Vol.2019-AL-173 No.7
2019/5/10

パラメタ化文字列に対する線形サイズ接尾辞トライ

中島 克仁[1]　ヘンリアン ディプタラマ[1]　吉仲 亮[1]　篠原 歩[1]

概要：パラメタ化パターン照合問題とは，入力テキスト T と入力パターン P が与えられたとき，P を T の部分文字列と一致させるような文字集合上の置換が存在するかどうかを判定する問題である．本論文では，効率的にこの問題を解くための新たな索引構造としてパラメタ化線形サイズ接尾辞トライ (PLST) を提案する．PLST は，既存のパラメタ化接尾辞木と同様に，T に対し線形サイズであるが，T そのものを保持する必要がない．また，PLST を用いて P に対し線形時間で照合を行うアルゴリズムを提案する．

キーワード：文字列アルゴリズム，パラメタ化パターン照合，索引構造，接尾辞木

1. はじめに

パターン照合は，パターン文字列をテキスト文字列から見つける処理であり，データ処理における最も基本的な要素技術である．静的なテキストに対して効率的にパターン照合を行うために，数多くの索引構造が提案されている．

テキスト T の接尾辞トライは，T のすべての接尾辞を表現するトライ木である．各辺には文字がラベル付けされており，根からパターン P を 1 文字づつ読み込むことで P が T に出現するか否かを判定できる．T の接尾辞トライの大きさは $O(|T|^2)$ である．接尾辞木 [15] は，接尾辞トライにおいて分岐のない頂点を削除し，分岐頂点を辺で直接結んだ圧縮版であり，大きさは $O(|T|)$ である．接尾辞トライの元のパスラベルを復元できるように，テキスト T を保存するとともに，接尾辞木の各辺は T の対応する部分文字列への参照を持っている．接尾辞木は，広く使われている索引構造であり，パターン照合を含む様々な用途がある [5], [11].

近年，Crochemore ら [6] は，接尾辞トライの別の圧縮版である**線形サイズ接尾辞トライ** *(Linear-size Suffix Trie, LST)* と呼ばれる新たな索引構造を提案した．LST は，接尾辞木と同様に非分岐頂点のみで構成されるパスを辺で置き換えるが，接尾辞木とは異なり，LST 自体の他の辺ラベルを参照することによって元のパスラベルを復元する．このことにより，もとのテキストを保存する必要がないため，同じ文字列を索引付けする上で，LST は接尾辞木より省メモリとなる可能性がある．LST は接尾辞木の代替としてみることができ，パターン照合問題に限らず様々な用途に用いられることが期待されている．

一方で，パターン照合の様々な変種が提案され盛んに研究されている．変数記号を含む文字列をパラメタ化文字列といい，この変数記号は別の変数記号に置き換えることができる．パラメタ化パターン照合とは，パラメタ化文字列 T と P が与えられたとき，P の変数記号を一対一に置き換えた文字列を T から見つける処理である．Baker [2] によって導入されたこのパラメタ化パターン照合は，ソフトウェアメンテナンス [1], [2], [3]，剽窃の検出 [9]，遺伝子構造の分析 [14] などの応用先がある．通常のパターン照合と同様に，効率的にパラメタ化パターン照合を行うための索引構造が提案されている．パラメタ化接尾辞木 [2]，構造化接尾辞木 [14]，パラメタ化接尾辞配列 [7], [12]，パラメタ化ポジションヒープ [8], [10] などである．

本論文では，パラメタ化文字列に対する新たな索引構造として**パラメタ化線形サイズ接尾辞トライ** *(Parameterized Linear-size Suffix Trie, PLST)* を提案する．PLST は p-文字列の直前符号化された接尾辞に対する接尾辞木，つまりパラメタ化接尾辞木 [2] の変種である．また，PLST を用いることで，与えられたパターン文字列に対して，$O(m)$ 時間でパラメタ化パターン照合問題を解くアルゴリズムを提案する．ここで，m はパターンの長さである．

2. 準備

2.1 基本的な定義と表記

非負整数の集合を $\mathcal{N}$ と表す．Δ をアルファベットとする．文字列 $w = xyz \in \Delta^*$ に対し，x, y, z をそれぞれ w の**接頭辞**, **部分文字列**, **接尾辞**と呼ぶ．w の長さを $|w|$ で表し，$1 \leq i \leq |w|$ に対して，w の i 番目の文字を $w[i]$ と表

[1] 東北大学大学院情報科学研究科
Graduate School of Information Sciences, Tohoku University

す．$1 \le i \le j \le |w|$ に対して，$w[i:j]$ は位置 i から位置 j までの w の部分文字列を表す．便宜上，$1 \le i \le |w|$ に対して，$w[1:i]$ を $w[:i]$，$w[:i]$ を $w[i:]$ と表す．空文字列を ε と表す．すなわち $|\varepsilon| = 0$ である．$i > j$ に対しては，$w[i:j] = \varepsilon$ と定義する．文字列 u とその延長 uv に対し，$\mathsf{str}(u, uv) = v$ と定義する．

本論文では，2 つのアルファベット Σ と Π を固定する．Σ の要素を**定数**文字，Π の要素を**変数**文字と呼ぶ．Σ^* の要素を**定数文字列**，$(\Sigma \cup \Pi)^*$ の要素を**パラメタ化文字列**，または略して **p-文字列**と呼ぶ．また，Σ と Π の大きさを定数であると仮定する．

長さ n の 2 つの p-文字列 w_1 と w_2 が与えられたとき，すべての $a \in \Sigma$ に対し $f(a) = a$ であり，すべての $1 \le i \le n$ に対して，$f(w_1[i]) = w_2[i]$ であるような $\Sigma \cup \Pi$ 上の全単射 f が存在するならば，w_1 と w_2 は**パラメタ化一致**または **p-マッチ**であり，$w_1 \approx w_2$ と表す．次のように定義された**直前符号化**という符号化を用いることで，$w_1 \approx w_2$ かどうかを判断することができる．

定義 1 (直前符号化 [2]) $\Sigma \cup \Pi$ 上の長さ n の p-文字列 w に対し，$\mathsf{prev}(w)$ と表される w の**直前符号化**は，以下を満たすような $\Sigma \cup \mathcal{N}$ 上の長さ n の文字列として定義される．各 $1 \le i \le n$ に対して，

$$\mathsf{prev}(w)[i] = \begin{cases} w[i] & \text{if } w[i] \in \Sigma, \\ 0 & \text{if } w[i] \in \Pi \text{ and} \\ & w[i] \neq w[j] \text{ for } 1 \le j < i, \\ i - k & \text{otherwise.} \end{cases}$$

ここで，$k = \max\{j \mid w[j] = w[i] \text{ and } 1 \le j < i\}$ とする．$\Sigma \cup \mathcal{N}$ 上の文字列を **pv-文字列**と呼ぶ．

