Finding Characteristic Substrings from Compressed Texts

Shunsuke Inenaga

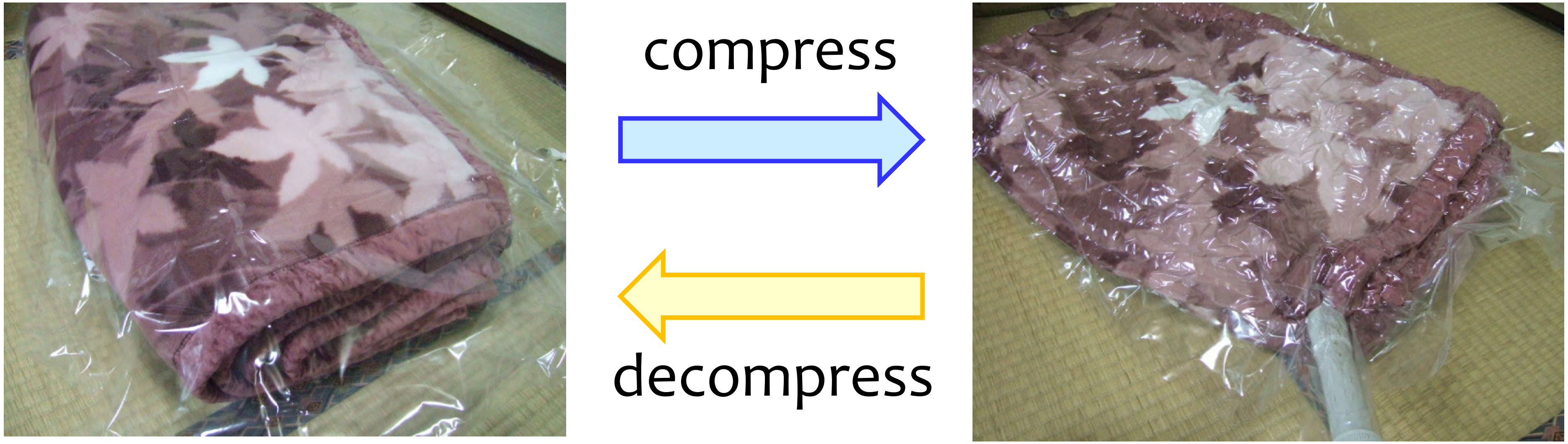
Kyushu University, Japan

Hideo Bannai

Kyushu University, Japan

Text Mining and Text Compression

- *Text mining* is a task of finding some rule and/or knowledge from given textual data.
- *Text compression* is to reduce a space to store given textual data by removing redundancy.



Our Contribution

- We present efficient algorithms to find *characteristic* substrings (patterns) from given compressed strings *directly* (i.e., *without decompression*).
  - Longest repeating substring (LRS)
  - Longest non-overlapping repeating substring (LNRS)
  - Most frequent substring (MFS)
  - Most frequent non-overlapping substring (MFNS)
  - Left and right contexts of given pattern

Text Compression by Straight Line Program

SLP $\mathcal{T}$

$X_1 = \mathrm{a}$
$X_2 = \mathrm{b}$
$X_3 = X_1 X_2$
$X_4 = X_3 X_1$
$X_5 = X_3 X_4$
$X_6 = X_5 X_5$
$X_7 = X_4 X_6$
$X_8 = X_7 X_5$

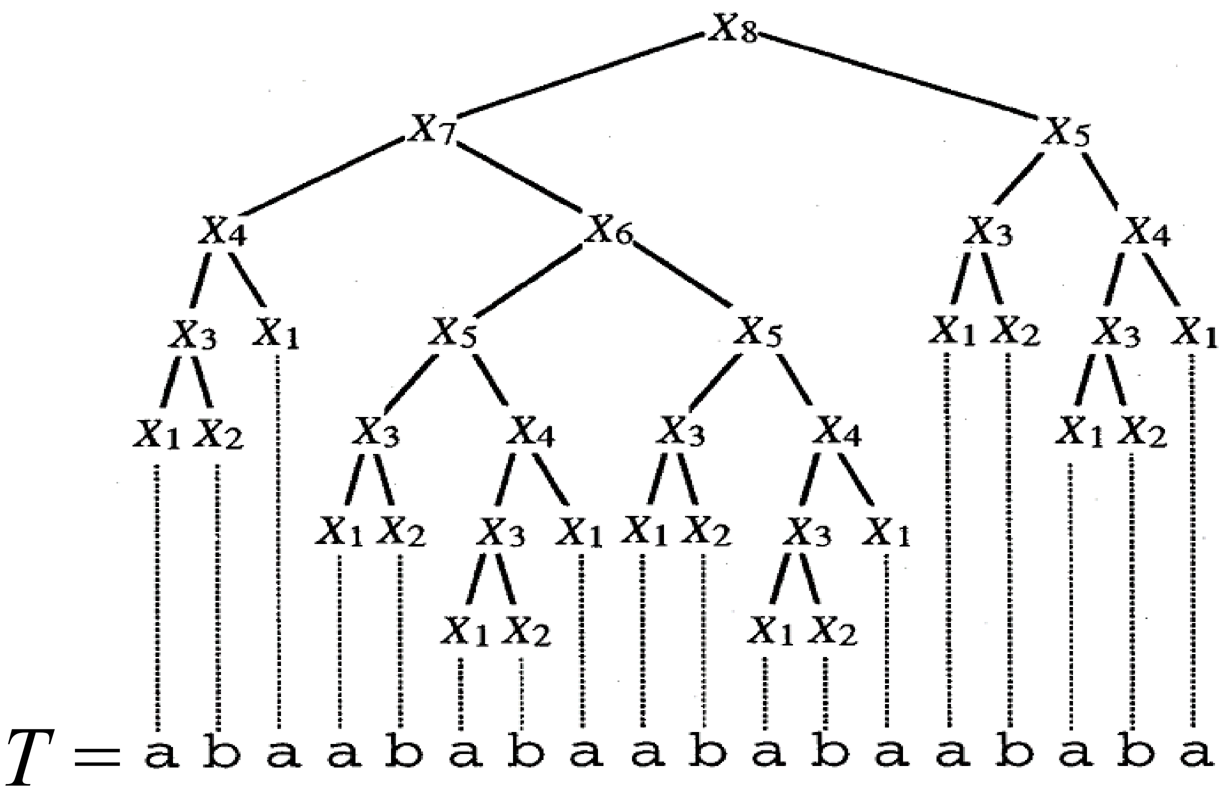
$T = \mathrm{a\ b\ a\ a\ b\ a\ b\ a\ a\ b\ a\ b\ a\ a\ b\ a\ b\ a}$

SLP $\mathcal{T}$ is a CFG in the Chomsky normal form which generates language $\{T\}$.

Text Compression by Straight Line Program

SLP $\mathcal{T}$

$X_1 = \mathrm{a}$
$X_2 = \mathrm{b}$
$X_3 = X_1X_2$
$X_4 = X_3X_1$
$X_5 = X_3X_4$
$X_6 = X_5X_5$
$X_7 = X_4X_6$
$X_8 = X_7X_5$



Encodings of the LZ-family, run-length, Sequitur, etc. can quickly be transformed into SLP.

