Answer Set Programming を用いた圧縮指標の計算

番原 睦則[1]、　クップル ドミニク[2]

[1]: 名古屋大学, [2]: 山梨大学

設定

ユーザー入力

- $T[1..n]$：文字列
- n: T の長さ

定義 (予備知識)

- $T[i..j]$: i から始まり j で終わる T の部分文字列
- $\mathrm{occ}(T[i..j]) := \{k \mid T[k..k+(j-i)] = T[i..j]\}$ を $T[i..j]$ の T 中の出現位置集合
- $\mathrm{cover}(T[i..j]) := \{k \mid \exists k \in occ(s), k \leq k \leq k+j-i\}$ を T 中における $T[i..j]$ のカバー位置集合とする
- $k \in \mathrm{cover}(T[i..j])$ のとき、位置 k は部分文字列 $T[i..j]$ をカバーすると言う
- $[x, y]$ は個数 x と サイズ y を示す

アトラクタ
Smallest String Attractor

アトラクタ

定義 (アトラクタ)

- $\Gamma \subseteq [1..n]$ とする
- T の任意の部分文字列 $T[i..j]$ について、$p \in \mathrm{cover}(T[i..j])$ になる位置 $p \in \Gamma$ が存在すると、Γ をアトラクタと呼ぶ

目的：一番要素が少ないアトラクタを計算する

定義 (minimal substring)

- 部分文字列 S にとして、S のすべて S より短い部分文字列は T の中にもっと多く出現する場合、S は minimal と呼ぶ
- つまり、任意 $S' = S[i..j], 1 \leq i \leq j \leq |S|$ かつ $S' \neq S$ に対して、$|\mathrm{occ}(S[i..j])| > |\mathrm{occ}(S)|$ である
- $\mathcal{M} \subset \{[i..j] : i, j \in [1..n]\}$ は minimal substring の集合を示す

定理 (Bannai+'22)

$\forall I \in \mathcal{M} : \Gamma \cap \mathit{cover}(I) \neq \emptyset \Rightarrow \forall 1 \leq i \leq j \leq n : \Gamma \cap \mathit{cover}([i..j]) \neq \emptyset.$

- つまり、$\mathcal{M}$ で Γ はアトラクタかどうかを判断できる
- $C : \mathcal{M} \to \{i \in [1..n]\}$ の写像を作る

前処理

- $C : \mathcal{M} \to \{i \in [1..n]\}$ の写像を作る
- ASP で C を任意 minimal substring $T[i..j]$ と $p \in C([i..j])$ に対して

`c(i,j,p).` で表現する。

文字列長さ n に対して、ASP 入力は $|C| \leq n^2(n-1)/2$ のビット

例

```
#const n=6.
c(6,6,2).
c(6,6,4).
c(6,6,6).
c(1,1,1).
c(3,5,3).
c(3,5,4).
c(3,5,5).
c(5,5,3).
c(5,5,5).
```

	1	2	3	4	5	6
$T =$	b	a	n	a	n	a
$\mathit{cover}(\underline{\text{a}}) =$		●		●		●
$\mathit{cover}(\text{an}) =$		●	·	●	·	
$\mathit{cover}(\text{ana}) =$		●	·	●	·	·
$\mathit{cover}(\text{anan}) =$		●	·	·	·	
$\mathit{cover}(\text{anana}) =$		●	·	·	·	·
$\mathit{cover}(\underline{\text{b}}) =$	●					
$\mathit{cover}(\text{ba}) =$	●	·				
$\mathit{cover}(\text{ban}) =$	●	·	·			
$\mathit{cover}(\text{bana}) =$	●	·	·	·		
$\mathit{cover}(\text{banan}) =$	●	·	·	·	·	
$\mathit{cover}(\text{banana}) =$	●	·	·	·	·	·
$\mathit{cover}(\underline{\text{n}}) =$			●		●	
$\mathit{cover}(\text{na}) =$			●	·	●	·
$\mathit{cover}(\underline{\text{nan}}) =$			●	·	·	
$\mathit{cover}(\text{nana}) =$			●	·	·	·

下線の部分文字列は minimal

Attractor ASP encoding

code:

```
{ p(1..n) }.
s(S,E) :- c(S,E,_).
:- not 1 { p(P) : c(S,E,P) }, s(S,E).
#minimize { 1,P : p(P) }.
#show p/1.
```

Attractor ASP encoding

code:

```
{ p(1..n) }.
s(S,E) :- c(S,E,_).
:- not 1 { p(P) : c(S,E,P) }, s(S,E).
#minimize { 1,P : p(P) }.
#show p/1.
```

1. $p(X) :\Leftrightarrow X \in \Gamma$

Attractor ASP encoding

code:

```
{ p(1..n) }.
s(S,E) :- c(S,E,_).
:- not 1 { p(P) : c(S,E,P) }, s(S,E).
#minimize { 1,P : p(P) }.
#show p/1.
```

