A bottom-up efficient algorithm learning substitutable languages from positive examples

François Coste, Gaelle Garet, Jacques Nicolas

ICGI, Kyoto, September 17, 2014

Motivation

Distributional Hypothesis (words that occur in the same contexts tend to have similar meanings [Harris, 1954]. "a word is characterized by the company it keeps" [Firth, 1957]) has been for long an influential idea in Linguistics :

- Part of the language acquisition discussion...
- Base of Statistical Semantics
- Unsupervised POS parsing (Constituent-Context Models [Klein & Manning, 2001]...)
- Learning expressive grammars from positive examples only
  - Heuristics : EMILE [Adriaans, 1992 ; Adriaans and Vervoort, 2002)] , ABL [van Zaanen, 2002], ADIOS [Solan et al., 2005]...
  - Characterizable inference of *substitutable languages* : [Clark & Eyraud 2007, Yoshinaka 2008, ...]

and [CGN2012] for proteins !

Substitutable Languages

L is *substitutable* [CLARK & EYRAUD, 2007] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, y_1, y_2 \neq \lambda$:

$$x_1 y_1 z_1 \in L \wedge x_1 y_2 z_1 \in L \wedge x_2 y_1 z_2 \in L \Rightarrow x_2 y_2 z_2 \in L$$

Substitutable Languages

L is *substitutable* [Clark & Eyraud, 2007] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, y_1, y_2 \neq \lambda$:

$$x_1 y_1 z_1 \in L \wedge x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L)$$

Substitutable Languages

L is *substitutable* [Clark & Eyraud, 2007] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, y_1, y_2 \neq \lambda$:

$$x_1 y_1 z_1 \in L \wedge x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L)$$

i.e. $[y_1] = [y_2]$

Substitutable Languages

L is *substitutable* [Clark & Eyraud, 2007] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, y_1, y_2 \neq \lambda$:

$$x_1 y_1 z_1 \in L \wedge x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L)$$

i.e. $[y_1] = [y_2]$

L is *k, l-substitutable* [Yoshinaka, 2008] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_1 u y_2 v z_1 \in L \Rightarrow (x_2 u y_1 v z_2 \in L \Leftrightarrow x_2 u y_2 v z_2 \in L)$$

Substitutable Languages

L is *substitutable* [Clark & Eyraud, 2007] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, y_1, y_2 \neq \lambda$:

$$x_1 y_1 z_1 \in L \wedge x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L)$$

i.e. $[y_1] = [y_2]$

L is *k, l-local substitutable* [CGN, 2012] if :

$x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_3 u y_2 v z_3 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L)$$

L is *k, l-substitutable* [Yoshinaka, 2008] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_1 u y_2 v z_1 \in L \Rightarrow (x_2 u y_1 v z_2 \in L \Leftrightarrow x_2 u y_2 v z_2 \in L)$$

Substitutable Languages

L is *substitutable* [Clark & Eyraud, 2007] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, y_1, y_2 \neq \lambda$:

$$x_1 y_1 z_1 \in L \wedge x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L)$$

i.e. $[y_1] = [y_2]$

L is k, l*-local substitutable* [CGN, 2012] if :

$x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_3 u y_2 v z_3 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L)$$

L is k, l*-substitutable* [Yoshinaka, 2008] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_1 u y_2 v z_1 \in L \Rightarrow (x_2 u y_1 v z_2 \in L \Leftrightarrow x_2 u y_2 v z_2 \in L)$$

L is k, l*-local-context substitutable* [CGN, 2012] if :

$x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_3 u y_2 v z_3 \in L \Rightarrow (x_2 u y_1 v z_2 \in L \Leftrightarrow x_2 u y_2 v z_2 \in L)$$

Substitutable Languages

L is *substitutable* [Clark & Eyraud, 2007] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, y_1, y_2 \neq \lambda$:

$$x_1 y_1 z_1 \in L \wedge x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L)$$

i.e. $[y_1] = [y_2]$

L is k, l*-local substitutable* [CGN, 2012] if :

$x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_3 u y_2 v z_3 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L)$$

L is k, l*-context-substitutable* [Yoshinaka, 2008] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_1 u y_2 v z_1 \in L \Rightarrow (x_2 u y_1 v z_2 \in L \Leftrightarrow x_2 u y_2 v z_2 \in L)$$

L is k, l*-local-context substitutable* [CGN, 2012] if :

$x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_3 u y_2 v z_3 \in L \Rightarrow (x_2 u y_1 v z_2 \in L \Leftrightarrow x_2 u y_2 v z_2 \in L)$$

Substitutable Languages

L is *zero-substitutable* [Clark & Eyraud, 2007] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, y_1, y_2 \neq \lambda$:

$$x_1 y_1 z_1 \in L \wedge x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L)$$

i.e. $[y_1] = [y_2]$

L is *k, l-local substitutable* [CGN, 2012] if :