すべての p-文字列 w_1 および w_2 に対して，$\mathsf{prev}(w_1) = \mathsf{prev}(w_2)$ の場合に限り，$w_1 \approx w_2$ となる．例えば，$\Sigma = \{\mathtt{a}, \mathtt{b}\}$ および $\Pi = \{\mathtt{u}, \mathtt{v}, \mathtt{x}, \mathtt{y}\}$ が与えられたとき，$s_1 = \mathtt{uvvvauuvb}$ と $s_2 = \mathtt{xyyyaxxyb}$ は $f(\mathtt{u}) = \mathtt{x}$ かつ $f(\mathtt{v}) = \mathtt{y}$ となる f によりパラメタ化一致である．このとき，$\mathsf{prev}(s_1) = \mathsf{prev}(s_2) = 0011\mathtt{a}514\mathtt{b}$ となる．

パラメタ化パターン照合を次のように定義する．

定義 2 (パラメタ化パターン照合) 2 つの p-文字列であるテキスト T とパターン P が与えられたとき，T が P とパラメタ化一致する部分文字列を持つか判定する．

例えば，$\Sigma = \{\mathtt{a}, \mathtt{b}\}$ および $\Pi = \{\mathtt{u}, \mathtt{v}, \mathtt{x}, \mathtt{y}\}$ 上のテキスト $T = \mathtt{auvaubuavbv}$ とパターン $P = \mathtt{xayby}$ を考えたとき，T は P とパラメタ化一致する部分文字列 $T[3:7] = \mathtt{vaubu}$ と $T[7:11] = \mathtt{uavbv}$ を持つ．

本論文では，テキスト T が T の終端文字以外に出現しない記号 $\$ \in \Sigma$ で終わっていると仮定する．

2.2 接尾辞トライ，接尾辞木，線形サイズ接尾辞トライ

接尾辞トライ $\mathsf{STrie}(T)$ は，$T \in \Sigma^*$ のすべての部分文字列に対応する頂点を持つ木である．図 1 (a) に接尾辞トライの例を示す．本論文を通して，便宜上，頂点をそれに対応する文字列で識別する．ただし，各頂点は対応する文字列を明示的には記憶しない．$a \in \Sigma$ としたとき，T の空でない部分文字列 ua ごとに，$a \in \Sigma$ でラベル付けされた u から ua までの辺が存在する．また，根から頂点 u へのパス上のラベルを読むことで，頂点が対応する文字列 u が取得できる．$u, v \in \Sigma^*$ において，頂点 u から子孫 uv へのパスラベルは $\mathsf{str}(u, uv) = v$ である．T の部分文字列は $O(|T|^2)$ 個あるため，$\mathsf{STrie}(T)$ の大きさは $O(|T|^2)$ である．

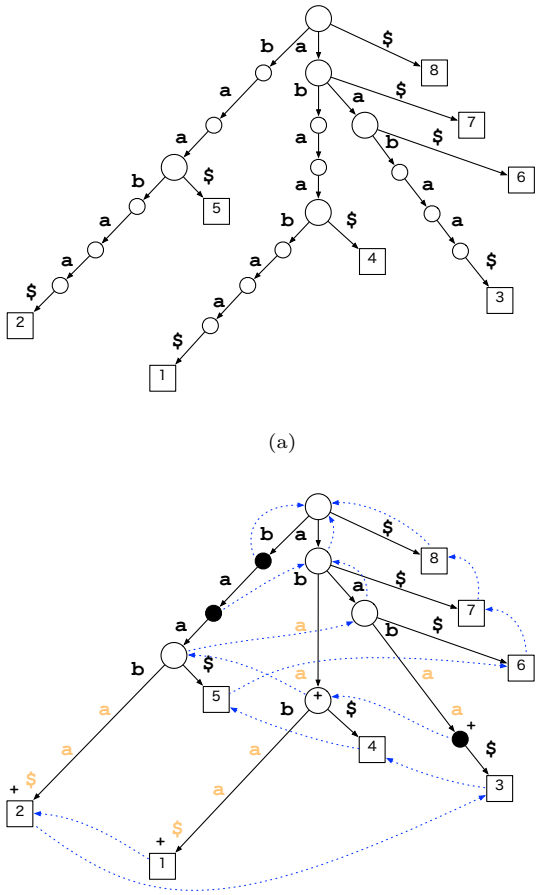
接尾辞木 $\mathsf{STree}(T)$ は，$\mathsf{STrie}(T)$ から，分岐のない頂点をすべて削除し，分岐がない頂点での各パスをそのパスラベルが対応するテキスト T の区間を参照するような単一の辺で置き換えることによって得られる木構造である．つまり，辺 (u, v) のラベルは，$T[i:j] = \mathsf{str}(u, v)$ となるような 2 つ組 (i, j) である．$|T| - 1$ 個の分岐頂点があるため，$\mathsf{STree}(T)$ の大きさは $O(|T|)$ である．

頂点上の重要な補助関数を**接尾辞リンク**といい，SL で表す．これは $a \in \Sigma$ と $w \in \Sigma^*$ としたとき，各頂点 aw に対して $\mathsf{SL}(aw) = w$ と定義される．

文字列 T の**線形サイズ接尾辞トライ (LST)** [6] $\mathsf{LST}(T)$ は接尾辞トライの別のコンパクトな変形である（図 1 (b) 参照）．LST は接尾辞トライの（ほとんどの）分岐していない頂点を削除し，接尾辞木のようにパスを辺で置き換えるが，これらの新しい辺のラベルは入力テキストの区間を参照しない．各辺 (u, v) は元のパスラベル $\mathsf{str}(u, v)$ の最初の文字 $\mathsf{str}(u, v)[1]$ だけを保持する．元のラベル $\mathsf{str}(u, v)$ を復元するには，$\mathsf{str}(u, v) = \mathsf{str}(\mathsf{SL}(u), \mathsf{SL}(v))$ という事実にもとづき，接尾辞リンクを使用して LST 自体の別の辺またはパスを参照する．この参照は再帰的になるが，最終的に，保持している文字を集めることによって元のパスラベルを復元することができる．この復元のために，$\mathsf{LST}(T)$ は，$\mathsf{STrie}(T)$ の分岐しない内部頂点をいくつか保持する．よって $\mathsf{LST}(T)$ は，$\mathsf{STree}(T)$ よりも多くの頂点を持つことになるが，$|T|$ に対して線形の大きさを保つ．$\mathsf{STrie}(T)$ の頂点を以下のように Type 1，Type 2，そしてそれら以外の 3 つに分類する．そのうち $\mathsf{STree}(T)$ の頂点である Type 1 頂点と追加の Type 2 頂点が $\mathsf{LST}(T)$ を構成する頂点である．

(1) Type 1 頂点は葉頂点か分岐頂点である．
(2) Type 2 頂点は接尾辞リンクが Type 1 頂点を指すような非分岐頂点である．

各辺 (u, v) は，$|v| - |u| = 1$ かどうかを示す 1 ビットのフラグがある．もし $|v| - |u| = 1$ であれば，元の完全なラベル $\mathsf{str}(u, v)$ は LST の辺 (u, v) のラベルから分かる．そうでなければ，ラベルの先頭文字以外の文字を復元するために接尾辞リンクを辿る必要がある．LST では接尾辞リンクを