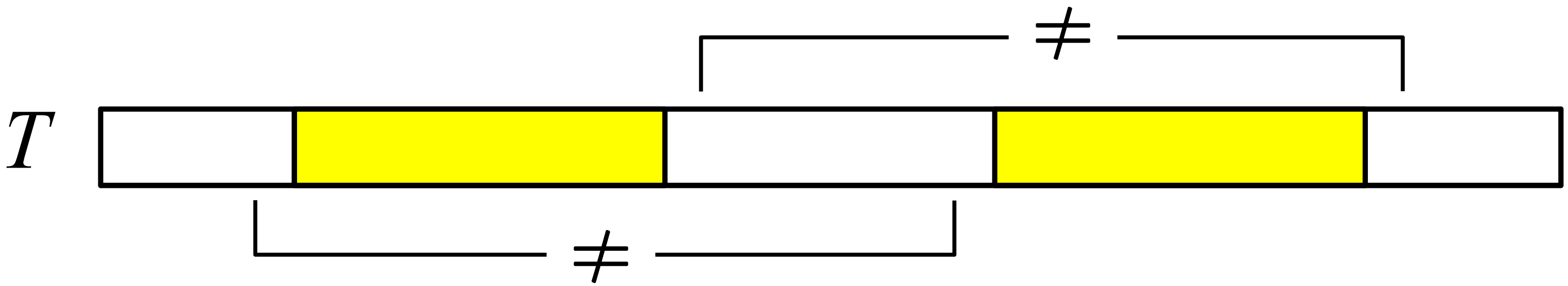
Exponential Compression by SLP

- Highly repetitive texts can be *exponentially large* w.r.t. the corresponding SLP-compressed texts.
- Text T = ababab···ab (T is an N repetition of ab)
- SLP $\mathcal{T}$: $X_1 = \text{a},\ X_2 = \text{b},\ X_3 = X_1X_2,\ X_4 = X_3X_3,$
  $X_5 = X_4X_4,\ \ldots\ ,\ X_n = X_{n\text{-}1}X_{n\text{-}1}$
- $N = O(2^n)$
- *Any* algorithms that decompress given SLP-compressed texts can take exponential time!

We present efficient (i.e., *polynomial-time*) algorithms *without decompression*.

Finding Longest Repeating Substring

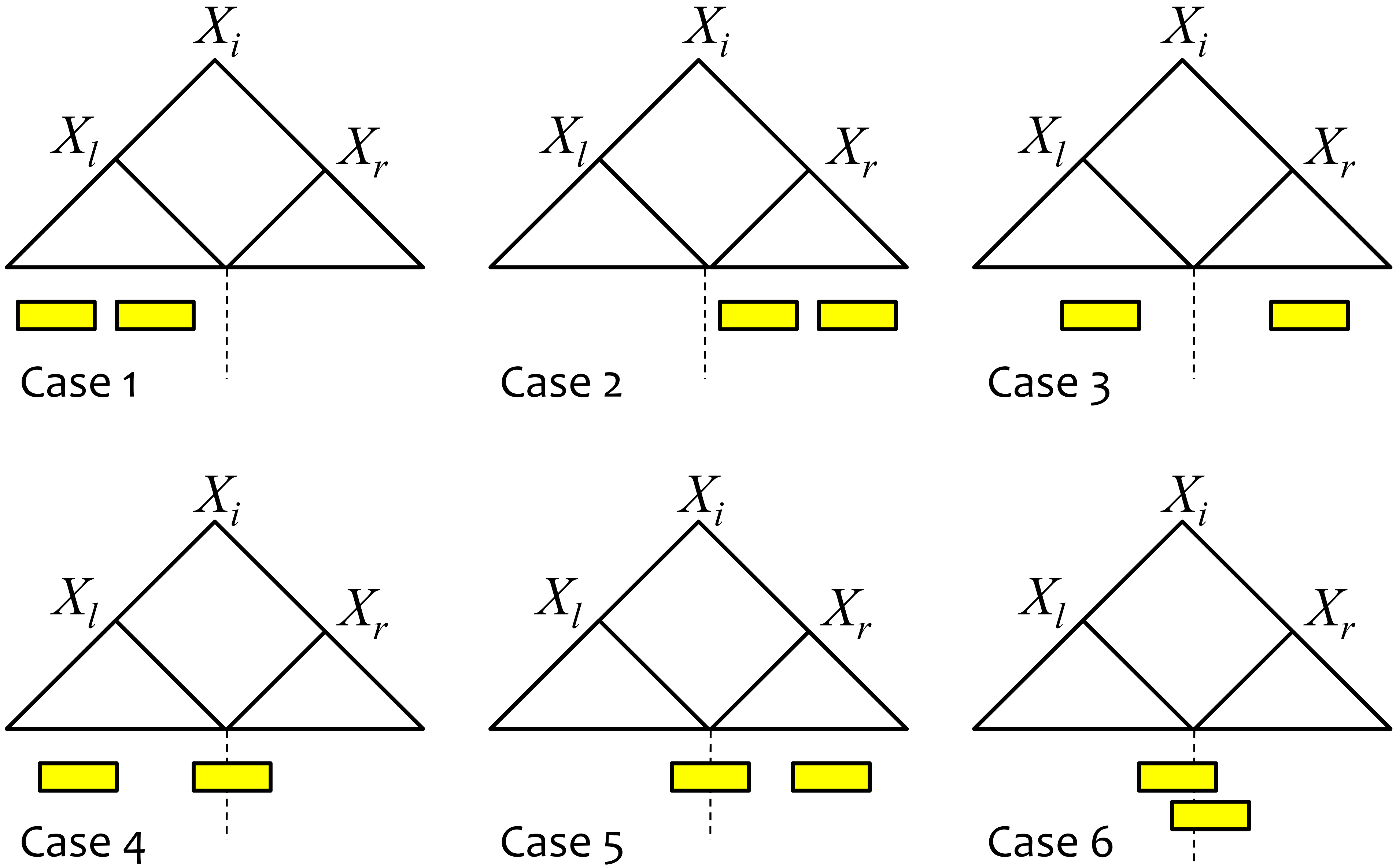
- Input: SLP $\mathcal{T}$ which generates text T
- Output: *A longest repeating substring (LRS)* of T



Example

$$T = \text{aabaabcabaabb}$$

Key Observation – 6 Cases of Occurrences of LRS



Algorithm to Compute LRS

Input: SLP $\mathcal{T}$

Output: LRS of text T

foreach variable X_i of SLP $\mathcal{T}$ **do**

compute LRS of Case 1;
compute LRS of Case 2;
compute LRS of Case 3;
compute LRS of Case 4;
compute LRS of Case 5;
compute LRS of Case 6;

return two positions and the length of the "longest" LRS above;

Algorithm to Compute LRS

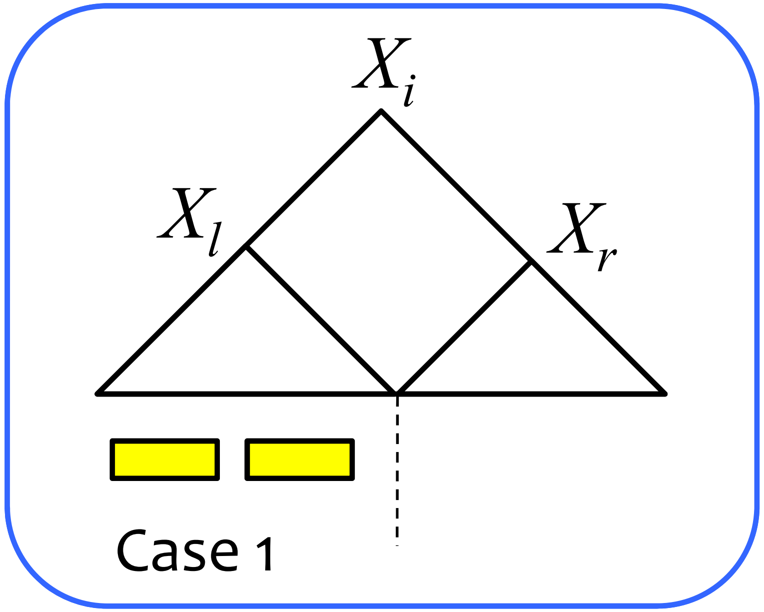
Input: SLP $\mathcal{T}$

Output: LRS of text T

foreach variable X_i of SLP $\mathcal{T}$ **do**

compute LRS of Case 1;
compute LRS of Case 2;
compute LRS of Case 3;
compute LRS of Case 4;
compute LRS of Case 5;
compute LRS of Case 6;

return two positions and the length of the "longest" LRS above;