$x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_3 u y_2 v z_3 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L)$$

L is *k, l-context-substitutable* [Yoshinaka, 2008] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_1 u y_2 v z_1 \in L \Rightarrow (x_2 u y_1 v z_2 \in L \Leftrightarrow x_2 u y_2 v z_2 \in L)$$

L is *k, l-local-context substitutable* [CGN, 2012] if :

$x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_3 u y_2 v z_3 \in L \Rightarrow (x_2 u y_1 v z_2 \in L \Leftrightarrow x_2 u y_2 v z_2 \in L)$$

Substitutable Languages

L is *zero-substitutable* [Clark & Eyraud, 2007] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, y_1, y_2 \neq \lambda$:

$$x_1 y_1 z_1 \in L \wedge x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L)$$

i.e. $[y_1] = [y_2]$

L is *k, l-local substitutable* [CGN, 2012] if :

$x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_3 u y_2 v z_3 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L)$$

i.e. $[y_1] = [y_2]$

L is *k, l-context-substitutable* [Yoshinaka, 2008] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_1 u y_2 v z_1 \in L \Rightarrow (x_2 u y_1 v z_2 \in L \Leftrightarrow x_2 u y_2 v z_2 \in L)$$

L is *k, l-local-context substitutable* [CGN, 2012] if :

$x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_3 u y_2 v z_3 \in L \Rightarrow (x_2 u y_1 v z_2 \in L \Leftrightarrow x_2 u y_2 v z_2 \in L)$$

Substitutable Languages

L is *zero-substitutable* [Clark & Eyraud, 2007] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, y_1, y_2 \neq \lambda$:

$$x_1 y_1 z_1 \in L \wedge x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L)$$

i.e. $[y_1] = [y_2]$

L is *k, l-local substitutable* [CGN, 2012] if :

$x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_3 u y_2 v z_3 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L)$$

i.e. $[y_1] = [y_2]$

L is *k, l-context-substitutable* [Yoshinaka, 2008] if :

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_1 u y_2 v z_1 \in L \Rightarrow (x_2 u y_1 v z_2 \in L \Leftrightarrow x_2 u y_2 v z_2 \in L)$$

i.e. $[u y_1 v] = [u y_2 v]$

L is *k, l-local-context substitutable* [CGN, 2012] if :

$x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$

$$x_1 u y_1 v z_1 \in L \wedge x_3 u y_2 v z_3 \in L \Rightarrow (x_2 u y_1 v z_2 \in L \Leftrightarrow x_2 u y_2 v z_2 \in L)$$

i.e. $[u y_1 v] = [u y_2 v]$

“Weak-implies-Strong” Generalization

Let K be the following set of strings :

Major General was here yesterday morning.
Major General went here yesterday morning.
Major General will be there tomorrow morning.
He will be gone tomorrow evening.

Strings to add to get a $1,1$-local substitutable language :

"Weak-implies-Strong" Generalization

Let K be the following set of strings :

Major General was here yesterday morning.
Major General went here yesterday morning.
Major General will be there tomorrow morning.
He will be gone tomorrow evening.

Strings to add to get a $1,1$-local substitutable language :

Major General will be gone tomorrow morning.
He will be there tomorrow evening.

"Weak-implies-Strong" Generalization

Let K be the following set of strings :

Major General was here yesterday morning.
Major General went here yesterday morning.
Major General will be there tomorrow morning.
He will be gone tomorrow evening.

Strings to add to get a $1,1$-local substitutable language :

Major General will be gone tomorrow morning.
He will be there tomorrow evening.

He will be gone tomorrow morning.
Major General will be there tomorrow evening.
Major General was here yesterday evening.
Major General went here yesterday evening.

“Weak-implies-Strong” Generalization

Let K be the following set of strings :

Major General was here yesterday morning.
Major General went here yesterday morning.
Major General will be there tomorrow morning.
He will be gone tomorrow evening.

Strings to add to get a $1,1$-local substitutable language :

Major General will be gone tomorrow morning.
He will be there tomorrow evening.

He will be gone tomorrow morning.
Major General will be there tomorrow evening.
Major General was here yesterday evening.
Major General went here yesterday evening.

Major General will be gone tomorrow evening.
He will be there tomorrow morning.