1. $p(X) :\Leftrightarrow X \in \Gamma$
2. interval $[S..E]$ is minimal substring

Attractor ASP encoding

code:

```
{ p(1..n) }.
s(S,E) :- c(S,E,_).
:- not 1 { p(P) : c(S,E,P) }, s(S,E).
#minimize { 1,P : p(P) }.
#show p/1.
```

1. $p(X) :\Leftrightarrow X \in \Gamma$
2. interval $[S..E]$ is minimal substring
3. for each minimal substring $[S..E]$, at least one covered position is in Γ

$$\forall [i..j] \in \mathcal{M} : \Gamma \cap [i..j] \neq \emptyset$$

$$[|\mathcal{M}| \subseteq \mathcal{O}(n^2), \mathcal{O}(n)]$$

Attractor ASP encoding

code:

```
{ p(1..n) }.
s(S,E) :- c(S,E,_).
:- not 1 { p(P) : c(S,E,P) }, s(S,E).
#minimize { 1,P : p(P) }.
#show p/1.
```

1. $p(X) :\Leftrightarrow X \in \Gamma$
2. interval $[S..E]$ is minimal substring
3. for each minimal substring $[S..E]$, at least one covered position is in Γ

$$\forall [i..j] \in \mathcal{M} : \Gamma \cap [i..j] \neq \emptyset$$

$$[|\mathcal{M}| \subseteq \mathcal{O}(n^2), \mathcal{O}(n)]$$

4. minimize size of Γ

Attractor ASP encoding

code:

```
{ p(1..n) }.
s(S,E) :- c(S,E,_).
:- not 1 { p(P) : c(S,E,P) }, s(S,E).
#minimize { 1,P : p(P) }.
#show p/1.
```

1. $p(X) :\Leftrightarrow X \in \Gamma$
2. interval $[S..E]$ is minimal substring
3. for each minimal substring $[S..E]$, at least one covered position is in Γ

$$\forall [i..j] \in \mathcal{M} : \Gamma \cap [i..j] \neq \emptyset$$

$$[|\mathcal{M}| \subseteq \mathcal{O}(n^2), \mathcal{O}(n)]$$

4. minimize size of Γ
5. output elements of Γ

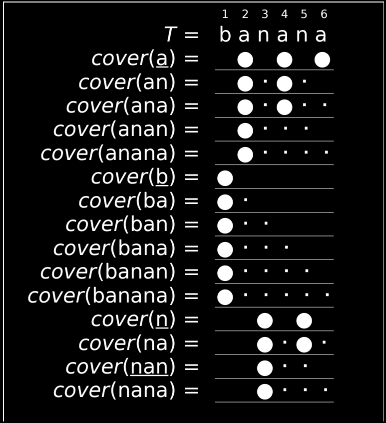
例

clingo の出力:

```
p(1)
p(4)
p(5)
```

つまり

- $\Gamma := \{1, 4, 5\}$ は最小のアトラクタ



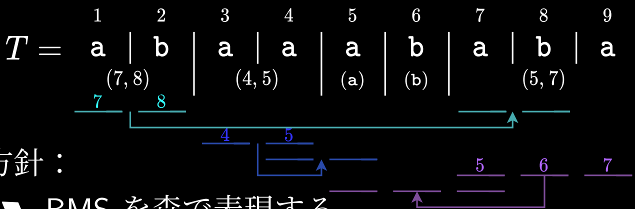
BMS

Smallest Bidirectional Macro Scheme

BMS Input

Σ を整数に写像 (例：ASCII)

- a ↦ 97, b ↦ 98



方針：

- BMS を森で表現する
- 各位置は参照先を持ちうる
- 参照先がない位置 v は root になる
- 一方で、v に参照先 w があると、w は v の親になる

文字列長さ n に対して、ASP 入力は n の文字

例

入力 $T :=$ abaaababa にとして、#const

```
n=10.
t(1,97).
t(2,98).
t(3,97).
t(4,97).
t(5,97).
t(6,98).
t(7,97).
t(8,98).
t(9,97).
```

BMS ASP encoding 1/2

code:

```
v(I) :- t(I,_).
e(I,J) :- t(I,A), t(J,A), I < J.
{ root(X) : v(X) }.
:- root(X), root(Y), e(X,Y).
{ r(X,Y) ; r(Y,X) } 1 :- e(X,Y).
:- root(X), r(X,_).
:- v(X), not root(X), not 1 { r(X,_)
    } 1.
#edge (X,Y): r(X,Y).
```

to continue · · ·

BMS ASP encoding 1/2

code:

```
v(I) :- t(I,_).
e(I,J) :- t(I,A), t(J,A), I < J.
{ root(X) : v(X) }.
:- root(X), root(Y), e(X,Y).
{ r(X,Y) ; r(Y,X) } 1 :- e(X,Y).
:- root(X), r(X,_).
:- v(X), not root(X), not 1 { r(X,_) } 1.
#edge (X,Y): r(X,Y).
```