Algorithm to Compute LRS

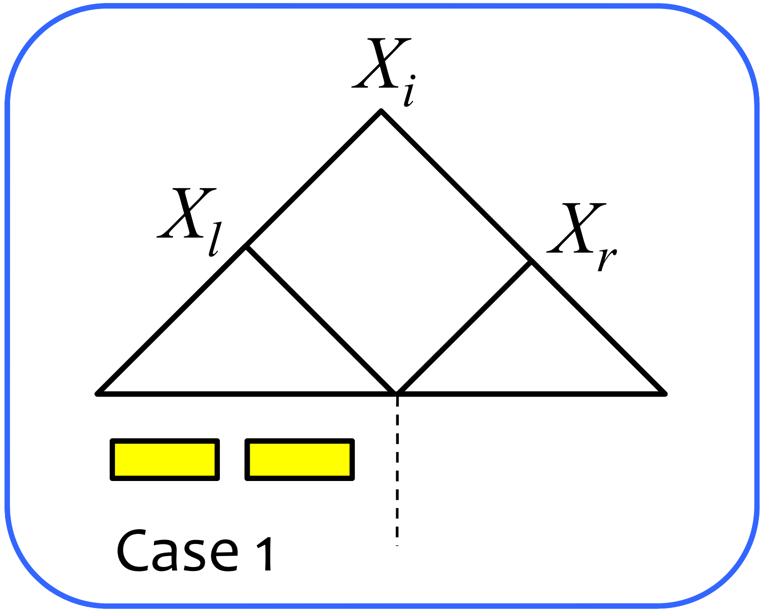
Input: SLP $\mathcal{T}$

Output: LRS of text T

foreach variable X_i of SLP $\mathcal{T}$ **do**

compute LRS of Case 1;
compute LRS of Case 2;
compute LRS of Case 3;
compute LRS of Case 4;
compute LRS of Case 5;
compute LRS of Case 6;

return two positions and the length of the "longest" LRS above;



Algorithm to Compute LRS

Input: SLP $\mathcal{T}$

Output: LRS of text T

foreach variable X_i of SLP $\mathcal{T}$ **do**

compute LRS of Case 2;
compute LRS of Case 3;
compute LRS of Case 4;
compute LRS of Case 5;
compute LRS of Case 6;

return two positions and the length of the "longest" LRS above;

Algorithm to Compute LRS

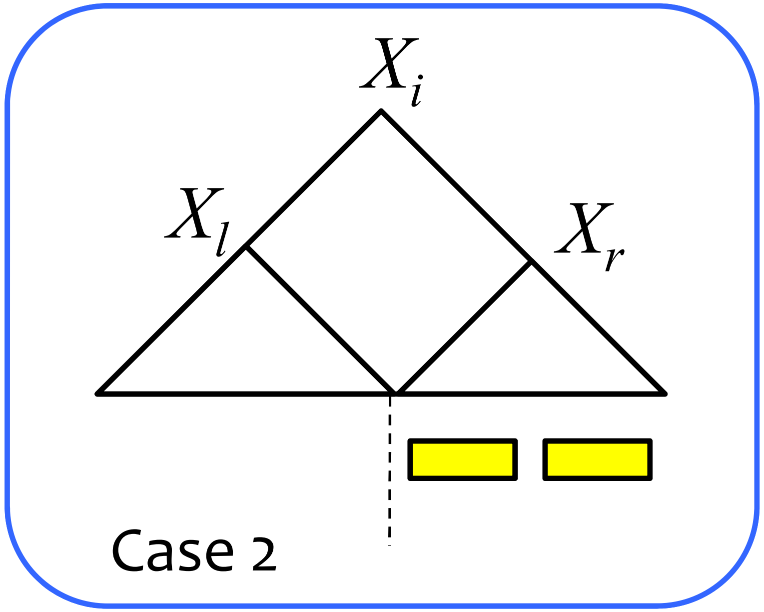
Input: SLP $\mathcal{T}$

Output: LRS of text T

foreach variable X_i of SLP $\mathcal{T}$ **do**

compute LRS of Case 2;
compute LRS of Case 3;
compute LRS of Case 4;
compute LRS of Case 5;
compute LRS of Case 6;

return two positions and the length of the "longest" LRS above;



Algorithm to Compute LRS

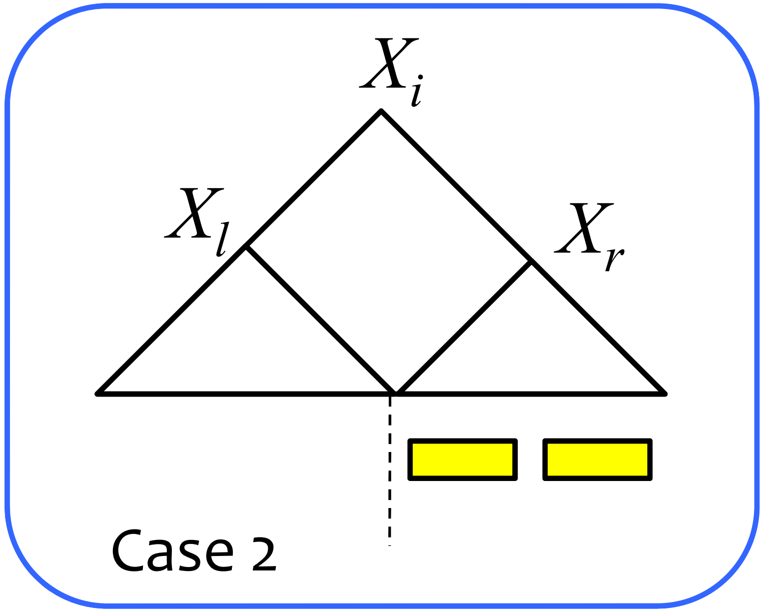
Input: SLP $\mathcal{T}$

Output: LRS of text T

foreach variable X_i of SLP $\mathcal{T}$ **do**

compute LRS of Case 2;
compute LRS of Case 3;
compute LRS of Case 4;
compute LRS of Case 5;
compute LRS of Case 6;

return two positions and the length of the "longest" LRS above;



Algorithm to Compute LRS

Input: SLP $\mathcal{T}$

Output: LRS of text T

foreach variable X_i of SLP $\mathcal{T}$ **do**

compute LRS of Case 3;
compute LRS of Case 4;
compute LRS of Case 5;
compute LRS of Case 6;

return two positions and the length of the "longest" LRS above;

Algorithm to Compute LRS

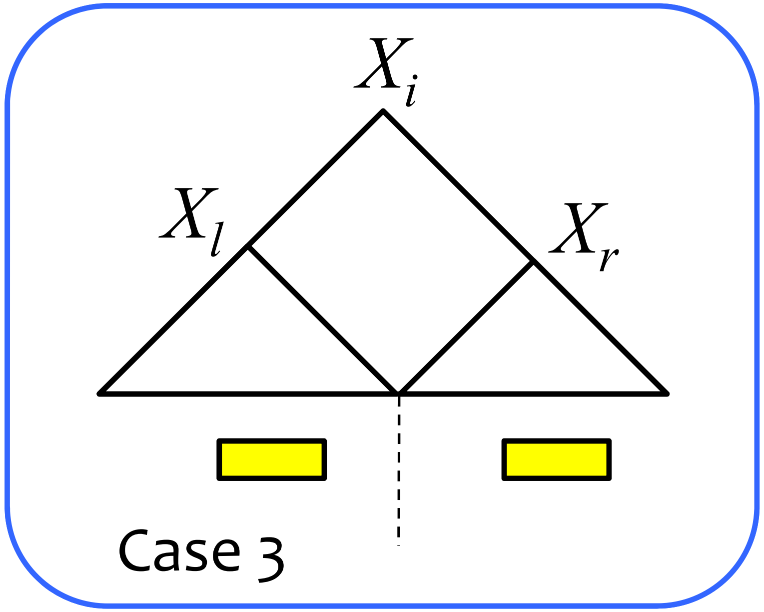
Input: SLP $\mathcal{T}$

Output: LRS of text T

foreach variable X_i of SLP $\mathcal{T}$ **do**

compute LRS of Case 3;
compute LRS of Case 4;
compute LRS of Case 5;
compute LRS of Case 6;

return two positions and the lengt
the "longest" LRS above;



LRS of X_i of Case 3 is the *longest common substring* of X_l and X_r.

Longest Common Substring of Two SLPs

Theorem 1 [Matsubara et al. 2009]

For every pair of variables X_l and X_r, we can compute a longest common substring of X_l and X_r in total of $O(n^4 \log n)$ time.