⋮

Learning k, l-local substitutable languages

Adaptation of SGL algorithm [Clark & Eyraud, 2007] :

SGL_{LS} (Substitution Graph Learner for Local Substitutable languages)

Input: Set of sequences K on alphabet Σ, int k, int l
Output: Grammar $G = \langle \Sigma, N_K, S_K, P_K \rangle$

```
/* Partition Sub(K) in Local Substitutability classes */
C_K ← LS_classes(K, k, l) /* ∀y ∈ Sub(K), C_K(y) = C ∈ C_K : y ∈ C */
/* Build grammar */
N_K ← ∅, P_K ← ∅
for C ∈ C_K do
    /* A non-terminal for each substitutability class */
    N_K ← N_K ∪ {⟦C⟧}
    /* Productions rules for each substring in the class */
    for y ∈ C do
        if |y| > 1 then
            /* Branching rules: a 'CNF' rule for each split */
            for y_1 ∈ Σ^+, y_2 ∈ Σ^+ : y_1 y_2 = y do
                P_K ← P_K ∪ {⟦C⟧ → ⟦C_K(y_1)⟧⟦C_K(y_2)⟧}
        else
            /* Terminal rule */
            P_K ← P_K ∪ {⟦C⟧ → y}
S_K ← ⟦C_K(k)⟧ : k ∈ K
return ⟨Σ, N_K, S_K, P_K⟩
```

Local substitutability classes

LS_classes()

Input: Set of sequences K on alphabet Σ, int k, int l
Output: k, l local substitutability classes on K

```
/* Build substitutability graph on substrings */
```

$V \leftarrow \{y \in \Sigma^+ : y \in Sub(K)\}$

$E \leftarrow \{\{y_1, y_2\} \in V \times V :$
$\quad uy_1v \in Sub(K), uy_2v \in Sub(K), y_1 \neq y_2, u \in \Sigma^k, v \in \Sigma^l\}$

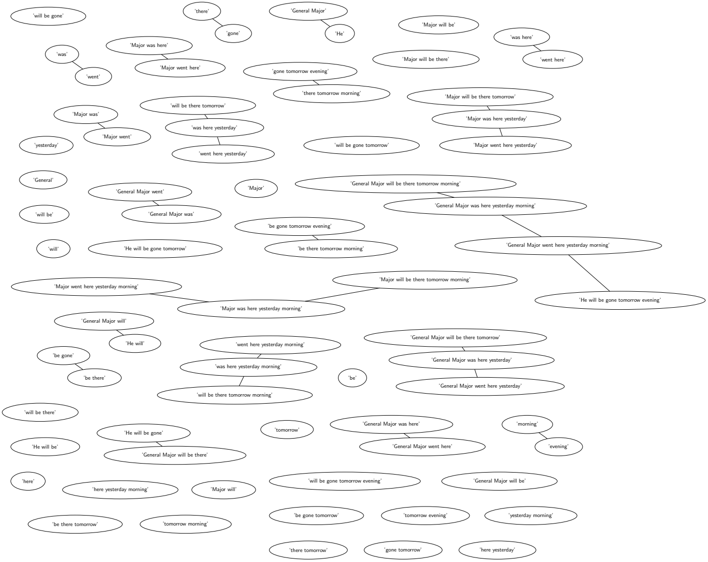
```
/* Return connected components of graph */
```

return `Connected_components`$(\langle V, E \rangle)$

where $Sub(K)$ denotes the set of substrings of K

- This is the unique difference with SGL!
- To learn other substitutable classes, change E and V initialization.

Substitutability graph



source: Gaelle Garet

```
SGL_LS          (Substitution Graph Learner for Local Substitutable languages)

Input: Set of sequences K on alphabet Σ, int k, int l
Output: Grammar G = ⟨Σ, N_K, S_K, P_K⟩
/* Partition Sub(K) in Local Substitutability classes */
𝒞_K ← LS_classes(K, k, l) /* ∀y ∈ Sub(K), C_K(y) = C ∈ 𝒞_K : y ∈ C */
/* Build grammar */
N_K ← ∅, P_K ← ∅
for C ∈ 𝒞_K do
    /* A non-terminal for each substitutability class */
    N_K ← N_K ∪ {⟦C⟧}
    /* Productions rules for each substring in the class */
    for y ∈ C do
        if |y| > 1 then
            /* Branching rules: a 'CNF' rule for each split */
            for y_1 ∈ Σ^+, y_2 ∈ Σ^+ : y_1 y_2 = y do
                P_K ← P_K ∪ {⟦C⟧ → ⟦C_K(y_1)⟧⟦C_K(y_2)⟧}
        else
            /* Terminal rule */
            P_K ← P_K ∪ {⟦C⟧ → y}
S_K ← ⟦C_K(k)⟧ : k ∈ K
return ⟨Σ, N_K, S_K, P_K⟩
```

Resulting grammar :