(a)

(b)

図 1 (a) 文字列 T = abaabaa$ に対する接尾辞トライ STrie(T). (b) T に対する線形サイズ接尾辞トライ LST(T). 実線と破線の矢印は，それぞれ辺と接尾辞リンクを表す．LST は各辺にラベルの先頭文字（黒）のみを保持し，残りの文字（オレンジ）は保持しない．大きな白丸と小さな黒丸は，それぞれ Type 1 と Type 2 の頂点を表す．頂点 v に + 記号がある場合，辺 (u, v) は STrie(T) の長さが 1 より大きいパスラベルを持つ．

用いて，接尾辞トライでの元のパスラベルを復元する．もし LST に Type 1 頂点しか存在しなければ，ある辺 (u, v) に対して，SL(u) と SL(v) の間に分岐頂点が存在することがあり，元のラベルを一意に復元することが困難になる．Type 2 頂点を持つことにより，すべての辺 (u, v) に対し，SL(u) と SL(v) の間に分岐頂点は存在しない．よって，元のパスラベルを復元するのには，SL(u) から下に辿るだけで良い．

2.3 パラメタ化接尾辞トライ，パラメタ化接尾辞木

p-文字列 $T \in (\Sigma \cup \Pi)^*$ に対し，T の **直前符号化部分文字列 (pv-部分文字列)** を T の部分文字列 w の直前符号化 prev(w) とする．T の pv-部分文字列の集合を PrevSub(T) と表す．

T のパラメタ化接尾辞トライ PSTrie(T) は T のすべての pv-部分文字列を表すトライ構造である．PSTrie(T) の

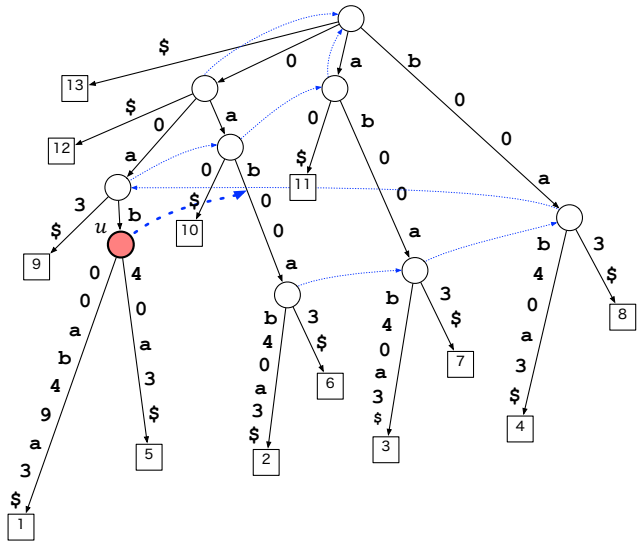


図 2 Σ = {a, b, $} および Π = {x, y, z, w} 上の T = xyabzwabzxaz$ に対するパラメタ化接尾辞木 PSTree(T). 青い破線の矢印は接尾辞リンクを表す．一部の接尾辞リンクは分岐頂点を指さない（太い破線の矢印で示されている接尾辞リンクが一例）．

大きさは $O(|T|^2)$ である．

pv-文字列 $u \in (\Sigma \cup \mathcal{N})^*$ に対し，$\langle u \rangle$ と表される u の **再符号化** は，以下を満たすような pv-文字列として定義される．各 $1 \le i \le |u|$ に対して，

$$\langle u \rangle[i] = \begin{cases} 0 & \text{if } u[i] \in \mathcal{N} \text{ and } u[i] \ge i, \\ u[i] & \text{otherwise.} \end{cases}$$

任意の p-文字列 $w \in (\Sigma \cup \Pi)^*$ と $i, j \le |w|$ に対して，$\langle \mathsf{prev}(w)[i:j] \rangle = \mathsf{prev}(w[i:j])$ となる．

通常，接尾辞リンクは接尾辞木の頂点に定義されるが，STrie(T) の根以外のすべての頂点，すなわち T のすべての空でない部分文字列に「仮想接尾辞リンク」を持つと便利である．空でない pv-文字列 $u \in (\Sigma \cup \mathcal{N})^+$ に対し，sl(u) を u の先頭文字を削除した文字列の再符号化 $\langle u[2:] \rangle$ とする．パラメタ化接尾辞トライ（およびこれに基づく索引構造）の頂点 u の接尾辞リンクを sl(u) と定義する．定数文字列とは異なり，$u \in$ PrevSub(T) のとき，必ずしも $u[2:] \in$ PrevSub(T) ではなく，sl(u) = $\langle u[2:] \rangle \in$ PrevSub(T) である．

PSTree(T) で表される T の **パラメタ化接尾辞木（p-接尾辞木）** [2] は，パラメタ化接尾辞トライの圧縮版である．図 2 は p-接尾辞木の例を示している．Σ 上の定数文字列の接尾辞木のように，PSTree(T) は，分岐のない内部頂点を削除し，各辺のラベルを元のテキスト T への参照とすることによって得られる．Lee ら [13] は，頂点 u と sl(u) を結ぶ接尾辞リンクを使用することによって，PSTree(T) を確率的 $O(|T|)$ 時間でオンラインで構築できることを示した．

接尾辞木では，分岐頂点の接尾辞リンクは必ず分岐頂点を指す．つまり，u が頂点の場合，sl(u) も同様に頂点であ

る．しかし，パラメタ化接尾辞木では，「接尾辞リンク」が分岐頂点を指していない分岐頂点がある．図 2 に例を示す．ここで，頂点 00ab（赤い丸）は $\mathsf{PSTree}(T)$ の分岐頂点だが $\mathsf{sl}(\mathtt{00ab}) = \mathtt{0ab}$ は分岐頂点ではない．

接尾辞リンクは p-接尾辞木の構築にとって重要であるが，p-接尾辞木を使ったパターンマッチングには使われない．しかし，パラメタ化線形サイズ接尾辞トライでは，元のラベルを復元するために接尾辞リンクを再帰的にたどる必要があるため，接尾辞リンクは重要である．この点については，次の節で詳しく説明する．

3. パラメタ化線形サイズ接尾辞トライ

この節では，p-文字列のための新たな索引構造である **パラメタ化線形サイズ接尾辞トライ (PLST)** を提案する．この索引構造は，節 2.2 で紹介した線形サイズ接尾辞トライに基づいている．PLST の一例を図 3 に示す．p-文字列を扱うよう LST を拡張するためには問題がある．具体的には，辺 (u, v) に対して，$\mathsf{str}(u, v)$ を復元する上で次の 2 つが問題となる．