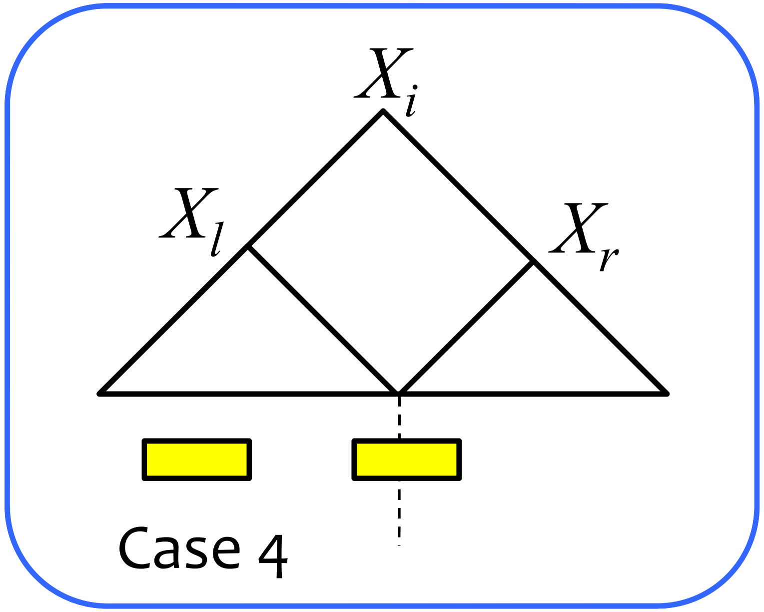
n is num. of variables in SLP

Algorithm to Compute LRS

Input: SLP $\mathcal{T}$

Output: LRS of text T

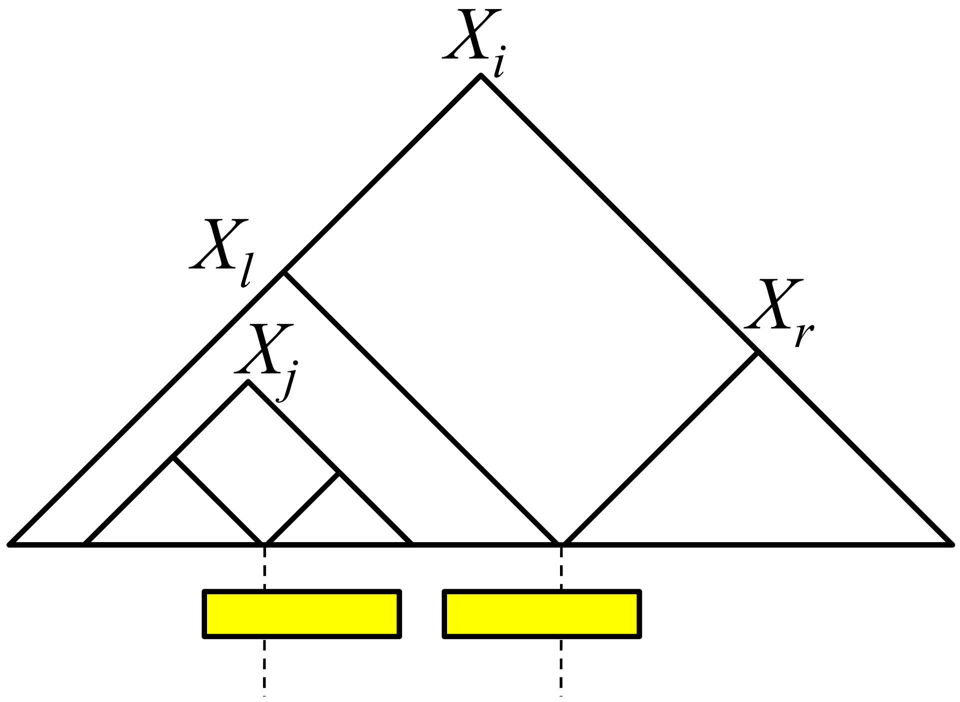
foreach variable X_i of SLP $\mathcal{T}$ **do**



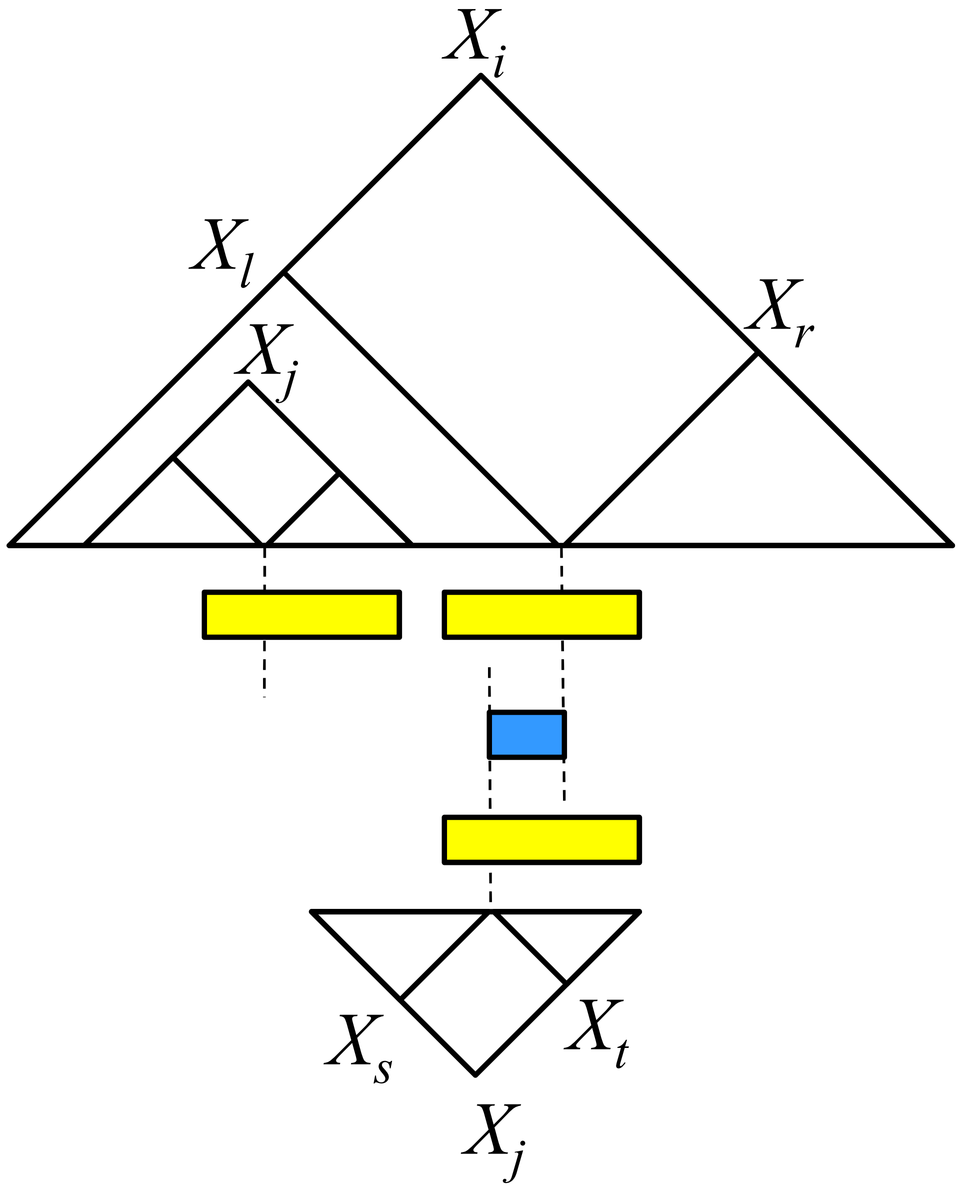
compute LRS of Case 3;
compute LRS of Case 4;
compute LRS of Case 5;
compute LRS of Case 6;

return two positions and the length of the "longest" LRS above;

