \\ &\text{式 (15) の右辺のサイズ：}\quad 16 + \#B + \#D\end{aligned}$$

となり，式 (15) のほうが小さい．

以降では，繰り返しパターン $b^n d$ のチャーチ演算表現を用いたラムダ式において，このように引数部にある文字列の式 B, D を関数部の束縛変数の代わりに直接置くことを，文字列を**内包させる**という．$\mathcal{C}^*[n]$ に文字列の式を内包させたものを**内包表現**と呼び，$\mathcal{C}^*[n][,]$ と表記する．$[,]$ 内には束縛変数 f, x の代わりに内包させる文字列 b, d のラムダ式をそれぞれ書く．この表記を用いると，式 (15) は，$\mathcal{C}^*[n][B, D]$ と書ける．文字列は b, d 両方を内包させなくても，片方だけ内包させることができる．単に $\mathcal{C}^*[n][,]$ と書いたときは，どちらも内包させていない，すなわち $\mathcal{C}^*[n]$ と等しいラムダ式を表すことにする．

定理 3 $\mathcal{C}^*[n]\ B\ D$ とその内包表現に関して以下が成り立つ．

$$\#\mathcal{C}^*[n][, D] < \#\mathcal{C}^*[n][,],$$

$$\begin{cases}\#\mathcal{C}^*[n][B,] < \#\mathcal{C}^*[n][,] \\ \qquad ((N_p(T_\varphi[n]) - 1)(\#B - 1) < 4 \text{ の場合}), \\ \#\mathcal{C}^*[n][B,] \geq \#\mathcal{C}^*[n][,] \qquad (\text{それ以外の場合}).\end{cases}$$

証明 まず $\mathcal{C}^*[n]$ の関数部の束縛変数 x に対応する D の内包を考える．この内包では D の関数適用と $\mathcal{C}^*[n]$ の関数部において変数 x を束縛する λ 記号がなくなり，代わりに $\mathcal{C}^*[n]$ の関数部の束縛変数 x が D に置き換わる．したがってこれによる全体のラムダ式のサイズの増減は，

$$-(\#D + 1 + 1) + (-1 + \#D) = -3$$

となるから，内包表現は $\#D$ によらず常に -3 だけ変化する．

次に $\mathcal{C}^*[n]$ の関数部の束縛変数 f に対応する B の内包を考える．$\mathcal{C}^*[n]$ の関数部における束縛変数 f の出現回数が $N_p(T_\varphi[n]) + 1$ であることに注意して D の場合と同様に考えると，全体のラムダ式のサイズの増減は，

$$\begin{aligned}&-(\#B + 1 + 1) + (-(N_p(T_\varphi[n]) + 1) + N_p(T_\varphi[n])\#B) \\ &\quad = N_p(T_\varphi[n])\#B - N_p(T_\varphi[n]) - \#B - 3 \\ &\quad = (N_p(T_\varphi[n]) - 1)(\#B - 1) - 4\end{aligned}$$

となる．ゆえに，$(N_p(T_\varphi[n]) - 1)(\#B - 1) < 4$ のときは，内包表現の方がサイズが小さくなる． □

定理 3 より，$\#B = 1$，すなわち b がひとつの文字であるときは，B を内包した内包表現の方が常にサイズが小さくなる．提案手法では，内包表現に変換した場合とそうでない場合に対して，サイズが小さくなる方のラムダ式を圧縮データとして取り出している．すなわち，$b^n d$ を表すラムダ式を

$$\begin{cases}\mathcal{C}^*[n][B, D] \\ \quad ((N_p(T_\varphi[n]) - 1)(\#B - 1) < 4 \text{ の場合}), \\ \mathcal{C}^*[n][, D]\ B \qquad (\text{それ以外の場合})\end{cases}$$