1. each text position $i \in [1..n]$ is a vertex v_i

$$\forall i \in [1..n] : v_i := 1$$

defines n variables

to continue $\cdots$

BMS ASP encoding 1/2

code:

```
v(I) :- t(I,_).
e(I,J) :- t(I,A), t(J,A), I < J.
{ root(X) : v(X) }.
:- root(X), root(Y), e(X,Y).
{ r(X,Y) ; r(Y,X) } 1 :- e(X,Y).
:- root(X), r(X,_).
:- v(X), not root(X), not 1 { r(X,_) } 1.
#edge (X,Y): r(X,Y).
```

2. for all $i < j$ with $T[i] = T[j]$ create edge candidate $\{v_i, v_j\}$

$$\forall i,j \in [1..n], i < j \land (T[i] = T[j]) : \quad e_{i,j} := 1$$

defines $\mathcal{O}(n^2)$ variables

to continue $\cdots$

BMS ASP encoding 1/2

code:

```
v(I) :- t(I,_).
e(I,J) :- t(I,A), t(J,A), I < J.
{ root(X) : v(X) }.
:- root(X), root(Y), e(X,Y).
{ r(X,Y) ; r(Y,X) } 1 :- e(X,Y).
:- root(X), r(X,_).
:- v(X), not root(X), not 1 { r(X,_) } 1.
#edge (X,Y): r(X,Y).
```

3. each vertex can be a root

$$\forall i \in [1..n] : \mathrm{root}_i \in \{0, 1\}$$

defines n variables

to continue $\cdots$

BMS ASP encoding 1/2

code:

```
v(I) :- t(I,_).
e(I,J) :- t(I,A), t(J,A), I < J.
{ root(X) : v(X) }.
:- root(X), root(Y), e(X,Y).
{ r(X,Y) ; r(Y,X) } 1 :- e(X,Y).
:- root(X), r(X,_).
:- v(X), not root(X), not 1 { r(X,_) } 1.
#edge (X,Y): r(X,Y).
```

4. If vertices v_i and v_j are roots, there cannot be an edge candidate $\{v_i, v_j\}$

$$\forall i, j \in [1..n] :$$

$$\mathrm{root}_i \ \wedge \ \mathrm{root}_j \Rightarrow \neg e_{i,j}$$

$$[\mathcal{O}(n^2), \mathcal{O}(1)]$$

to continue $\cdots$

BMS ASP encoding 1/2

code:

```
v(I) :- t(I,_).
e(I,J) :- t(I,A), t(J,A), I < J.
{ root(X) : v(X) }.
:- root(X), root(Y), e(X,Y).
{ r(X,Y) ; r(Y,X) } 1 :- e(X,Y).
:- root(X), r(X,_).
:- v(X), not root(X), not 1 { r(X,_)
     } 1.
#edge (X,Y): r(X,Y).
```

5. If there is an edge candidate $\{v_i, v_j\}$, we are allowed to make either v_i parent of v_j, or vice versa

$$\forall i, j \in [1..n] :$$
$$e_{i,j} = 1 \Rightarrow r_{i,j} + r_{j,i} \leq 1$$
$$[\mathcal{O}(n^2), \mathcal{O}(1)]$$

to continue $\cdots$

BMS ASP encoding 1/2

code:

```
v(I) :- t(I,_).
e(I,J) :- t(I,A), t(J,A), I < J.
{ root(X) : v(X) }.
:- root(X), root(Y), e(X,Y).
{ r(X,Y) ; r(Y,X) } 1 :- e(X,Y).
:- root(X), r(X,_).
:- v(X), not root(X), not 1 { r(X,_) } 1.
#edge (X,Y): r(X,Y).
```

6. No root can have a parent

$$\forall i \in [1..n]:$$

$$\mathrm{root}_i \Leftrightarrow \nexists j \in [1..n] : r_{i,j} = 1$$

$$\Leftrightarrow \sum_{j=1}^{n} r_{i,j} = 0$$

$[\mathcal{O}(n), \mathcal{O}(n)]$

to continue $\cdots$

BMS ASP encoding 1/2

code:

```
v(I) :- t(I,_).
e(I,J) :- t(I,A), t(J,A), I < J.
{ root(X) : v(X) }.
:- root(X), root(Y), e(X,Y).
{ r(X,Y) ; r(Y,X) } 1 :- e(X,Y).
:- root(X), r(X,_).
:- v(X), not root(X), not 1 { r(X,_) } 1.
#edge (X,Y): r(X,Y).
```