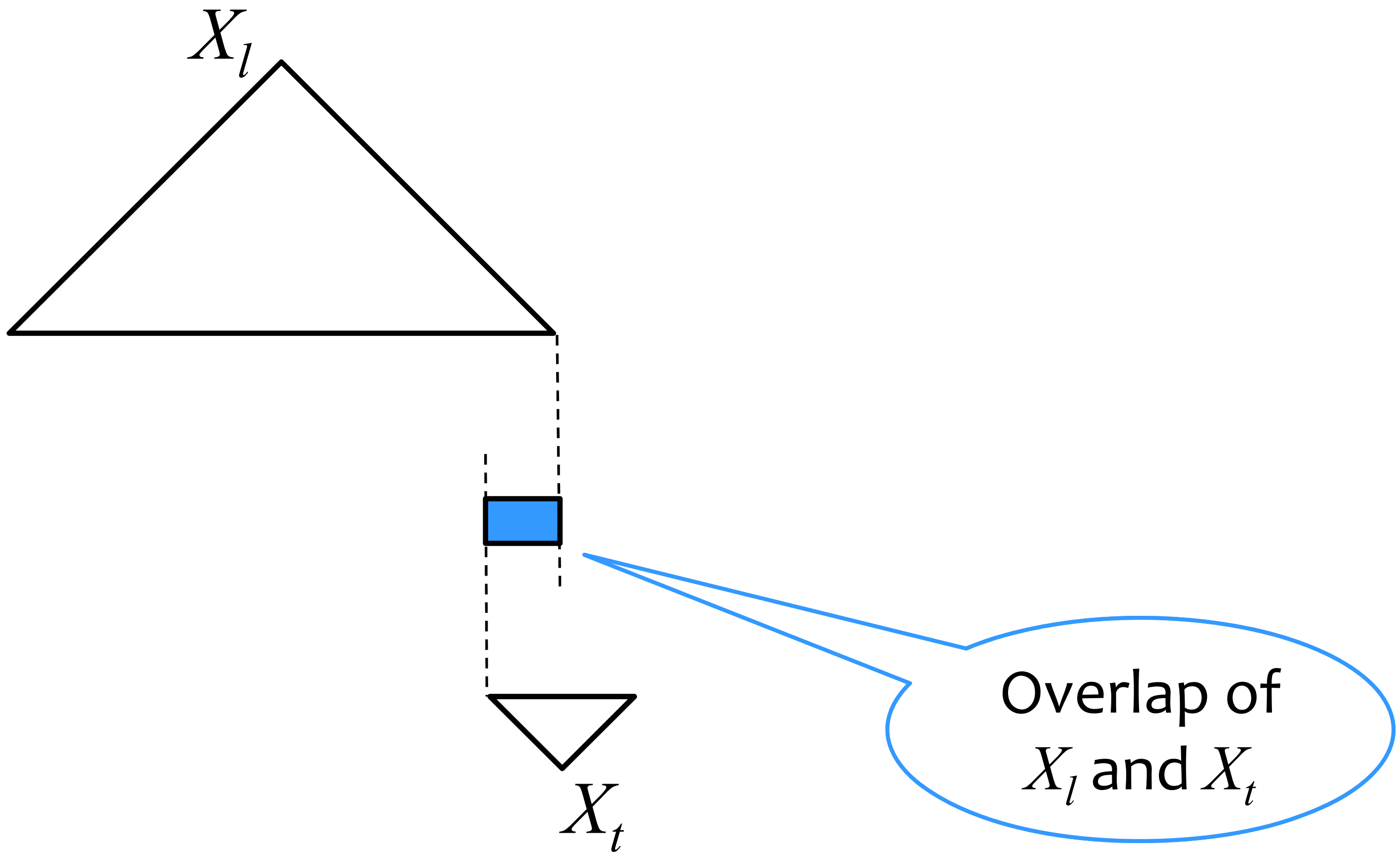
Case 4



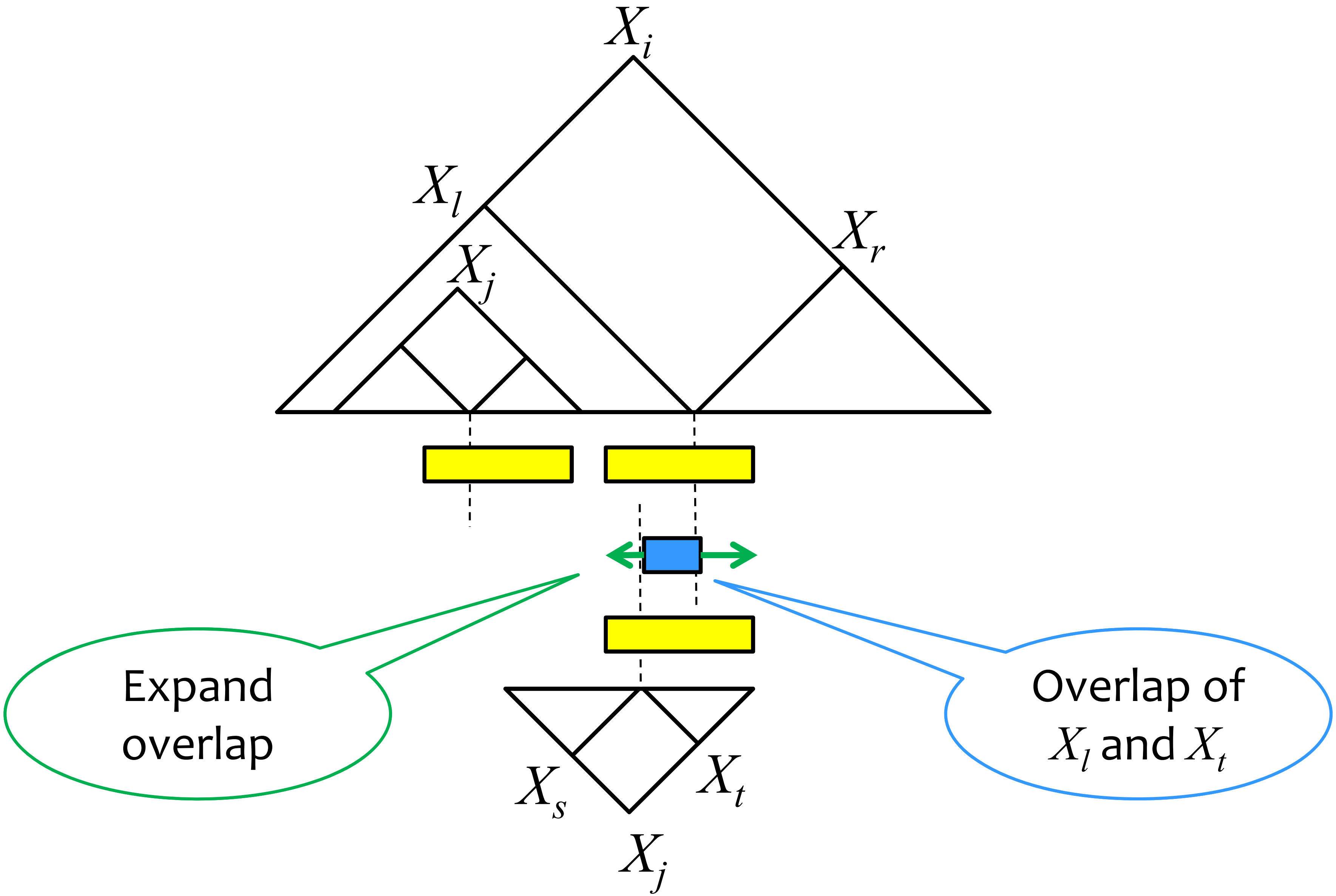
Case 4-1



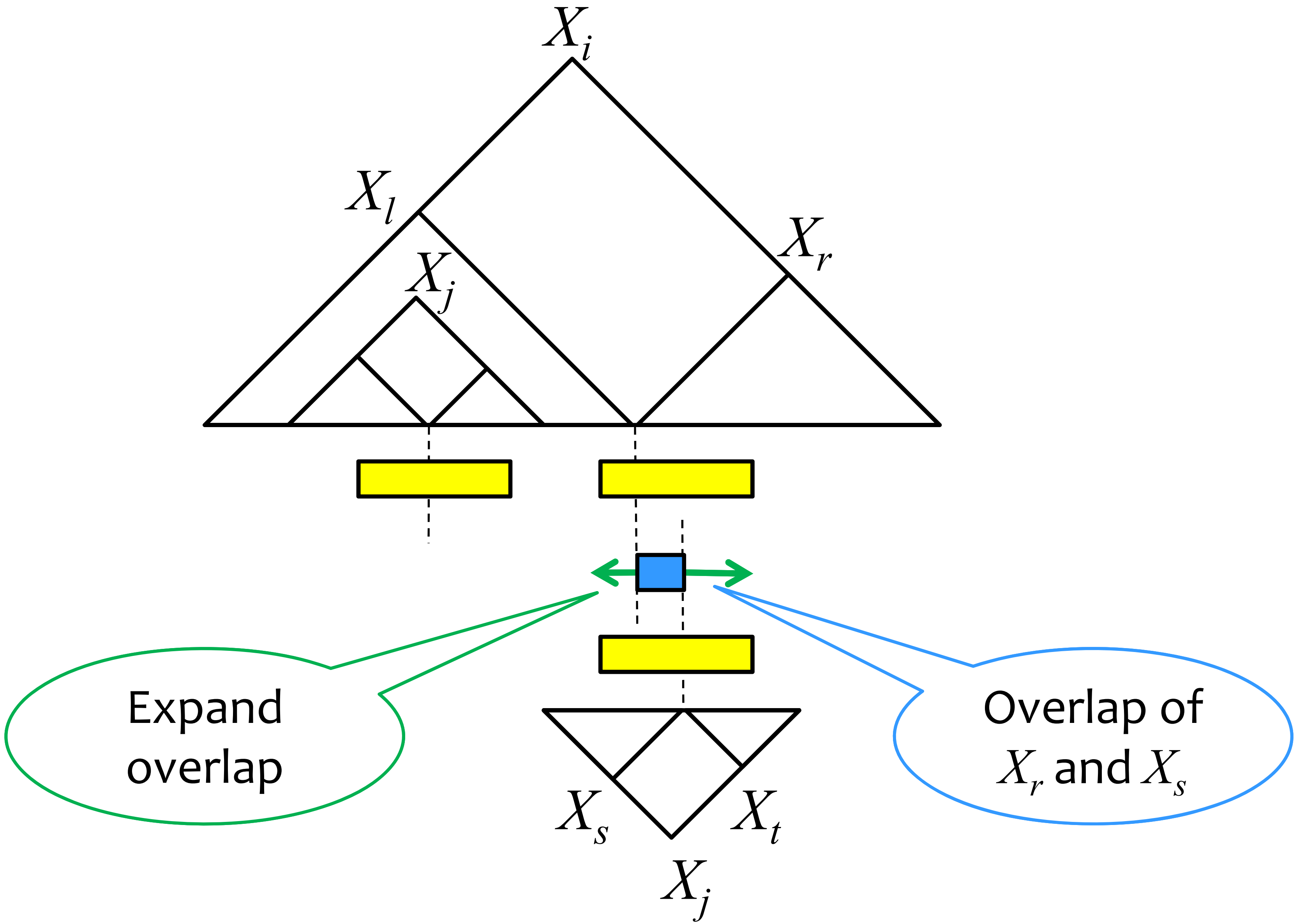
Case 4-1



Case 4-1



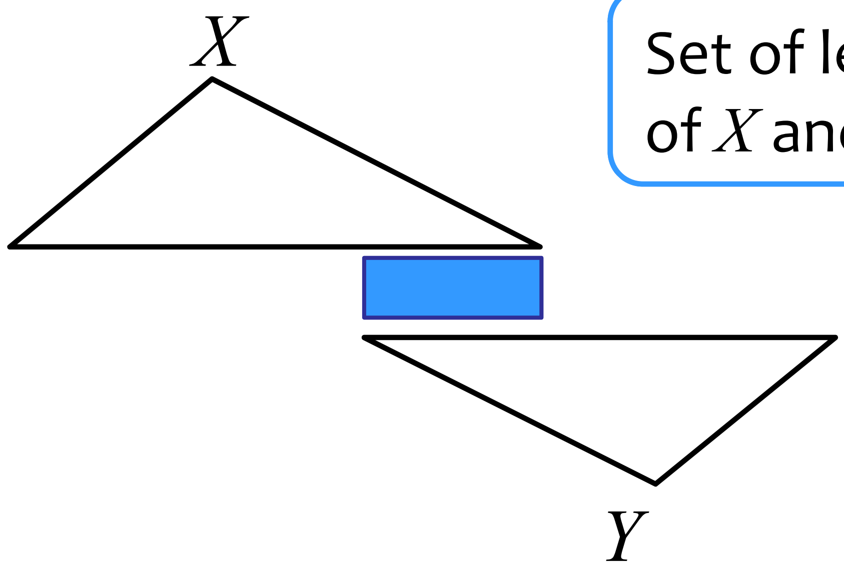
Case 4-2



Set of Overlaps

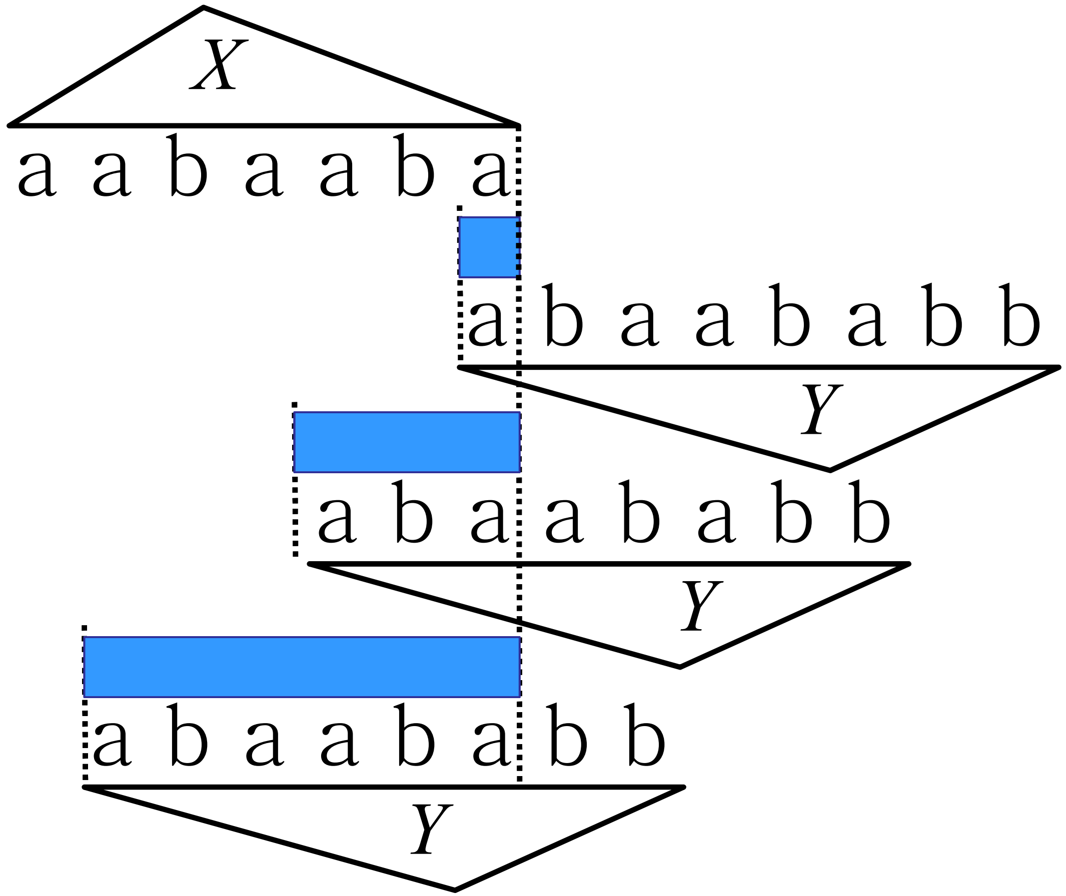
$$OL(X, Y) = \{k > 0 \mid X[|X| - k + 1 : |X|] = Y[1 : k]\}$$

Set of length of overlaps of X and Y



Set of Overlaps

$OL(\text{aabaaba}, \text{abaababb}) = \{1, 3, 6\}$



Set of Overlaps

Lemma 1 [Kaprinski et al. 1997]

For every pair of variables X_i and Y_j,
$OL(X_i, Y_j)$ forms $O(n)$ arithmetic progressions.