$X_{47} \rightarrow '\ yesterday'$
$X_{46} \rightarrow X_3X_{43}|X_{11}X_{19}|X_{23}X_{13}$
$X_{45} \rightarrow X_{20}X_2$
$X_{44} \rightarrow X_{20}X_1|X_{45}X_{29}|X_{34}X_{16}|X_9X_{15}$
$X_{43} \rightarrow X_{20}X_{19}|X_{45}X_{13}$
$X_{42} \rightarrow '\ tomorrow'$
$X_{41} \rightarrow X_{42}X_{15}$
$X_{40} \rightarrow X_{30}X_{46}|X_{21}X_{43}|X_{39}X_{19}|X_{25}X_{13}|X_{21}X_{34}|X_8X_{13}$
$N_0 \rightarrow X_{30}X_{24}|X_{21}X_{31}|X_{10}X_{32}|X_{36}X_{17}|X_{26}X_{15}|X_{39}X_1$
$|X_{25}X_{29}|X_{40}X_{41}|X_{21}X_{44}|X_8X_{29}|X_{40}X_{16}|X_{33}X_{15}$
$X_{29} \rightarrow X_{13}X_{16}|X_4X_{15}|X_{13}X_{41}|X_{38}X_{15}$
$X_{28} \rightarrow X_{27}X_{47}$
$X_{25} \rightarrow X_{30}X_{23}|X_{21}X_{45}|X_{39}X_2$
$X_{24} \rightarrow X_3X_{31}|X_{18}X_{32}|X_{37}X_{17}|X_{14}X_{15}|X_{11}X_1|X_{23}X_{29}|X_{46}X_{41}$
$X_{27} \rightarrow '\ here'$
$X_{26} \rightarrow X_{30}X_{14}|X_{21}X_{12}|X_{10}X_{28}|X_{36}X_{47}|X_{39}X_{35}|X_{25}X_{38}|X_{40}X_{42}$
$X_{21} \rightarrow '\ He'|X_{30}X_3$
$X_{20} \rightarrow '\ will'$
$X_{23} \rightarrow X_3X_{45}|X_{11}X_2$
$X_{22} \rightarrow X_6X_{27}$
$X_8 \rightarrow X_{21}X_{45}|X_{39}X_2$
$X_9 \rightarrow X_{20}X_5|X_{45}X_4|X_{34}X_{42}$
$X_2 \rightarrow '\ be'$
$X_3 \rightarrow '\ Major'$

$X_1 \rightarrow X_2X_{29}|X_{19}X_{16}|X_5X_{15}|X_{19}X_{41}|X_{35}X_{15}$
$X_6 \rightarrow '\ was'|'\ went'$
$X_4 \rightarrow X_{13}X_{42}$
$X_5 \rightarrow X_2X_4|X_{19}X_{42}$
$X_{32} \rightarrow X_{27}X_{17}|X_{28}X_{15}$
$X_{33} \rightarrow X_{21}X_9|X_{39}X_5|X_8X_4|X_{40}X_{42}$
$X_{30} \rightarrow General$
$X_{31} \rightarrow X_6X_{32}|X_{22}X_{17}|X_{12}X_{15}|X_{20}X_1|X_{45}X_{29}|X_{43}X_{41}$
$X_{36} \rightarrow X_{30}X_{37}|X_{21}X_{22}|X_{10}X_{27}$
$X_{37} \rightarrow X_3X_{22}|X_{18}X_{27}$
$X_{34} \rightarrow X_{20}X_{19}|X_{45}X_{13}$
$X_{35} \rightarrow X_2X_{38}|X_{19}X_{42}$
$X_{38} \rightarrow X_{13}X_{42}$
$X_{39} \rightarrow X_{30}X_{11}|X_{21}X_{20}$
$X_{18} \rightarrow X_3X_6$
$X_{19} \rightarrow X_2X_{13}$
$X_{10} \rightarrow X_{30}X_{18}|X_{21}X_6$
$X_{11} \rightarrow X_3X_{20}$
$X_{12} \rightarrow X_6X_{28}|X_{22}X_{47}|X_{20}X_{35}|X_{45}X_{38}|X_{43}X_{42}$
$X_{13} \rightarrow '\ there'|'\ gone'$
$X_{14} \rightarrow X_3X_{12}|X_{18}X_{28}|X_{37}X_{47}|X_{11}X_{35}|X_{23}X_{38}|X_{46}X_{42}$
$X_{15} \rightarrow '\ morning'|'\ evening'$
$X_{16} \rightarrow X_{42}X_{15}$
$X_{17} \rightarrow X_{47}X_{15}$

source : GaelleGaret

Limitations

- A lot of non-terminals and rules
- A lot of redundancy and ambiguity

$\Rightarrow$ Parsing time and learning time problems[1] (+ Illegible)

1. About a day for one experiment on a simple set of proteins

Limitations

- A lot of non-terminals and rules
- A lot of redundancy and ambiguity

$\Rightarrow$ Parsing time and learning time problems [1] (+ Illegible)

A solution

Reduce the grammar before parsing

1. About a day for one experiment on a simple set of proteins

Limitations

- A lot of non-terminals and rules
- A lot of redundancy and ambiguity

$\Rightarrow$ Parsing time and learning time problems [1] (+ Illegible)

A solution

Reduce the grammar during the inference

1. About a day for one experiment on a simple set of proteins

Ideas

1 Avoid unnecessary derivations by removing non-terminals with a deterministic derivation :
If A is the left-hand-side of only one rule $A \rightarrow \alpha$ then delete them and replace all $B \rightarrow \ldots A \ldots$ by $B \rightarrow \ldots \alpha \ldots$.