7. A non-root node must have exactly one parent

$$\forall i \in [1..n] :$$

$$\neg \mathrm{root}_i \Leftrightarrow \sum_{j=1}^{n} r_{i,j} = 1$$

$$[\mathcal{O}(n), \mathcal{O}(n)]$$

to continue $\cdots$

BMS ASP encoding 1/2

code:

```
v(I) :- t(I,_).
e(I,J) :- t(I,A), t(J,A), I < J.
{ root(X) : v(X) }.
:- root(X), root(Y), e(X,Y).
{ r(X,Y) ; r(Y,X) } 1 :- e(X,Y).
:- root(X), r(X,_).
:- v(X), not root(X), not 1 { r(X,_) } 1.
#edge (X,Y): r(X,Y).
```

8. ASP-magic : enforce acyclicity on r

idea

transitive closure with variable $c_{i,j}$ such that

$$\forall i \in [1..n] : r_{i,j} \Rightarrow c_{i,j}$$

$[\mathcal{O}(n^2), \mathcal{O}(1)]$

$$\forall i, j, k \in [1..n] : c_{i,j} \wedge r_{j,k} \Rightarrow c_{i,k}$$

$[\mathcal{O}(n^3), \mathcal{O}(1)]$

$$\forall i, j \in [1..n] : c_{i,j} \Rightarrow \neg c_{j,i}$$

$[\mathcal{O}(n^2), \mathcal{O}(1)]$

BMS ASP encoding 2/2

code:

```
p(X) :- root(X).
p(X) :- r(X,_), X == 1.
p(X) :- r(X,Y), Y == 1.
p(X) :- r(X,Y), t(X-1,A), t(Y-1,B), A != B.
p(X) :- r(X,Y), not r(X-1,Y-1), v(X-1), v(Y-1).
#minimize { 1,X : p(X) }.
```

A factor starts

BMS ASP encoding 2/2

code:

```
p(X) :- root(X).
p(X) :- r(X,_), X == 1.
p(X) :- r(X,Y), Y == 1.
p(X) :- r(X,Y), t(X-1,A), t(Y-1,B), A != B.
p(X) :- r(X,Y), not r(X-1,Y-1), v(X-1), v(Y-1).
#minimize { 1,X : p(X) }.
```

A factor starts

1. at positions that are roots

$$\forall i \in [1..n] : \mathrm{root}_i \Rightarrow p_i$$

$[\mathcal{O}(n), \mathcal{O}(1)]$

improvement

Also possible to write:

$$\forall i \in [1..n] : \mathrm{root}_i \Rightarrow p_i \wedge p_{i+1}$$

$[\mathcal{O}(n), \mathcal{O}(1)]$

BMS ASP encoding 2/2

code:

```
p(X) :- root(X).
p(X) :- r(X,_), X == 1.
p(X) :- r(X,Y), Y == 1.
p(X) :- r(X,Y), t(X-1,A), t(Y-1,B), A != B.
p(X) :- r(X,Y), not r(X-1,Y-1), v(X-1), v(Y-1).
#minimize { 1,X : p(X) }.
```

A factor starts

2. at position 1

$$p_1 = 1$$

$$[\mathcal{O}(1), \mathcal{O}(1)]$$

BMS ASP encoding 2/2

code:

```
p(X) :- root(X).
p(X) :- r(X,_), X == 1.
p(X) :- r(X,Y), Y == 1.
p(X) :- r(X,Y), t(X-1,A), t(Y-1,B), A != B.
p(X) :- r(X,Y), not r(X-1,Y-1), v(X-1), v(Y-1).
#minimize { 1,X : p(X) }.
```

A factor starts

3. when it refers to position 1

$$\forall i \in [1..n] : r_{i,1} \Rightarrow p_i$$

$[\mathcal{O}(n), \mathcal{O}(1)]$

BMS ASP encoding 2/2

code:

```
p(X) :- root(X).
p(X) :- r(X,_), X == 1.
p(X) :- r(X,Y), Y == 1.
p(X) :- r(X,Y), t(X-1,A), t(Y-1,B), A != B.
p(X) :- r(X,Y), not r(X-1,Y-1), v(X-1), v(Y-1).
#minimize { 1,X : p(X) }.
```

A factor starts

4. when it cannot be prolonged to the left

$$\forall i, j \in [1..n] : \\ r_{i,j} \land (T[i-1] \neq T[j-1]) \Rightarrow p_i$$

$[\mathcal{O}(n^2), \mathcal{O}(1?)]$

info

maybe unnecessary condition?