Lemma 2 [Kaprinski et al. 1997]

For every pair of variables X_i and Y_j,
$OL(X_i, Y_j)$ can be computed in total of $O(n^4 \log n)$ time.

n is num. of variables in SLP

Case 4

Lemma 3

For every variable X_i, a longest repeating substring in Case 4 can be computed in $O(n^3 \log n)$ time.

[Sketch of proof]

- We can expand all elements of each arithmetic progression of $OL(X_i, X_j)$ in $O(n \log n)$ time.
- The size of $OL(X_i, X_j)$ is $O(n)$ by Lemma 1.
- There are at most n-1 descendants X_j of X_i.

Algorithm to Compute LRS

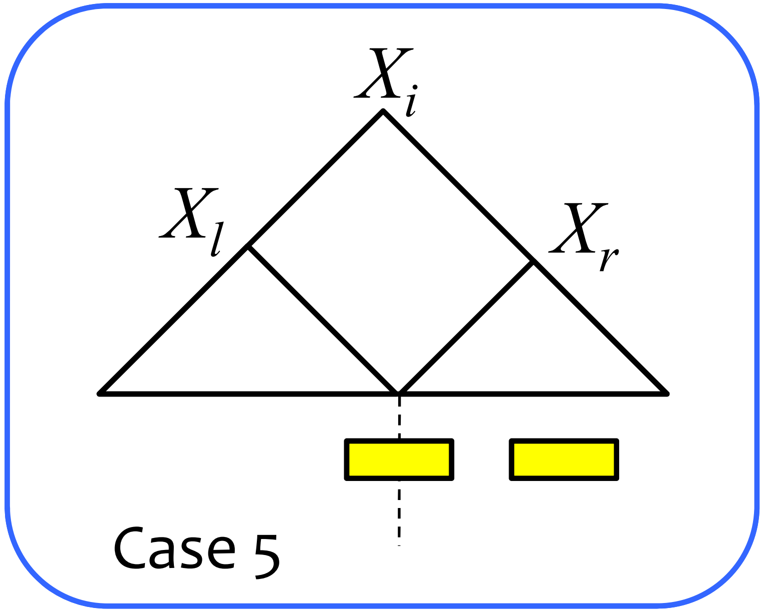
Input: SLP $\mathcal{T}$

Output: LRS of text T

foreach variable X_i of SLP $\mathcal{T}$ **do**

compute LRS of Case 3;
compute LRS of Case 4;
compute LRS of Case 5;
compute LRS of Case 6;

return two positions and the length of the "longest" LRS above;



Symmetric to Case 4

Algorithm to Compute LRS

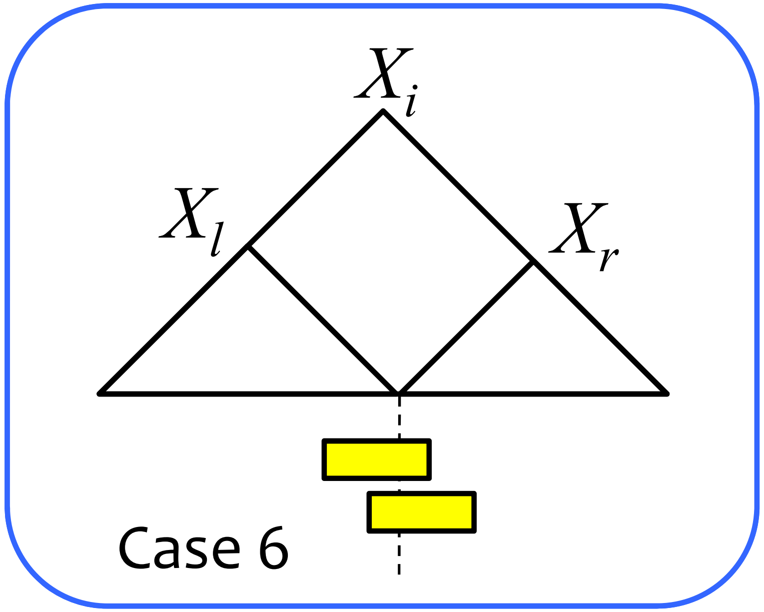
Input: SLP $\mathcal{T}$

Output: LRS of text T

foreach variable X_i of SLP $\mathcal{T}$ **do**

compute LRS of Case 3;
compute LRS of Case 4;
compute LRS of Case 5;
compute LRS of Case 6;

return two positions and the length of the "longest" LRS above;



Similarly to Case 4

Finding Longest Repeating Substring

Theorem 2

For any SLP $\mathcal{T}$ which generates text T, we can compute an LRS of T in $O(n^4 \log n)$ time.

n is num. of variables in SLP $\mathcal{T}$

Finding Longest *Non-Overlapping* Repeating Substring

- Input: SLP $\mathcal{T}$ which generates text T
- Output: *A longest non-overlapping repeating substring (LNRS)* of T

Example

$T =$ abababababab

LRS of T is abababab

LRNS of T is abab

Finding Longest *Non-Overlapping* Repeating Substring

Theorem 3

For any SLP $\mathcal{T}$ which generates text T, we can compute an LNRS of T in $O(n^6 \log n)$ time.

n is num. of variables in SLP $\mathcal{T}$

Finding Most Frequent Substring

- Input: SLP $\mathcal{T}$ which generates text T
- Output: *A most frequent substring (MFS)* of T

The solution is always
the empty string ε.

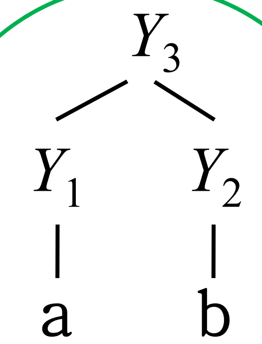
Finding Most Frequent Substring

- Input: SLP $\mathcal{T}$ which generates text T
- Output: *A most frequent substring (MFS)* of T
  of length 2

T

Algorithm to Compute MFS

$|\Sigma|^2$ substrings of length 2



Input: SLP $\mathcal{T}$

Output: MFS of text T

foreach substring P of T of length 2 **do**
  construct an SLP $\mathcal{P}$ which generates substring P;
  compute num. of occurrences of P in T;

return substring of maximum num. of occurrences;

Lemma 4

For every pair of variables X_i and Y_j, the number of occurrences of Y_j in X_i can be computed in total of $O(n^2)$ time.

Finding Most Frequent Substring

Theorem 4

For any SLP $\mathcal{T}$ which generates text T, we can compute an MFS of T of length 2 in $O(|\Sigma|^2 n^2)$ time.

n is num. of variables in SLP $\mathcal{T}$

Finding Most Frequent *Non-Overlapping* Substring

- Input: SLP $\mathcal{T}$ which generates text T
- Output: *A most frequent non-overlapping substring (MFNS)* of T of length 2

Example

$T =$ aaaaababab

MFS of T of length 2 is aa

MFNS of T of length 2 is ab

Finding Most Frequent *Non-Overlapping* Substring

Theorem 5

For any SLP $\mathcal{T}$ which generates text T, we can compute an MFNS of T of length 2 in $O(n^4 \log n)$ time.

n is num. of variables in SLP $\mathcal{T}$

Computing Left and Right Contexts of Given Pattern

- Input: Two SLPs $\mathcal{T}$ and $\mathcal{P}$ which generate text T and pattern P, respectively
- Output: Substring $\alpha P \beta$ of T such that
  - α (resp. β) always precedes (resp. follows) P in T
  - α and β are as long as possible