2 Reduce the right-hand-sides of the production rules :
For each $N \rightarrow \ldots \beta \ldots$, if there exists $\alpha : |\alpha| < |\beta| \wedge \alpha \Rightarrow \beta$, then replace $N \rightarrow \ldots \beta \ldots$ by $N \rightarrow \ldots \alpha \ldots$

(Teaser : it ensures also maximal generalization !)

1. Avoid unnecessary derivations

Keep prime classes

Recall : Each non-terminal corresponds to a substitutability congruence class.
Slight abuse of notation : Let $[x]$ denote the class of a non-terminal or terminal x.
N is deterministically derived by $N \to \alpha = \alpha_1 \dots \alpha_{|\alpha|}$ implies

$$[N] = [\alpha_1]...[\alpha_{|\alpha|}].$$

We name such useless class a *composite* class (They are those giving rise to *vacuous local derivation trees* [Clark, 2011]).
We say that a class is *prime* if it is not composite. We have :

Primality test

Let a language L whose set of non-zero and non-unit congruence classes is C^+.
A class $[y]$ in C^+ is *prime* for L iff $\forall y_1 y_2 \in [y], [y] \not\subset [y_1][y_2]$.

Sufficient test since for syntactic congruence, we have $[y_1 y_2] \supseteq [y_1][y_2]$

K-Primes

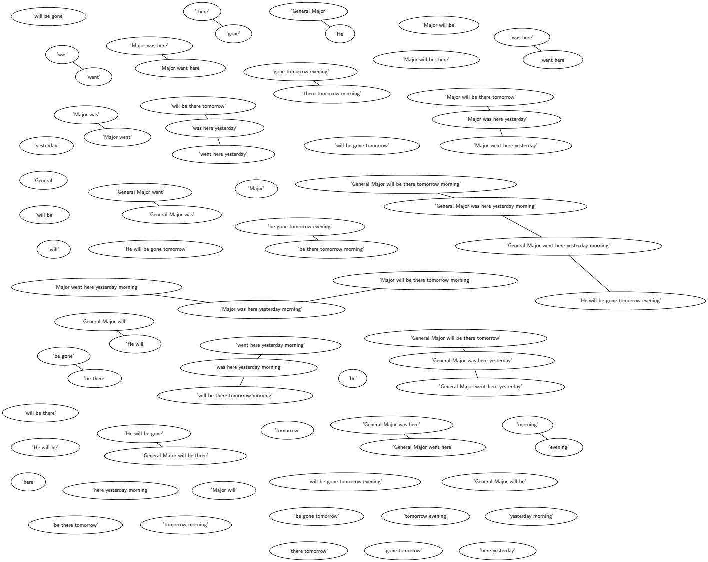
Due to monotonicity of generalization, it is safe to filter out composite classes on the basis of the sample K.
We introduce the function Prime() for that purpose :

Primes()

Input: Set of substitutability classes : $\mathcal{C}_K$
Output: Set of substitutability classes satisfying primality test in $\mathcal{C}_K$: $\mathcal{P}$
$\mathcal{P} \leftarrow \emptyset$
for $C \in \mathcal{C}_K$ **do**
  If $(\nexists C' \in \mathcal{C}_K : \forall y \in C, \exists y' \in C', \exists v \in \Sigma^+, y = y'v)$
  and $(\nexists C' \in \mathcal{C}_K : \forall y \in C, \exists y' \in C', \exists u \in \Sigma^+, y = uy')$
  $\mathcal{P} \leftarrow \mathcal{P} \cup C$

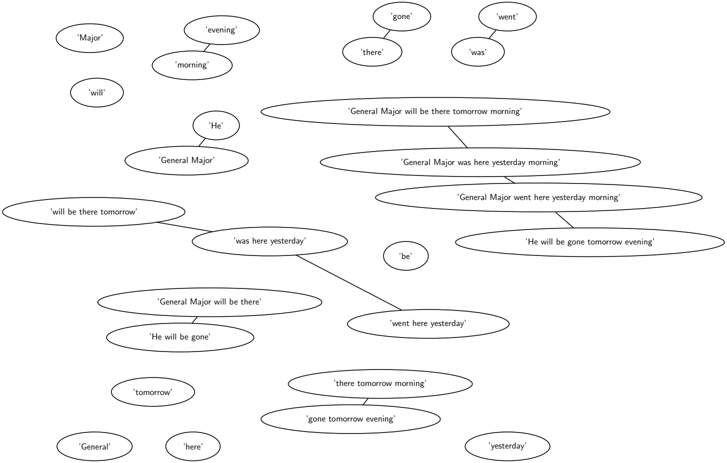
Primality test in K not in L !
but works well in practice and if the sample is informative enough.