BMS ASP encoding 2/2

code:

```
p(X) :- root(X).
p(X) :- r(X,_), X == 1.
p(X) :- r(X,Y), Y == 1.
p(X) :- r(X,Y), t(X-1,A), t(Y-1,B), A != B.
p(X) :- r(X,Y), not r(X-1,Y-1), v(X-1), v(Y-1).
#minimize { 1,X : p(X) }.
```

A factor starts

5. when the former position has a different reference

$$\forall i, j \in [1..n] : r_{i,j} \land \neg r_{i-1,j-1} \Rightarrow p_i = 1$$

$[\mathcal{O}(n^2), \mathcal{O}(1)]$

BMS ASP encoding 2/2

code:

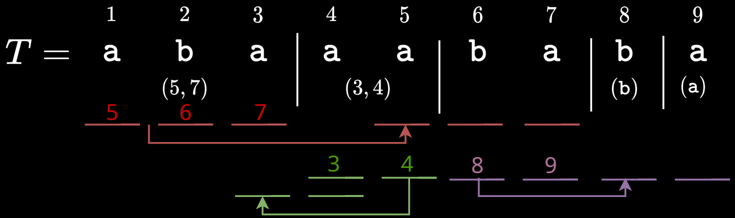
```
p(X) :- root(X).
p(X) :- r(X,_), X == 1.
p(X) :- r(X,Y), Y == 1.
p(X) :- r(X,Y), t(X-1,A), t(Y-1,B), A != B.
p(X) :- r(X,Y), not r(X-1,Y-1), v(X-1), v(Y-1).
#minimize { 1,X : p(X) }.
```

6. Minimize the factor starting positions p_i

例

clingo の出力:

```
p(1)
p(4)
p(6)
p(8)
p(9)
r(1,5)
r(2,6)
r(3,7)
r(4,3)
r(5,4)
r(6,8)
r(7,9)
root(8)
root(9)
```



改善

方針:

- root_i は必要？
  - $\mathrm{root}_i \Rightarrow p_i \wedge p_{i+1}$ は正しいけど、
  - $p_i \wedge p_{i+1} \Rightarrow \mathrm{root}_i$ にしても、最小の BMS を与える可能性がある
  - $\mathrm{root}_i \Leftrightarrow p_i \wedge p_{i+1}$ とすると、変数 root_i を定義しなくてもいい

SLP

Smallest Straight-Line Program

定理 (Bannai+'22)

A factorization $T = F_1 \cdots F_z$ for T is the grammar parsing of an SLP for T if and only if

- *for each factor F_x longer than 1, there exist two indices (u_x, v_x) with $u_x < v_x < x$ such that: $F_x = F_{u_x} \cdots F_{v_x}$, and*
- *for every pair of factors $F_x = F_{u_x} \cdots F_{v_x}$ and $F_y = F_{u_y} \cdots F_{v_y}$ longer than 1 ($|F_x| > 1$, $|F_y| > 1$), the intervals $\mathcal{I}_x := [u_x..v_x]$ and $\mathcal{I}_y := [u_y..v_y]$ are either disjoint or one is a sub-interval of the other ($\mathcal{I}_x \cup \mathcal{I}_y = \emptyset \vee \mathcal{I}_x \subseteq \mathcal{I}_y \vee \mathcal{I}_y \subseteq \mathcal{I}_x$).*

というわけで

- grammar parsing は特徴な BMS
- grammar parsing から簡単に SLP を導出できる（説明を割愛）
- 目的は最小の項が持つ grammar parsing を計算 (つまり、z を減らす)

方針

- 変数をテキスト位置 $[1..n]$ で表現しなくても、項の番号 $[1..z]$ で表現しやすい

項 F_x に対して、

- $s_x := 1 + \sum_{y=1}^{x-1} |F_y| \in [1..n]$ は項 F_x の開始位置を示す
- $\ell_x := |F_x|$ を示す
  - $\ell_x = s_{x+1} - s_x$
- $F_x = F_{u_x} \cdots F_{v_x}$ なら、$\mathrm{span}_x := v_x - u_x + 1$ は F_x で参照された項の個数を示す
  - $\ell_x = \sum_{y=u_x}^{\mathrm{span}_x} |F_y|$
  - $s_{v_x+1} = s_{u_x} + \sum_{y=u_x}^{\mathrm{span}_x} |F_y|$

前処理

- 写像 lce : $[1..n] \times [1..n] \to [1..n]$ を計算する
  $\mathrm{lce}(i,j) = \ell :\Leftrightarrow T[i..i+\ell-1] = T[j..j+\ell-1] \wedge (\max(i,j) = n-\ell+1 \vee T[i..i+\ell] \neq T[j..j+\ell])$
- lpnf : $[1..n] \to [1..n]$ は位置 i に対して i の前から始まる接尾辞の最大 lce 値を示す
  $\mathrm{lpnf}(i) := \max_{j\in[1..i-1]} \min(\mathrm{lce}(i,j), i-j)$
- ASP ですべての lce・lpnf 値を保存する

```
lce(i,j,l).
lpnf(i,l).
```

で表現する。

文字列長さ n に対して、ASP 入力は $|\mathrm{lpnf}| = n, |\mathrm{lce}| = n(n-1)/2$ の整数 $\in [1..n]$

i	1	2	3	4	5	6	7	8	9	10	11	12	13
$T =$	a	b	a	a	b	a	b	a	a	b	a	a	b

例

```
#const n=13.
lce(2,7,5).
lce(2,8,0).
lce(2,9,0).
lce(2,10,4).
...
lpnf(2,0).
lpnf(3,1).
lpnf(4,3).
lpnf(5,2).
...
```