(1) 必ずしも $\mathsf{str}(u, v) = \mathsf{str}(\mathsf{sl}(u), \mathsf{sl}(v))$ ではない．
(2) $\mathsf{sl}(u)$ が分岐頂点でないような分岐頂点 u が $\mathsf{PSTrie}(T)$ に存在する．

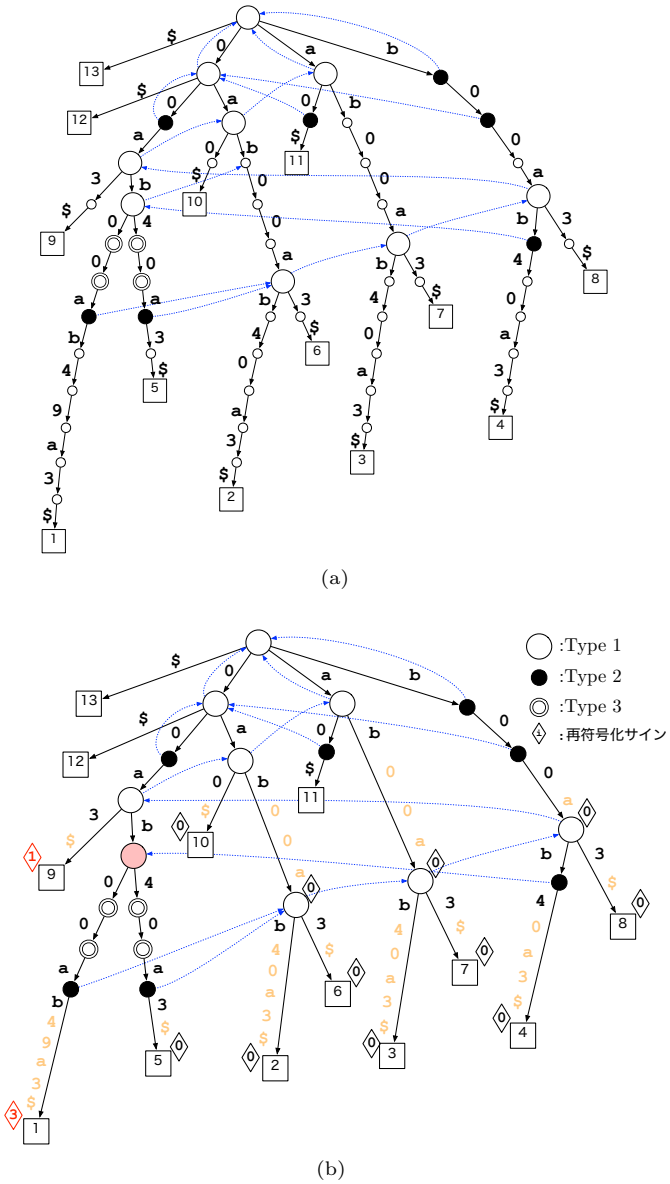
1 つ目は，$\mathsf{sl}(u) = u[2:]$ ではなく，$\mathsf{sl}(u) = \langle u[2:] \rangle$ であることにより生じる．このことにより，接尾辞リンクを用いて参照したパスラベル $\mathsf{str}(\mathsf{sl}(u), \mathsf{sl}(v))$ が期待していた文字列ではない場合がある．また，2 つ目により，$\mathsf{sl}(u)$ が頂点として存在しない場合，接尾辞リンクを用いて接尾辞トライでの元のパスラベルを復元する手法自体が使えなくなる．

3.1 パラメタ化線形サイズ接尾辞トライの定義と性質

$U = \mathsf{PrevSub}(T)$ を $\mathsf{PSTrie}(T)$ の頂点集合と呼ぶ．T に対する PLST $\mathsf{PLST}(T)$ の頂点集合 V は U の部分集合であり，$V = V_1 \cup V_2 \cup V_3 \subseteq U$ として分割する．$i = 1, 2, 3$ に対し，V_i に属する頂点を **Type** i と呼ぶ．Type 1 と Type 2 の定義はそれぞれ元の LST [6] の定義に従う．

(1) $\mathsf{PSTrie}(T)$ の頂点 u が葉頂点もしくは分岐頂点ならば，$u \in U$ は Type 1 である．
(2) $u \notin V_1$ かつ $\mathsf{sl}(u) \in V_1$ ならば，$u \in U$ は Type 2 である．

パラメタ化文字列では，$\mathsf{sl}(u) \notin V_1 \cup V_2$ となるような頂点 $u \in V_1$ が存在する場合があり，u の子 v に対して，$\mathsf{str}(u, v)$ を復元できないため，これらの頂点だけでは不十分である．ここで，$\mathsf{sl}(u) \notin V_1 \cup V_2$ ならば，$u \in V_1$ を **悪い頂点** と呼ぶ．この問題を克服するための 1 つのアイデアとして，V が sl に閉じるように，$i = 1, \ldots, |u|$ と $u \in V_1$ に対し，$\mathsf{sl}^i(u)$ を V に追加することが考えられる．ここで，$\mathsf{sl}^i(u) = \mathsf{sl}(\mathsf{sl}^{i-1}(u))$ および $\mathsf{sl}^0(u) = u$ である．しかし，これらの追加頂点の数が，$\Omega(|T|^2)$ となる例がある．そこで，



(a)

(b)

図 3 $\Sigma = \{\mathtt{a}, \mathtt{b}, \$\}$ および $\Pi = \{\mathtt{x}, \mathtt{y}, \mathtt{z}, \mathtt{w}\}$ 上の文字列 $T = \mathtt{xyabzwabzxaz\$}$ ($\mathsf{prev}(T) = \mathtt{00ab00ab49a3\$}$) に対する (a) $\mathsf{PSTrie}(T)$ と (b) $\mathsf{PLST}(T)$．白丸，黒丸，二重丸はそれぞれ Type 1, Type 2 そして Type 3 頂点を表す．PLST は各辺の先頭の文字（黒）のみをラベルとして保持し，残りの文字（オレンジ）は保持しない．

u が悪い頂点であるとき，u から v へのパス上に $\mathsf{str}(u, v)$ を明示的に与える．すなわち，以下の Type 3 頂点を追加する．

(3) $u \notin V_1 \cup V_2$ かつ u の親頂点が悪い Type 1 もしくは Type 3 ならば，$u \in U$ は Type 3 である．

節 3.3 で $|V| \in O(|T|)$ であることを示す．$\mathsf{sl}(u) \in V_1$ ならば，$u \in V_1 \cup V_2$ を **良い頂点** と呼ぶ．そうでなければ，$u \in V_3$ の場合も含めて，悪い頂点と呼ぶ．根 ε は Type 1 の悪い頂点である．

$\mathsf{PLST}(T)$ の辺は自明に決まる．$v \neq \varepsilon$ かつ $uv' \in V$ であるような空でない v の接頭辞 v' がないとき，$(u, uv) \in V \times V$