On the example

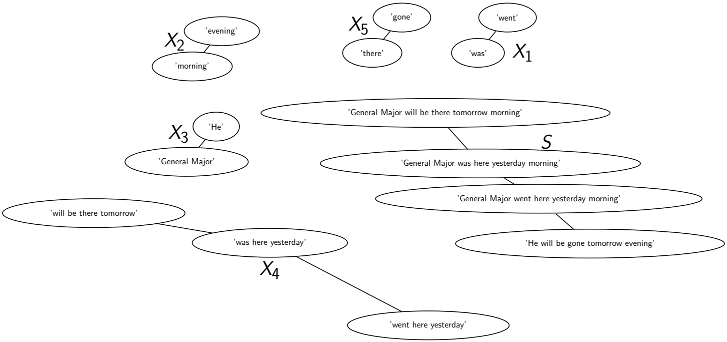


source: Gaelle Garet

On the example



On the example



ReGLiS Part1 (Learning **R**educed **G**rammar by k, l-**L**ocal **S**ubstitutability)
simplified !

Input: Set of sequences K on alphabet Σ, int k, int l
Output: Grammar $G = \langle \Sigma, N_K, S_K, P_K \rangle$

```
/* Prime substitutability classes on K */
```
$\mathcal{C}_K \leftarrow \texttt{Primes}(\texttt{LS\_classes}(K, k, l))$

```
/* Build initial grammar */
```
$N_K \leftarrow \emptyset, P_K \leftarrow \emptyset$

for $C \in \mathcal{C}_K$ **do**

```
    /* A non-terminal for each K-prime */
```
$\quad N_K \leftarrow N_K \cup \{[\![C]\!]\}$

```
    /* A direct production for each substring in the class */
```
$\quad$ **for** $y \in C$ **do**

$\quad\quad P_K \leftarrow P_K \cup \{[\![C]\!] \rightarrow y\}$

```
/* So far, we have built the initial grammar */
```

ReGLiS Part1 (Learning **R**educed **G**rammar by k, l-**L**ocal **S**ubstitutability) simplified !

Input: Set of sequences K on alphabet Σ, int k, int l
Output: Grammar $G = \langle \Sigma, N_K, S_K, P_K \rangle$

```
/* Prime substitutability classes on K */
C_K ← Primes(LS_classes(K, k, l))
/* Build initial grammar */
N_K ← ∅, P_K ← ∅
for C ∈ C_K do
    /* A non-terminal for each K-prime */
    N_K ← N_K ∪ {⟦C⟧}
    /* A direct production for each substring in the class */
    for y ∈ C do
        P_K ← P_K ∪ {⟦C⟧ → y}
/* So far, we have built the initial grammar */
```

Initial 'bottom' grammar

$G =< \Sigma, V_k, P, S >$
$\Sigma = \{$ General, Major, will, be, gone, there, was, went, He, here, tomorrow, yesterday, morning, evening $\}$
$V_k = \{ S, X_1, X_2, X_3, X_4, X_5 \}$
$P = \{$
S $\rightarrow$ General Major will be there tomorrow morning | General Major was here yesterday morning | General Major went here yesterday morning | He will be gone tomorrow morning
$X_1 \rightarrow$ was | went
$X_2 \rightarrow$ morning | evening
$X_3 \rightarrow$ He | General Major
$X_4 \rightarrow$ will be there tomorrow | was here yesterday | went here yesterday
$X_5 \rightarrow$ there | gone
$\}$

- A non-terminal for each K-Prime
- $L(G) = K$ (NO language generalization)
- for each non-terminal N, $L(G, N) = [N]$

2. Reduce right-hand-sides

And generalize at once

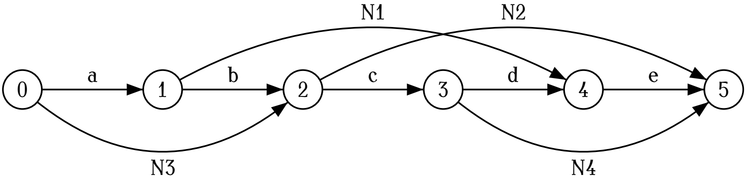
- We 'know' the interesting substitutability classes (but only part of their content)
- If a substring from a substitutability class is present in one right-hand side, we have to replace this occurrence by the non-terminal of the class (Weak-implies-Strong generalization)

$$\frac{N_1 \rightarrow \ldots \beta \ldots,\ N_2 \Rightarrow^* \beta}{N_1 \rightarrow \ldots N_2 \ldots,\ N_2 \Rightarrow^* \beta}$$