SLP ASP encoding 1/2

code:

```
p(I) :- lpnf(I,_).
1 { z(X) : X = 1..n } 1.
s(1,1).
s(Z+1,n+1) :- z(Z).
1 { s(X+1,I) : p(I), I > J, s(X,J) }
    1 :- p(X), X < Z, z(Z).
l(X,L) :- L = J - I, s(X,I), s(X+1,J)
    .
```

SLP ASP encoding 1/2

code:

```
p(I) :- lpnf(I,_).
1 { z(X) : X = 1..n } 1.
s(1,1).
s(Z+1,n+1) :- z(Z).
1 { s(X+1,I) : p(I), I > J, s(X,J) }
     1 :- p(X), X < Z, z(Z).
l(X,L) :- L = J - I, s(X,I), s(X+1,J).
```

1. set p_i for all text positions $i \in [1..n]$

to continue ···

SLP ASP encoding 1/2

code:

```
p(I) :- lpnf(I,_).
1 { z(X) : X = 1..n } 1.
s(1,1).
s(Z+1,n+1) :- z(Z).
1 { s(X+1,I) : p(I), I > J, s(X,J) }
    1 :- p(X), X < Z, z(Z).
l(X,L) :- L = J - I, s(X,I), s(X+1,J)
    .
```

1. set p_i for all text positions $i \in [1..n]$
2. select the number of factors z in $[1..n]$

to continue $\cdots$

SLP ASP encoding 1/2

code:

```
p(I) :- lpnf(I,_).
1 { z(X) : X = 1..n } 1.
s(1,1).
s(Z+1,n+1) :- z(Z).
1 { s(X+1,I) : p(I), I > J, s(X,J) }
    1 :- p(X), X < Z, z(Z).
l(X,L) :- L = J - I, s(X,I), s(X+1,J)
    .
```

1. set p_i for all text positions $i \in [1..n]$
2. select the number of factors z in $[1..n]$
3. the first factor starts at position 1: $s_1 = 1$

SLP ASP encoding 1/2

code:

```
p(I) :- lpnf(I,_).
1 { z(X) : X = 1..n } 1.
s(1,1).
s(Z+1,n+1) :- z(Z).
1 { s(X+1,I) : p(I), I > J, s(X,J) }
    1 :- p(X), X < Z, z(Z).
l(X,L) :- L = J - I, s(X,I), s(X+1,J)
    .
```

1. set p_i for all text positions $i \in [1..n]$
2. select the number of factors z in $[1..n]$
3. the first factor starts at position 1: $s_1 = 1$
4. for convenience, define $(z+1)$-st factor at position $s_{z+1} = n + 1$

to continue $\cdots$

SLP ASP encoding 1/2

code:

```
p(I) :- lpnf(I,_).
1 { z(X) : X = 1..n } 1.
s(1,1).
s(Z+1,n+1) :- z(Z).
1 { s(X+1,I) : p(I), I > J, s(X,J) }
    1 :- p(X), X < Z, z(Z).
l(X,L) :- L = J - I, s(X,I), s(X+1,J)
    .
```

1. set p_i for all text positions $i \in [1..n]$
2. select the number of factors z in $[1..n]$
3. the first factor starts at position 1: $s_1 = 1$
4. for convenience, define $(z+1)$-st factor at position $s_{z+1} = n+1$
5. for each $x \in [1..z]$, select $s_x \in [s_{x-1}..n]$

$[z, n]$

to continue $\cdots$

SLP ASP encoding 1/2

code:

```
p(I) :- lpnf(I,_).
1 { z(X) : X = 1..n } 1.
s(1,1).
s(Z+1,n+1) :- z(Z).
1 { s(X+1,I) : p(I), I > J, s(X,J) }
     1 :- p(X), X < Z, z(Z).
l(X,L) :- L = J - I, s(X,I), s(X+1,J)
     .
```

1. set p_i for all text positions $i \in [1..n]$
2. select the number of factors z in $[1..n]$
3. the first factor starts at position 1: $s_1 = 1$
4. for convenience, define $(z+1)$-st factor at position $s_{z+1} = n + 1$
5. for each $x \in [1..z]$, select $s_x \in [s_{x-1}..n]$

   $[z, n]$

6. set $\ell_x := s_{x+1} - s_x$

to continue $\cdots$

SLP ASP encoding 2/2

code:

```
1 { r(X,Y) : Y = 1..X-1, s(Y,J), lce(J,I,LCE), LCE >= L } 1 :- s(X,I), l(X,L), L > 1.
1 { span(X,S) : S = 2..Z-Y+1 } 1 :- s(X,_), r(X,Y), l(X,L), L > 1, z(Z).
:- r(X,Y), span(X,S), Y+S > X.
:- r(X,Y), s(Y,J), l(X,L), span(X,S), not s(Y+S,J+L).
:- r(X,U), r(Y,V), span(X,L), span(Y,K), U < V, V < U+L, U+L < V+K.
:- lpnf(I,V), l(X,L), s(X,I), L > 1, not V >= L.
#minimize { Z : z(Z) }.
```