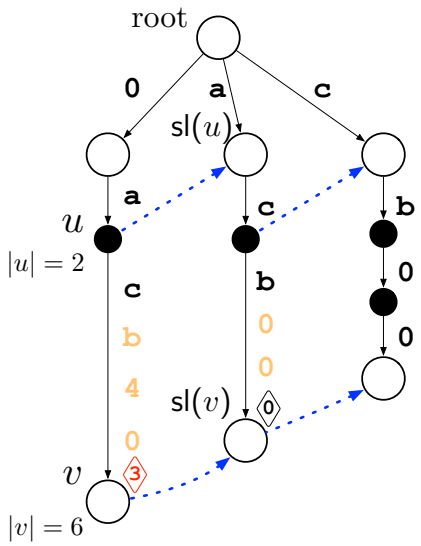


図 4 再符号化記号の説明図．菱形の数字は，各頂点の再符号化記号を表す．例えば，頂点 $v = \mathsf{0acb40}$ に対しては，$v[5] = 4$ かつ v の親頂点は長さ 2 の $u = \mathsf{0a}$ であるため，$\mathsf{Re}(v) = 5-2 = 3$ である．頂点 $\mathsf{sl}(v) = \mathsf{acb40}$ に対しては，$\mathsf{Re}(\mathsf{sl}(v)) = 0$ である．

を辺とする．辺 (u, uv) のラベルは $v[1]$ であり，uv は u の $v[1]$-**子**といい，$\mathsf{child}(u, v[1])$ で表される．

良い頂点に対しては，接尾辞リンクを定義をするが，悪い頂点に対しては，接尾辞リンクを未定義のままにする．

定義より，次の性質が容易に導かれる．

観察 1 $\mathsf{PLST}(T)$ のすべての辺 (u, v) に対し，u が悪い頂点ならば，$|\mathsf{str}(u, v)| = 1$ である．

もし u が良いならば，$\mathsf{PLST}(T)$ の辺 (u, v) に対する，$\mathsf{str}(u, v)$ を接尾辞リンクを用いて復元する．重要な観察として，式 $\mathsf{str}(u, v) = \mathsf{str}(\mathsf{sl}(u), \mathsf{sl}(v))$ が（パラメタ化ではない）LST において元のラベルを復元するための重要な性質であったが，PLST では必ずしもこの方程式が成り立たないことが挙げられる．図 4 に例を示す．ここで，$\mathsf{str}(u, v) = \mathsf{cb40} \neq \mathsf{cb00} = \mathsf{str}(\mathsf{sl}(u), \mathsf{sl}(v))$ であり，$\mathsf{str}(u, v)$ の 3 文字目 4 は $\mathsf{str}(\mathsf{sl}(u), \mathsf{sl}(v))$ において 0 に再符号化されている．もし，このような値の変化が多くの位置で起きるならば，接尾辞リンクを用いたラベルの復元は難しくなるが，$\mathsf{str}(u, v)$ と $\mathsf{str}(\mathsf{sl}(u), \mathsf{sl}(v))$ での変化は限られたものである．

観察 2 テキスト T のすべての直前符号化された部分文字列 v には $v[i] = i - 1$ であるよな位置 i が高々 1 つ存在する．そして，そのような位置 i に対し，$\mathsf{sl}(v)[i-1] = 0$ となる．それゆえに，$\mathsf{PLST}(T)$ の辺 (u, v) の $\mathsf{str}(u, v)$ において，そのような位置は一意に定まる．

各辺 (u, v) に対してのみ，$\mathsf{str}(\mathsf{sl}(u), \mathsf{sl}(v))$ から $\mathsf{str}(u, v)$ を復元するために，**再符号化記号**を以下のように定義する．

定義 3 (再符号化記号) 各頂点 $v \in V$ に対し，u を v の親頂点とする．**再符号化記号**を以下のように定義する．

$$\mathsf{Re}(v) = \begin{cases} i - |u| & \text{if there exists } i \text{ such that} \\ & v[i] = i - 1 \text{ and } |u| < i \leq |v|, \\ 0 & \text{otherwise.} \end{cases}$$

観察 2 より，再符号化記号 $\mathsf{Re}(v)$ は一意に定義される．図 4 に再符号化記号の例を示す．観察 2 と定義 3 より，直ちに次の補題 1 が示される．

補題 1 (u, v) を u が根でないよな $\mathsf{PLST}(T)$ の辺とする．すべての $i \in \{1, \ldots, |\mathsf{str}(u, v)|\} \setminus \{\mathsf{Re}(v)\}$ に対して，$\mathsf{str}(u, v)[i] = \mathsf{str}(\mathsf{sl}(u), \mathsf{sl}(v))[i]$ である．また，$\mathsf{Re}(v) \geq 1$ ならば，$\mathsf{str}(u, v)[\mathsf{Re}(v)] = |u| + \mathsf{Re}(v) - 1$ であり $\mathsf{str}(\mathsf{sl}(u), \mathsf{sl}(v))[\mathsf{Re}(v)] = 0$ である．

補題 1 より，v の再符号化記号を用いることで，$\mathsf{str}(\mathsf{sl}(u), \mathsf{sl}(v))$ から，$\mathsf{str}(u, v)$ を復元することができる．

まとめると，$\mathsf{PLST}(T)$ は良い Type 1, 悪い Type 1, Type 2 (すべて良い) そして Type 3 (すべて悪い) の 4 種類の頂点から構成されている．$u \in V$ が良い頂点ならば，u は深さ，接尾辞リンク，再符号化記号の 3 つ，つまり $(|u|, \mathsf{SL}(u), \mathsf{Re}(u))$ の 3 つ組を持つ．ここで，$\mathsf{SL}(u) = \mathsf{sl}(u)$ である．ここでは，接尾辞リンク $\mathsf{SL}(u)$ が文字列そのものではなく文字列 $\mathsf{sl}(u)$ に対応する頂点へのポインタであることを強調するため，$\mathsf{SL}(u)$ の表記を用いる．それゆえに，必要なメモリ量は定数である．$u \in V$ が悪い頂点ならば，u は接尾辞リンクを持たない．つまり，$(|u|, \mathsf{null}, \mathsf{Re}(u))$ の 3 つ組を持つ．そして，各辺 (u, v) はラベル $\mathsf{str}(u, v)[1]$ を持つ．

3.2 パラメタ化線形サイズ接尾辞トライでのパラメタ化照合

この小節では，PLST の応用としてパラメタ化パターン照合問題を解くためのアルゴリズムを提案する．Algorithm 1 の関数 P-MATCH は pv-文字列 p と $\mathsf{PLST}(T)$ の頂点を入力とし，$p = \mathsf{str}(u, v)$ となるような $v \in \mathsf{PrevSub}(T)$ が存在するか判定する．存在しなければ null を返す．存在する場合，$v' \in V$ となるような v の最も短い延長である v' を返す．言い換えれば，p は $\mathsf{str}(u, v')$ の接頭辞である．また，もし $v \in V$ ならば，v' は v となる．p-文字列であるパターン P が T の k 個の位置 $j_1, \ldots, j_k$ から開始する部分文字列とパラメタ化一致するならば，$v = \text{P-MATCH}(\mathsf{prev}(P), \varepsilon)$ は子孫の葉が $\mathsf{prev}(T[j_1 :]), \ldots, \mathsf{prev}(T[j_k :])$ であるような頂点である．