- Take care of overlapping occurrences
- Don't keep/introduce redundant rules, introduce most general only
- Some kinship with *Minimal Grammar Parsing* for smallest grammar problem [Carrascosa et al 2011, Gallé, 2011], but with more than one substring per non-terminal

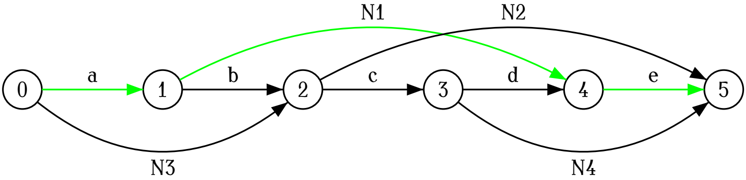
Example

Let us consider that so far the grammar is such that
$P = \{ \ldots ; N \rightarrow abcde \mid \ldots ; N_1 \rightarrow bcd \mid \ldots ; N_2 \rightarrow cde \mid \ldots ; N_3 \rightarrow ab \mid \ldots ; N_4 \rightarrow de \mid \ldots ; \ldots \}$ (where a, b, c, d, e is a terminal or non-terminal symbols)
The parsing graph for $N \rightarrow abcde$ is then :



Example

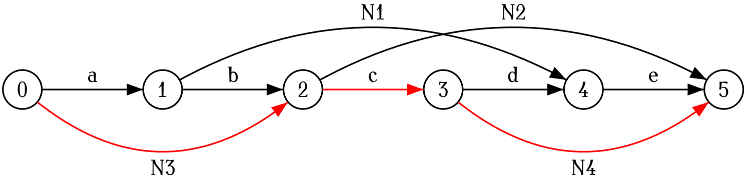
Let us consider that so far the grammar is such that
$P = \{ \ldots ; N \to abcde \mid \ldots ; N_1 \to bcd \mid \ldots ; N_2 \to cde \mid \ldots ; N_3 \to ab \mid \ldots ; N_4 \to de \mid \ldots ; \ldots \}$ (where a, b, c, d, e is a terminal or non-terminal symbols)
The parsing graph for $N \to abcde$ is then :



$N \to aN_1e$

Example

Let us consider that so far the grammar is such that
$P = \{ \ldots ; N \rightarrow abcde \mid \ldots ; N_1 \rightarrow bcd \mid \ldots ; N_2 \rightarrow cde \mid \ldots ; N_3 \rightarrow ab \mid \ldots ; N_4 \rightarrow de \mid \ldots ; \ldots \}$ (where a, b, c, d, e is a terminal or non-terminal symbols)
The parsing graph for $N \rightarrow abcde$ is then :



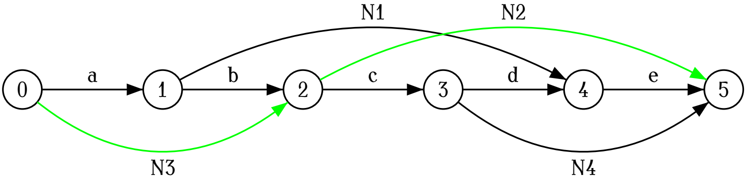
Can be reduced ! (025 strict subsequence[2] of 0235)

$N \rightarrow aN_1e$

2. typo in final version of paper ! replace p7 'strict substring' by 'strict subsequence'

Example

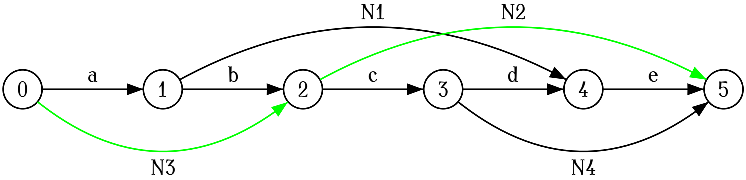
Let us consider that so far the grammar is such that
$P = \{ \ldots ; N \to abcde \mid \ldots ; N_1 \to bcd \mid \ldots ; N_2 \to cde \mid \ldots ; N_3 \to ab \mid \ldots ; N_4 \to de \mid \ldots ; \ldots \}$ (where a, b, c, d, e is a terminal or non-terminal symbols)
The parsing graph for $N \to abcde$ is then :



$N \to aN_1e | N_3N_2$

Example

Let us consider that so far the grammar is such that
$P = \{ \ldots ; N \rightarrow abcde \mid \ldots ; N_1 \rightarrow bcd \mid \ldots ; N_2 \rightarrow cde \mid \ldots ; N_3 \rightarrow ab \mid \ldots ; N_4 \rightarrow de \mid \ldots ; \ldots \}$ (where a, b, c, d, e is a terminal or non-terminal symbols)
The parsing graph for $N \rightarrow abcde$ is then :