SLP ASP encoding 2/2

code:

```
1 { r(X,Y) : Y = 1..X-1, s(Y,J), lce(J,I,LCE), LCE >= L } 1 :- s(X,I), l(X,L), L > 1.
1 { span(X,S) : S = 2..Z-Y+1 } 1 :- s(X,_), r(X,Y), l(X,L), L > 1, z(Z).
:- r(X,Y), span(X,S), Y+S > X.
:- r(X,Y), s(Y,J), l(X,L), span(X,S), not s(Y+S,J+L).
:- r(X,U), r(Y,V), span(X,L), span(Y,K), U < V, V < U+L, U+L < V+K.
:- lpnf(I,V), l(X,L), s(X,I), L > 1, not V >= L.
#minimize { Z : z(Z) }.
```

1. variable $r_{x,y}$ denotes that F_x refers to F_y. This is only possible if $\mathrm{lce}(s_y, s_x) \geq \ell_x$.

$$\forall x \in [1..z], y \in [1..x-1] : \mathrm{lce}(s_x, s_y) \geq \ell_x \Leftarrow r_{x,y}$$

Defines $\mathcal{O}(z^2)$ variables.

SLP ASP encoding 2/2

code:

```
1 { r(X,Y) : Y = 1..X-1, s(Y,J), lce(J,I,LCE), LCE >= L } 1 :- s(X,I), l(X,L), L > 1.
1 { span(X,S) : S = 2..Z-Y+1 } 1 :- s(X,_), r(X,Y), l(X,L), L > 1, z(Z).
:- r(X,Y), span(X,S), Y+S > X.
:- r(X,Y), s(Y,J), l(X,L), span(X,S), not s(Y+S,J+L).
:- r(X,U), r(Y,V), span(X,L), span(Y,K), U < V, V < U+L, U+L < V+K.
:- lpnf(I,V), l(X,L), s(X,I), L > 1, not V >= L.
#minimize { Z : z(Z) }.
```

2. select span_x for each referencing factor, which must be at least two by definition $\ell_x \geq 2 \wedge r_{x,y} \Rightarrow \mathrm{span}_x \in [2..z - y + 1]$

   Defines $\mathcal{O}(z^2)$ variables.

SLP ASP encoding 2/2

code:

```
1 1 { r(X,Y) : Y = 1..X-1, s(Y,J), lce(
      J,I,LCE), LCE >= L } 1 :- s(X,I),
       l(X,L), L > 1.
2 1 { span(X,S) : S = 2..Z-Y+1 } 1 :- s
      (X,_), r(X,Y), l(X,L), L > 1, z(Z
      ).
3 :- r(X,Y), span(X,S), Y+S > X.
4 :- r(X,Y), s(Y,J), l(X,L), span(X,S),
       not s(Y+S,J+L).
5 :- r(X,U), r(Y,V), span(X,L), span(Y,
      K), U < V, V < U+L, U+L < V+K.
6 :- lpnf(I,V), l(X,L), s(X,I), L > 1,
      not V >= L.
7 #minimize { Z : z(Z) }.
```

3. The span span_x cannot extend into F_x:

$$\forall x, y \in [1..z], y < x :$$
$$r_{x,y} \Rightarrow \mathrm{span}_x + y \leq x$$

$[\mathcal{O}(z^2), \mathcal{O}(z)?]$

SLP ASP encoding 2/2

code:

```
1 { r(X,Y) : Y = 1..X-1, s(Y,J), lce(J,I,LCE), LCE >= L } 1 :- s(X,I), l(X,L), L > 1.
1 { span(X,S) : S = 2..Z-Y+1 } 1 :- s(X,_), r(X,Y), l(X,L), L > 1, z(Z).
:- r(X,Y), span(X,S), Y+S > X.
:- r(X,Y), s(Y,J), l(X,L), span(X,S), not s(Y+S,J+L).
:- r(X,U), r(Y,V), span(X,L), span(Y,K), U < V, V < U+L, U+L < V+K.
:- lpnf(I,V), l(X,L), s(X,I), L > 1, not V >= L.
#minimize { Z : z(Z) }.
```