入力の組 (p, u) に対し，$p = \varepsilon$ のとき，P-MATCH は u を返す．$p \neq \varepsilon$ のとき，u が $p[1]$ の子を持つならば，u の $p[1]$ の子 v に対する $\mathsf{str}(u, v)$ の復元を始める．はじめに，$|p| \geq |v| - |u|$ の場合を考える．$l = |v| - |u|$ として，$p[1 : l] = \mathsf{str}(u, v)$ かどうかを確かめる．$l = 1$ ならば，$p[1 : l] = \mathsf{str}(u, v)$ であることは確認済みである．そのときは，v へ行き，$(p[2 :], v)$ を入力として再帰的に関数を呼び

出す．$l \geq 2$ ならば，先頭文字 $\mathsf{str}(u,v)[1] = p[1]$ を除き，$\mathsf{str}(u,v)$ がどのような文字列であるかを辺 (u,v) から知ることはできない．そこで，$\mathsf{str}(u,v)$ を復元するために，u の接尾辞リンクを用いる．$|v| - |u| \geq 2$ のときは観察 1 より u は良い頂点であり，それ故に $\mathsf{SL}(u)$ も定義されている．$\mathsf{Re}(v) = 0$ ならば，補題 1 より $\mathsf{str}(u,v) = \mathsf{str}(\mathsf{sl}(u), \mathsf{sl}(v))$ である．このときは，単に P-Match$(p[1:l], \mathsf{SL}(u))$ を呼ぶ．そうでなければ，補題 1 より，$p[\mathsf{Re}(v)] = |u| + \mathsf{Re}(v) - 1$ かつ $p'[1:l] = \mathsf{str}(\mathsf{sl}(u), \mathsf{sl}(v))$ の場合に限り，$p[1:l] = \mathsf{str}(u,v)$ である．ここで $i = 1, \ldots, |p|$ に対し，

$$p'[i] = \begin{cases} 0 & \text{if } i = \mathsf{Re}(v), \\ p[i] & \text{otherwise} \end{cases}$$

である．したがって，P-Match$(p'[1:l], \mathsf{SL}(u))$ の再帰呼び出しは，$p[1:l] \neq \mathsf{str}(u,v)$ の場合に限り null を返す．$\mathsf{sl}(v) \notin V$ の場合もあるかもしれないが，このアルゴリズムには影響しない．この再帰呼び出しは，$p'[1:l] = \mathsf{str}(\mathsf{sl}(u), \mathsf{sl}(v))$ かどうかを判定するが，$\mathsf{sl}(v)$ は使用しないからである．P-Match$(p'[1:l], \mathsf{SL}(u))$ が頂点を返した場合，$p[1:l] = \mathsf{str}(u,v)$ であり，P-Match$(p[l+1:], v)$ を呼び出すことで照合を続ける．

上記の議論は $|p| \leq |v| - |u|$ のときも有効である．$\mathsf{Re}(v) = 0$ または $\mathsf{Re}(v) > |p|$ ならば，p が $\mathsf{str}(\mathsf{sl}(u), \mathsf{sl}(v))$ の接頭辞である場合に限り，p は $\mathsf{str}(u,v)$ の接頭辞である．そうでなければ，p' が $\mathsf{str}(\mathsf{sl}(u), \mathsf{sl}(v))$ の接頭辞であり，$p[\mathsf{Re}(v)] = |u| + \mathsf{Re}(v) - 1$ のとき，p は $\mathsf{str}(u,v)$ の接頭辞である．したがって，再帰は正当である．P-Match$(p'[1:l], \mathsf{SL}(u))$ が頂点を返した場合，p は $\mathsf{str}(u,v)$ の接頭辞であり，P-Match(ε, v) は v を返す．

補題 2 Algorithm 1 を用いることで，P とパラメタ化一致する T の部分文字列が存在するかどうか判定することができる．

ここからは，Algorithm 1 の計算量について議論する．P-Match(p,u) が呼び出されたとする．$|v| - |u| \geq |p| \geq 2$ であり $\mathsf{Re}(v) = 0$ または $\mathsf{Re}(v) > l$ である場合がある．ここで，$v = \mathsf{child}(u, p[1])$ である．この場合，アルゴリズムは単に P-Match$(p, \mathsf{SL}(u))$ を呼ぶが，このときの引数は前の呼び出しのときと変わらない．このような再帰が何度も繰り返される場合がある．図 5 は，そのような辺 (u,v) の一例を示す．ここで，$u = 00\mathsf{ab}00\mathsf{a}$ である．同様の問題およびそれに対する解決策が（パラメタ化ではない）LST について Crochemore ら [6] によって既に議論されている．その解決策に従って，以下のように**高速リンク**を導入する．このリンクにより，元の引数と同じ引数を用いる再帰呼び出しを飛ばすことができる．

定義 4 (高速リンク) $|v| - |u| > 1$ であるような各辺 $(u,v) \in V \times V$ に対して，(u,v) の**高速リンク**は $\mathsf{FL}(u,v) = \mathsf{SL}^k(u)$ と定義される．ここで，$k \geq 1$ は次のどちらかを満たす最小の整数である．

(1) $|v_k| < |v| - k$, または，

(2) $0 < \mathsf{Re}(v_k) < |v| - k$.

ここで，$a = \mathsf{str}(u,v)[1]$ に対し，$v_k = \mathsf{child}(\mathsf{SL}^k(u), a)$ である．

Algorithm 1 は，15 行目の $\mathsf{SL}(u)$ を $\mathsf{FL}(u,v)$ に置き換えることで，線形時間で動作する．$|v_k| < |v| - k$ のときは，$|v_k| - |\mathsf{SL}^k(u)| < |p|$ である．$0 < \mathsf{Re}(v_k) < |v| - k$ のときは，p の $\mathsf{Re}(v_k)$ 番目の文字を 0 に変更する．したがって，高速リンクを辿る回数は，$2|p|$ に抑えられる．図 5 の例で

Algorithm 1: Parameterized pattern matching algorithm (P-Match(p,u))

Input: A string p and a node u in $\mathsf{PLST}(T)$
Output: The highest descendant v of u such that p is a prefix of $\mathsf{str}(u,v)$

```
1  if p = ε then
2  |  return u;
3  else
4  |  if child(u, p[1]) is undefined then
5  |  |  return null;
6  |  else
7  |  |  v ← child(u, p[1]);
8  |  |  l ← min{|p|, |v| − |u|};
9  |  |  if l ≥ 2 then
10 |  |  |  if 1 ≤ Re(v) ≤ |p| then
11 |  |  |  |  if p[Re(v)] = |u| + Re(v) − 1 then
12 |  |  |  |  |  p[Re(v)] ← 0;
13 |  |  |  |  else
14 |  |  |  |  |  return null;
15 |  |  |  if P-Match(p[1 : l], SL(u)) = null then
16 |  |  |  |  return null;
17 |  |  return P-Match(p[l + 1 :], v);
```