としている．

4. 実験

本論文での提案手法の性能を評価するため，得られるラムダ式のサイズを，Kobayashi ら [4] のバイナリ表現を用いる方法（2.4 節）と比較した．以降では，Kobayashi ら [4] の手法をバイナリ表現法と呼ぶ．入力データには，連続パターンを持つ文字列として $a^n c$ を用いた．ここで a と c はそれぞれ大きさ 1 の終端文字で，n は 1 から 10000 までの自然数である．

実験の結果を図 1 に示している．図中において，バイナ

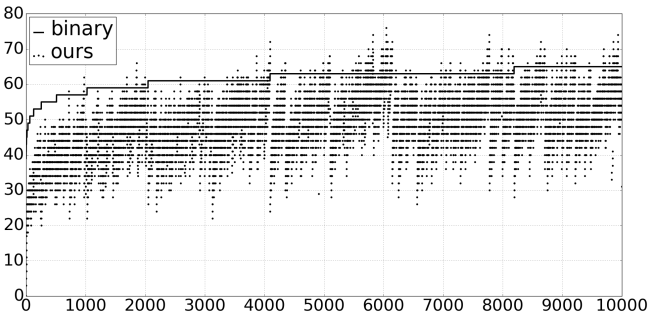

図 1

リ表現法を binary，提案手法を ours で示している．横軸は n を，縦軸はラムダ式のサイズを表している．バイナリ表現法において，文字列 $a^n c$ に対し圧縮データとして得られるラムダ式のサイズは $2\lceil \log n \rceil + 39$ である．定理 2 より，提案手法による圧縮で得られるラムダ式の最大サイズは $O(\log n)$ であるが，場合によってはバイナリ表現法よりサイズが大きくなることがある．実際，binary のサイズが ours のサイズよりも小さくなる n の個数は，1〜10000 のうち 375 個存在した．しかしながら，全体的に見ると，(ours のサイズ)/(binary のサイズ) の平均値は約 0.7887 であり，提案手法のほうが平均して約 21% ほど圧縮率が良いことがわかる．

5. おわりに

本論文では，高階圧縮において連続パターンに対しコンパクトな圧縮表現を生成する方法を提案した．また，既存の高階圧縮の手法と圧縮率を比較し，提案手法が実際に有効であることを確認した．

本論文で提案した連続パターンに対する圧縮は，既存の高階圧縮の方法にサブルーチンとして組み込むことで圧縮率の向上が期待できる．既存の高階圧縮の方法としては Kobayashi ら [4] による手法や，その高速化を図った矢口ら [6] の手法がある．提案手法をこれらのアルゴリズムに組み込む場合，長いラムダ式の中から同じパターンが連続している部分を効率よく発見する必要があるが，その処理方法には改善の余地が残っている．また，今回は連続なパターンについてのみ考えたが，高階圧縮では従来手法では扱えなかった複雑なパターンを抽出してまとめあげることができる．そうしたパターンを高速に発見する手法の開発も今後の課題である．

参考文献

[1] Barendregt, H.: *The Lambda Calculus, Its Syntax and Semantics: Revised edition*, North-Holland (1985).
[2] Huffman, D.: A Method for the Construction of Minimum-Redundancy Codes, *Proc. the Institute of Radio Engineers*, Vol. 40, No. 9, pp. 1098–1101 (1952).
[3] Kieffer, J. C. and Yang, E.-H.: Grammar-Based Codes: a New Class of Universal Lossless Source Codes, *IEEE Trans. on Inform. Theory*, Vol. 46, No. 3, pp. 737–754 (2000).
[4] Kobayashi, N., Matsuda, K., Shinohara, A. and Yaguchi, K.: Functional Programs As Compressed Data, *Higher Order Symbol. Comput.*, Vol. 25, No. 1, pp. 39–84 (2012).
[5] Sayood, K.(ed.): *Lossless Compression Handbook*, Academic Press (2002).
[6] 矢口和也，小林直樹，篠原　歩：高階圧縮の高速化と効率の良い符号化，電子情報通信学会技術研究報告. COMP, コンピュテーション， Vol. 113, No. 252, pp. 13–20 (2013).