4. If F_x refers to F_y, then ℓ_x and span_x determine the start of $F_{y+\mathrm{span}_x}$:

$$\forall x, y \in [1..z], y < x :$$

$$r_{x,y} \Rightarrow s_{y+\mathrm{span}_x} = s_y + \ell_x$$

$[\mathcal{O}(z^2), \mathcal{O}(z^3)?]$

SLP ASP encoding 2/2

code:

```
1 1 { r(X,Y) : Y = 1..X-1, s(Y,J), lce(J,I,LCE), LCE >= L } 1 :- s(X,I), l(X,L), L > 1.
2 1 { span(X,S) : S = 2..Z-Y+1 } 1 :- s(X,_), r(X,Y), l(X,L), L > 1, z(Z).
3 :- r(X,Y), span(X,S), Y+S > X.
4 :- r(X,Y), s(Y,J), l(X,L), span(X,S), not s(Y+S,J+L).
5 :- r(X,U), r(Y,V), span(X,L), span(Y,K), U < V, V < U+L, U+L < V+K.
6 :- lpnf(I,V), l(X,L), s(X,I), L > 1, not V >= L.
7 #minimize { Z : z(Z) }.
```

5. reference intervals are either disjoint or contained

$$\forall u, x, v, y \in [1..z], u < x, v < y, u < v : r_{x,u} \wedge r_{y,v} \Rightarrow \neg(v < u + \mathrm{span}_x \wedge u + \mathrm{span}_x < v + \mathrm{span}_y)$$

$[\mathcal{O}(z^4), \mathcal{O}(z^2)?]$

SLP ASP encoding 2/2

code:

```
1 1 { r(X,Y) : Y = 1..X-1, s(Y,J), lce(J,I,LCE), LCE >= L } 1 :- s(X,I), l(X,L), L > 1.
2 1 { span(X,S) : S = 2..Z-Y+1 } 1 :- s(X,_), r(X,Y), l(X,L), L > 1, z(Z).
3 :- r(X,Y), span(X,S), Y+S > X.
4 :- r(X,Y), s(Y,J), l(X,L), span(X,S), not s(Y+S,J+L).
5 :- r(X,U), r(Y,V), span(X,L), span(Y,K), U < V, V < U+L, U+L < V+K.
6 :- lpnf(I,V), l(X,L), s(X,I), L > 1, not V >= L.
7 #minimize { Z : z(Z) }.
```

6. optimization: can only select a length as long as the LPNF value, i.e., restrict $\ell_x \in \{1\} \cup [0..\mathrm{lpnf}(s_x)]$

$$\forall x \in [1..z] :$$
$$\ell_x > 1 \Rightarrow \ell_x \in [1..\mathrm{lpnf}(s_x)]$$

$[\mathcal{O}(z), \mathcal{O}(n)?]$

SLP ASP encoding 2/2

code:

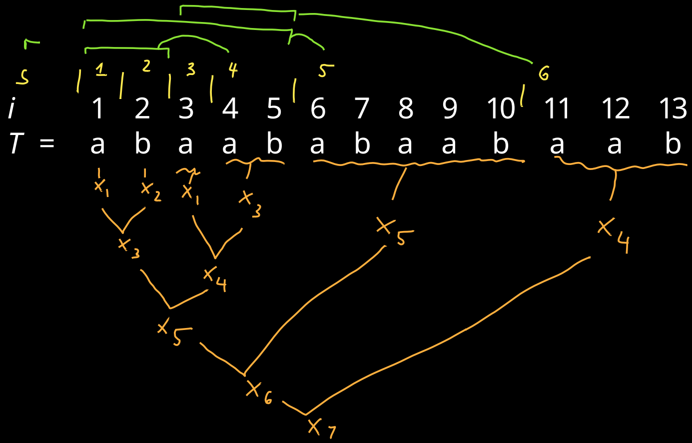
```
1 1 { r(X,Y) : Y = 1..X-1, s(Y,J), lce(J,I,LCE), LCE >= L } 1 :- s(X,I), l(X,L), L > 1.
2 1 { span(X,S) : S = 2..Z-Y+1 } 1 :- s(X,_), r(X,Y), l(X,L), L > 1, z(Z).
3 :- r(X,Y), span(X,S), Y+S > X.
4 :- r(X,Y), s(Y,J), l(X,L), span(X,S), not s(Y+S,J+L).
5 :- r(X,U), r(Y,V), span(X,L), span(Y,K), U < V, V < U+L, U+L < V+K.
6 :- lpnf(I,V), l(X,L), s(X,I), L > 1, not V >= L.
7 #minimize { Z : z(Z) }.
```

7. objective:
   minimize number of factors z

例

clingo の出力:

```
s(1,1)
s(2,2)
s(3,3)
s(4,4)
s(5,6)
s(6,11)
s(7,14)
r(4,1)
r(5,1)
r(6,3)
span(4,2)
span(5,4)
span(6,2)
```



変数

- $z \in [1..n]$: 項の個数
- $\forall x \in [1..z] : s_x \in [s_{x-1} + 1..s_{x+1} - 1] \subset [1..n]$: 項の開始位置
- $\forall x, y \in [1..z], x < y : r_{x,y} \in [1..z]$: x の参照先
- $\forall x \in [1..z] : \mathrm{span}_x \in [1..x - 1]$: 参照された項の個数