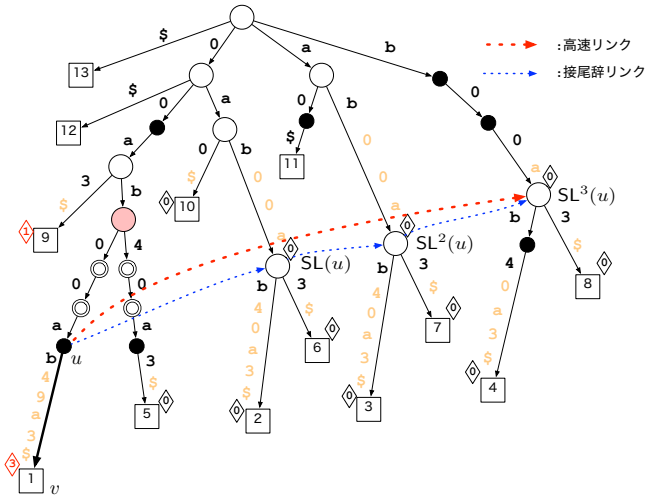


図 5 赤い破線の矢印で描かれた高速リンクは，青い破線の矢印で描かれた接尾辞リンクがたどった中間頂点をスキップすることによって得られる．高速リンク $\mathsf{FL}(u,v)$ を用いて u から $\mathsf{SL}^3(u)$ へ飛ぶ．

は，u から直接 $\mathsf{SL}^3(u)$ へ飛ぶ．

定理 1 パラメタ化線形サイズ接尾辞トライ $\mathsf{PLST}(T)$ と長さ m のパターン P が与えられたとき，P とパラメタ化一致する T の部分文字列が存在するかどうか $O(m)$ 時間で判定することができる．

3.3 パラメタ化線形サイズ接尾辞トライの大きさ

この小節では，$\mathsf{PLST}(T)$ の大きさが長さ n のテキスト T に対して線形であることを示す．まず，$\mathsf{PLST}(T)$ の頂点数が線形であることを示す．Type 1 頂点はパラメタ化接尾辞木に現れる頂点であるため，高々 $2n$ 個 [2] である．Type 2 と Type 3 の頂点も同様に線形であることを示すことができる．

補題 3 $\mathsf{PLST}(T)$ の Type 2 頂点の数は高々 $2n$ 個である．

証明 $1 \le k < n$ において，$w = \mathsf{prev}(T[:k])$ から始まる $\mathsf{PSTrie}(T)$ の仮想の接尾辞リンクの鎖を考える．つまり，$(w, \mathsf{sl}(w), \mathsf{sl}^2(w), \ldots, \mathsf{sl}^{|w|}(w))$ とする．$\mathsf{PSTrie}(T)$ には $n-1$ 個の接尾辞リンクの鎖が存在し，$\mathsf{PLST}(T)$ のすべての内部頂点は少なくとも 1 つの鎖に含まれている．もし鎖に 2 つの異なる Type 2 頂点 $\mathsf{sl}^i(w)$ と $\mathsf{sl}^j(w)$ があり，$i < j$ ならば，定義より $\mathsf{sl}^{i+1}(w)$ は Type 1 頂点であるため，鎖の 2 つの異なる Type 2 頂点の間には必ず Type 1 頂点が存在する．
V_1 と V_2 との二項関係 R を次のように定義する．

$$R = \{\, (u,v) \in V_1 \times V_2 \mid \text{there is } i \text{ such that } v = \mathsf{sl}^i(u) \text{ and } \mathsf{sl}^j(u) \notin V_2 \text{ for all } j < i \,\}$$

そして，$R_2 = \{\, v \in V_2 \mid (u,v) \in R \text{ for some } u \in V_1 \,\}$ とする．R は分岐頂点から Type 2 頂点への部分関数なので，$|R_2| \le n$ となる．鎖に関する上記の議論より，各鎖には $v \notin R_2$ であるような Type 2 頂点 $v \in V_2$ が高々 1 つ存在する．$n-1$ の鎖が存在するので，$|V_2 \setminus R_2| < n$ である．全体で，$|V_2| = |R_2| + |V_2 \setminus R_2| < n + n = 2n$ となる．□

Type 3 頂点の上界も同様に示す．

補題 4 (Baker [4]) 悪い頂点 $u \in V_1 \setminus \{\varepsilon\}$ は必ず 2 つの子 $\mathsf{child}(u,0)$ と $\mathsf{child}(u,|u|)$ を持つ．

ある悪い頂点 $u \in V_1 \setminus \{\varepsilon\}$ と Type 3 頂点 v の間の頂点がすべて Type 3 ならば，u は v を**支配する**という．すべての Type 3 頂点は固有の悪い頂点によって支配されている．

補題 5 ある悪い頂点 $u \in V_1 \setminus \{\varepsilon\}$ が支配する Type 3 頂点の数は高々 $2k_{f(u)}$ である．ここで，

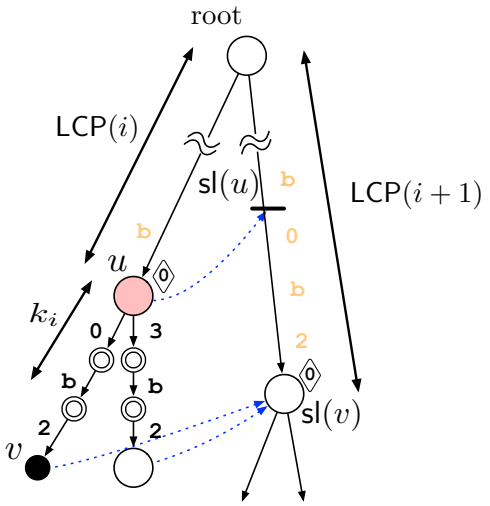


図 6 悪い Type 1 頂点 u は 2 つの枝においてそれぞれ 2 つの Type 3 頂点を支配している．ここでは，$u = \mathsf{prev}(T[i : i + \mathsf{LCP}(i) - 1])$, $v = \mathsf{prev}(T[i : i + \mathsf{LCP}(i+1)])$ そして $k_i = \mathsf{LCP}(i+1) - \mathsf{LCP}(i) = 2$ である．

$$f(u) = i \text{ such that } u = \mathsf{prev}(T[i : i + \mathsf{LCP}(i) - 1]),$$

$$k_i = \max\{0, \mathsf{LCP}(i+1) - \mathsf{LCP}(i)\},$$

$$\mathsf{LCP}(i) = \begin{cases} 0 & \text{for } i = 0, \\ \max\{\mathsf{lcp}(\mathsf{prev}(T[i:]), \mathsf{prev}(T[j:])) \mid 0 \le j < i\} & \text{for } i \ge 1, \end{cases}$$

$$\mathsf{lcp}(x,y) = \max\{\, |v| \mid v \text{ is a prefix of both } x \text{ and } y \,\}$$