Example

$T =$ bbaabaabbaabb

$P =$ ab

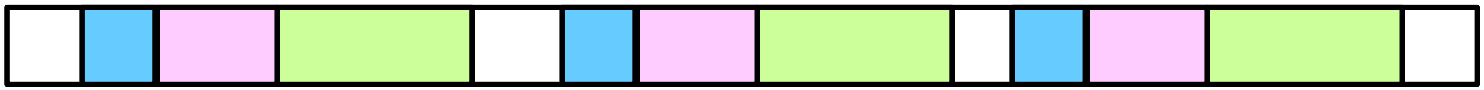
$\alpha =$ ba

$\beta = \varepsilon$

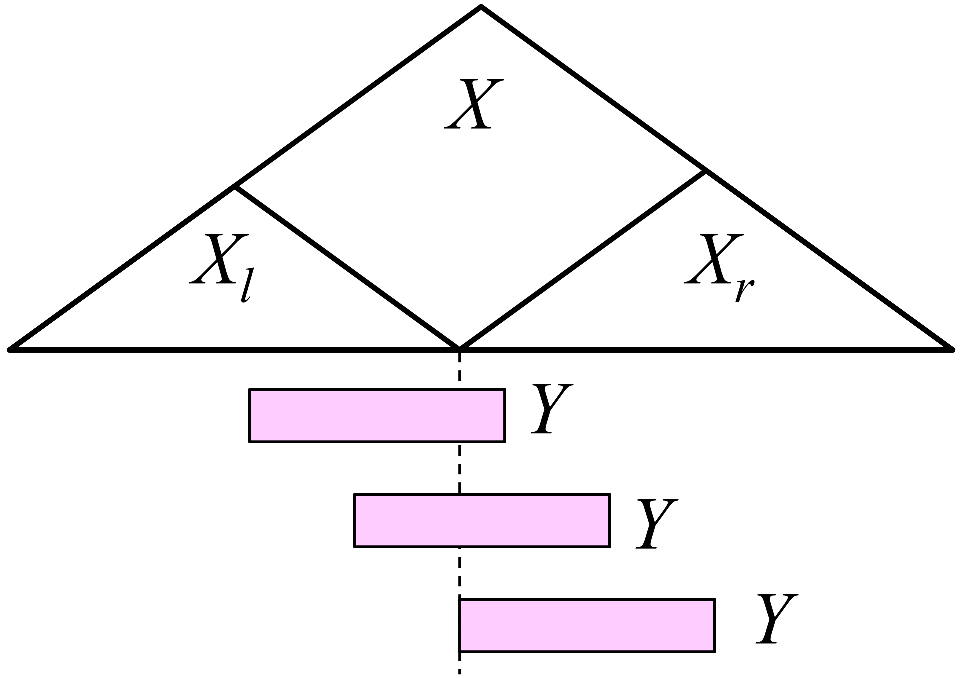
Computing Left and Right Contexts of Given Pattern

- Examples of applications of computing left and right contexts of patterns are:
  - Blog spam detection [Narisawa et al. 2007]
  - Compute maximal extension of most frequent substrings (MFS)

T



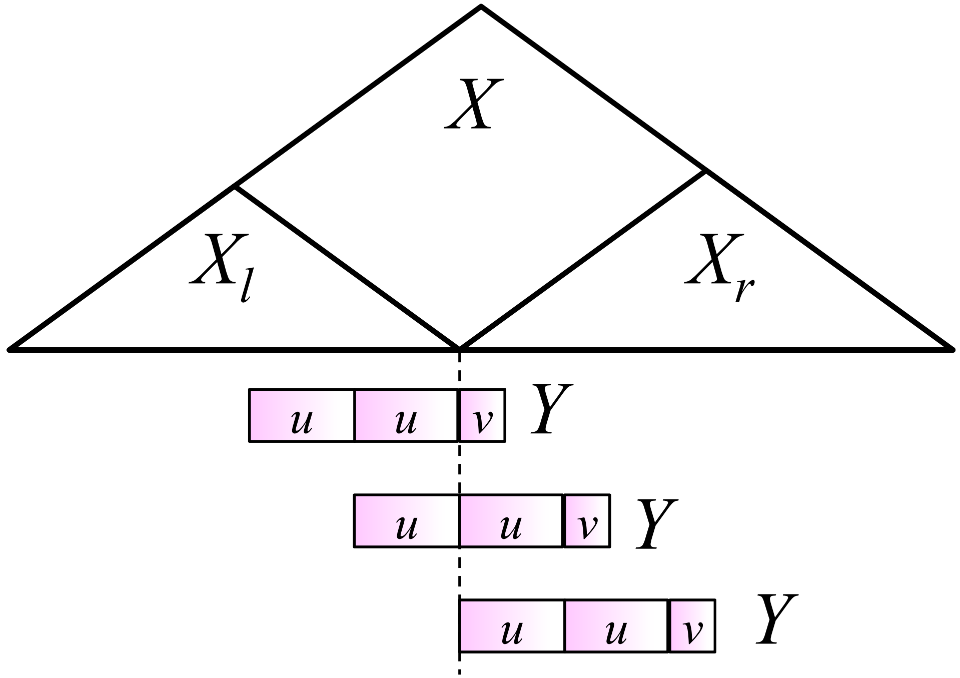
Boundary Lemma [1/2]



Lemma 5 [Miyazaki et al. 1997]

For any SLP variables $X = X_l X_r$ and Y, the occurrences of Y that touch or cover the boundary of X form a *single arithmetic progression.*

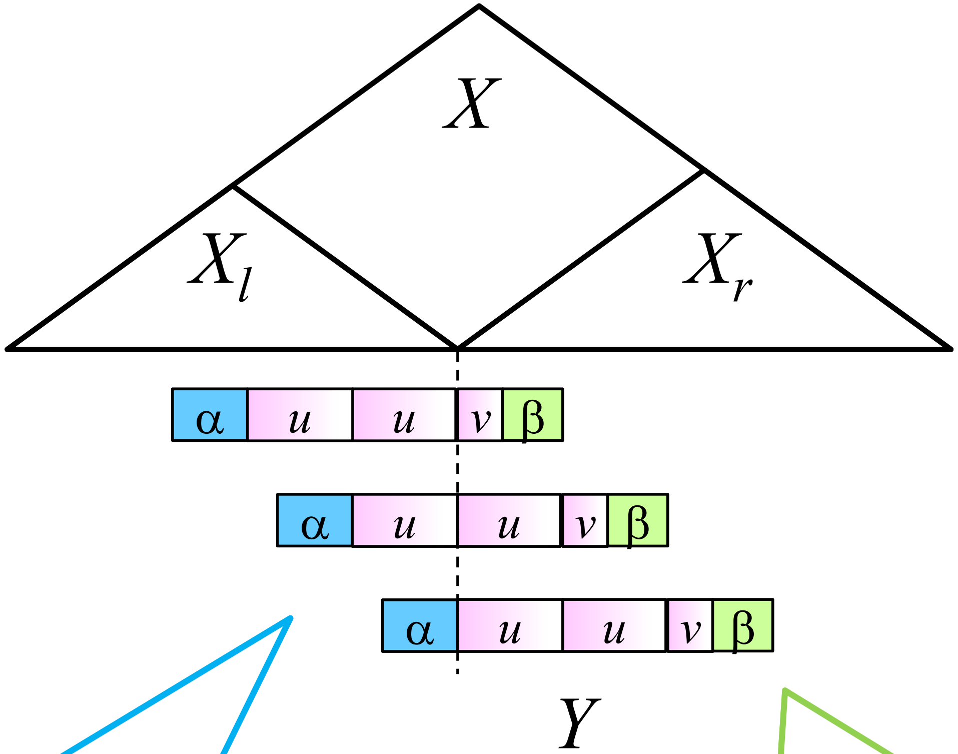
Boundary Lemma [2/2]



Lemma 5 [Miyazaki et al. 1997]

(Cont.) If the number of elements in the progression is more than 2, then the step of the progression is the smallest period of Y.

Left and Right Contexts



The left context α of *Y* in *X* is a suffix of *u*.

The right context β of *Y* in *X* is a prefix of $uv[|v|:|uv|]$.

Computing Left and Right Contexts of Given Pattern

Theorem 6

For any SLPs $\mathcal{T}$ and $\mathcal{P}$ which generate text T and pattern P, respectively, we can compute the left and right contexts of P in T in $O(n^4 \log n)$ time.

n is num. of variables in SLP $\mathcal{T}$

Conclusions and Future Work

- We presented polynomial time algorithms to find characteristic substrings of given SLP-compressed texts.
  - Our algorithms are more efficient than any algorithms that work on uncompressed strings.
- Would it be possible to efficiently find other types of substrings from SLP-compressed texts?
  - Squares (substrings of form xx)
  - Cubes (substrings of form xxx)
  - Runs (maximal substrings of form x^k with $k \geq 2$)
  - Gapped palindromes (substrings of form xyx^R with $|y| \geq 1$)