である．

証明 図 6 は以下の証明理解の補助となる．補題 4 より，悪い頂点 u はちょうど 2 つの子を持つ．それぞれの子の子孫について，u が支配する Type 3 頂点の数は最大でも $k_{f(u)}$ であることを示せば十分である．$i = f(u)$ とする．$\mathsf{sl}(u) = \mathsf{prev}(T[i+1 : i + \mathsf{LCP}(i) - 1])$ は枝分かれしていないことから，$\mathsf{LCP}(i+1) \ge \mathsf{LCP}(i)$ である．定義より，次を満たすような $j < i$ が存在する．

$$\mathsf{prev}(T[i+1 : i + \mathsf{LCP}(i+1)]) = \mathsf{prev}(T[j+1 : j + \mathsf{LCP}(i+1)]), \quad \text{and}$$

$$\mathsf{prev}(T[i+1 : i + \mathsf{LCP}(i+1) + 1]) \neq \mathsf{prev}(T[j+1 : j + \mathsf{LCP}(i+1) + 1]).$$

よって，$\mathsf{prev}(T[i+1 : i + \mathsf{LCP}(i+1)]) \in V_1$ である．それゆえに，$\mathsf{prev}(T[i : i + \mathsf{LCP}(i+1)]) \in V_1 \cup V_2$ であり，これは $u = \mathsf{prev}(T[i : i + |u| - 1])$ の子の子孫である．したがって，u が支配する頂点の数は，それぞれの子の子孫について，高々 $\mathsf{LCP}(i+1) - \mathsf{LCP}(i)$ である． □

補題 6 $\mathsf{PLST}(T)$ の Type 3 頂点の数は高々 $2n$ 個である．

証明 補題 5 での関数 f は単射である．したがって，$k_i = \max\{0, \mathsf{LCP}(i+1) - \mathsf{LCP}(i)\}$ に対して $\sum_{i=1}^{n} k_i < n$

を示せば十分である．$\mathsf{prev}(T[i:i+l]) = \mathsf{prev}(T[j:j+l])$ は $\mathsf{prev}(T[i+1:i+l]) = \mathsf{prev}(T[j+1:j+l])$ を意味するため，すべての i に対して $\mathsf{LCP}(i) \leq \mathsf{LCP}(i+1)+1$ である．$l = 1, \ldots, n$ について $\sum_{i=1}^{l} k_i < l + \mathsf{LCP}(l)$ であることを l に関する帰納法で容易に示すことができる．特に，$l = n$ に対しては，$\sum_{i=1}^{n} k_i < n + \mathsf{LCP}(n) = n$ となる．□

辺とそのラベルの数，そして頂点に対しての深さ，接尾辞リンク，再符号化記号の数は，$\mathsf{PLST}(T)$ の頂点数によって漸近的に抑えることができる．したがって，$\mathsf{PLST}(T)$ の大きさは $O(n)$ であるといえる．

定理 2 $\Sigma \cup \Pi$ 上の長さ n の p-文字列 T が与えられたとき，$\mathsf{PLST}(T)$ の大きさは $O(n)$ である．

4. まとめと今後の課題

本論文ではパラメタ化パターン照合問題のための新たな索引構造であるパラメタ化線形サイズ接尾辞トライを提案した．n を T の長さとしたとき，T に対するパラメタ化線形サイズ接尾辞トライの大きさは $O(n)$ である．また，長さ m の入力パターン P に関して，$O(m)$ 時間でパラメタ化パターン照合問題を解くアルゴリズムを示した．

Lee らのパラメタ化接尾辞木 [13] は，入力テキストを保持し，各辺はテキストの直前符号化部分文字列を復元するためにテキストの位置の 3 つの位置を持つ．PLST はテキストを持たず，各ノードはその深さ，接尾辞リンク，および再符号化記号の 3 つ組を持つ．したがって，パラメタ化線形サイズ接尾辞トライはパラメタ化接尾辞木よりも省メモリとなる可能性がある．最長共通部分文字列の計算など，PLST を様々な用途に役立てるためには PLST を効率的に構築する必要がある．パラメタ化接尾辞木は線形時間の構築アルゴリズムがいくつか知られており [13], [14], [16]，同様に PLST でも効率的な構築アルゴリズムを開発することが今後の課題である．

参考文献

[1] Brenda S. Baker. A program for identifying duplicated code. *Computing Science and Statistics*, 24:49–57, 1992.
[2] Brenda S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *Proc. 25th annual ACM symposium on Theory of computing*, pages 71–80, 1993.
[3] Brenda S. Baker. Parameterized pattern matching: Algorithms and applications. *Journal of Computer and System Sciences*, 52(1):28–42, 1996.
[4] Brenda S. Baker. Parameterized duplication in strings: Algorithms and an application to software maintenance. *SIAM Journal on Computing*, 26(5):1343–1362, 1997.
[5] M. Crochemore and W. Rytter. *Jewels of Stringology: Text Algorithms*. World Scientific, 2003.
[6] Maxime Crochemore, Chiara Epifanio, Roberto Grossi, and Filippo Mignosi. Linear-size suffix tries. *Theoretical Computer Science*, 638:171–178, 2016.
[7] Satoshi Deguchi, Fumihito Higashijima, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda. Parameterized suffix arrays for binary strings. In *Proceedings of the Prague Stringology Conference 2008*, pages 84–94, Czech Technical University in Prague, Czech Republic, 2008.
[8] Diptarama, Takashi Katsura, Yuhei Otomo, Kazuyuki Narisawa, and Ayumi Shinohara. Position heaps for parameterized strings. In *28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017)*, pages 8:1–8:13, 2017.
[9] Kimmo Fredriksson and Maxim Mozgovoy. Efficient parameterized string matching. *Information Processing Letters*, 100(3):91–96, 2006.
[10] Noriki Fujisato, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Right-to-left online construction of parameterized position heaps. In *Prague Stringology Conference 2018*, pages 91–102, 2018.
[11] Dan Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.
[12] Tomohiro I, Satoshi Deguchi, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda. Lightweight parameterized suffix array construction. In *Combinatorial Algorithms (IWOCA 2009)*, pages 312–323, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
[13] Taehyung Lee, Joong Chae Na, and Kunsoo Park. Online construction of parameterized suffix trees for large alphabets. *Information Processing Letters*, 111(5):201–207, 2011.
[14] Tetsuo Shibuya. Generalization of a suffix tree for RNA structural pattern matching. *Algorithmica*, 39(1):1–19, 2004.
[15] Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory (SWAT 1973)*, pages 1–11. IEEE, 1973.
[16] 藤里 法輝, 中島 祐人, 稲永 俊介, 坂内 英夫, 竹田 正幸. 右から左に構築するパラメタ化接尾辞木. 情報処理学会研究報告, Vol. 2019–AL–171, No. 2, pp. 1–7, 2019.