$N \rightarrow aN_1e|N_3N_2$

Implemented by dynamic programming on vertices in function *Non_redundant_rhs*()

ReGLiS Part2

simplified !

```
/* Generalization */
repeat
    P ← P_K ; P_K ← ∅
    /* Branching rules */
    for ([[C]] → α) ∈ P ordered by increasing |α| do
        PG ← Build_parsing_graph(α, P)
        for β ∈ Non_redundant_rhs(PG) do
            P_K ← P_K ∪ ([[C]] → β)
until P_K = P
S_K ← [[C_K(k)]]: k ∈ K
return ⟨Σ, N_K, S_K, P_K⟩
```

Build_parsing_graph

Input: Sequence α, Set of rules P
Output: Parsing graph $\langle V, E \rangle$

```
V ← {i ∈ [0, |α|]} /* vertices */ ; E ← ∅ /* labeled directed edges */
for i ∈ V do
    for j ∈ V : i < j and (i, j) ≠ (0, |α|) do
        if ∃([[C]] → α[i + 1, j]) ∈ P then
            E ← E ∪ (i, j, [[C]])
return ⟨V, E⟩
```

ReGLiS Part2 simplified !

```
/* Generalization */
repeat
    P ← P_K ; P_K ← ∅
    /* Branching rules */
    for ([[C]] → α) ∈ P ordered by increasing |α| do
        PG ← Build_parsing_graph(α, P)
        for β ∈ Non_redundant_rhs(PG) do
            P_K ← P_K ∪ ([[C]] → β)
until P_K = P
S_K ← [[C_K(k)]] : k ∈ K
return ⟨Σ, N_K, S_K, P_K⟩
```

Build_parsing_graph

Input: Sequence α, Set of rules P
Output: Parsing graph $\langle V, E\rangle$

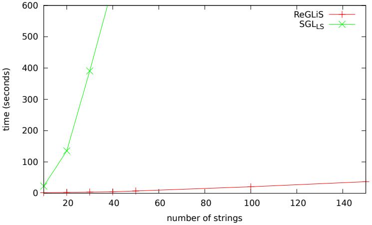
```
V ← {i ∈ [0, |α|]} /* vertices */ ; E ← ∅ /* labeled directed edges */
for i ∈ V do
    for j ∈ V : i < j and (i, j) ≠ (0, |α|) do
        if ∃([[C]] → α[i + 1, j]) ∈ P then
            E ← E ∪ (i, j, [[C]])
return ⟨V, E⟩
```

Final Grammar

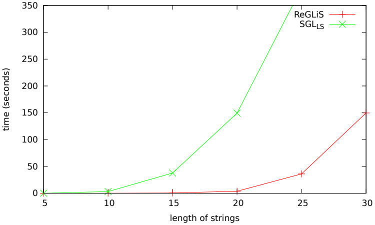
$$\mathbf{S} \rightarrow \mathbf{X_3}\ \mathbf{X_4}\ \mathbf{X_2}$$
$$\mathbf{X_1} \rightarrow \text{was} \mid \text{went}$$
$$\mathbf{X_2} \rightarrow \text{morning} \mid \text{evening}$$
$$\mathbf{X_3} \rightarrow \text{He} \mid \text{General Major}$$
$$\mathbf{X_4} \rightarrow \text{will be } \mathbf{X_5} \text{ tomorrow} \mid \mathbf{X_1} \text{ here yesterday}$$
$$\mathbf{X_5} \rightarrow \text{there} \mid \text{gone}$$

Complexity

- Complexity : $\mathcal{O}(\max(\mathbf{l^3}, \mathbf{l.t}))$
  - l : size of longest sequence
  - t : size of target grammar
- Run time comparison between old and new learning algorithms implementations :



wrt number of strings in training sample



wrt length of strings in training sample

- For our protein experiments : from a day to a few minutes

Protein experiments (10-fold cross-validation)

	Zinc finger			MPI phos.		
	Precision	Recall	F-measure	Precision	Recall	F-measure
Subst.	1	0.1	0.36	1	0.15	0.26
3,3-LS	1	0.2	0.33	1	0.5	0.67
4,4-LS	1	0.25	0.4	1	0.6	0.75
5,5-LS	1	0.33	0.5	**1**	**0.67**	**0.8**
6,6-LS	1	0.5	0.67	1	0.62	0.77
7,7-LS	**1**	**0.55**	**0.7**	1	0.53	0.69
SCFG	1	0.1	0.18	1	0.3	0.46
[Dyrka & Nebel,	0.15	1	0.26	0.5	1	0.67
2009]	0.75	0.87	0.85	0.98	0.89	0.93

	PS00219			PS00063		
	Precision	Recall	F-measure	Precision	Recall	F-measure
Subst.	1	0.2	0.33	1	0.23	0.37
3,3-LS	**1**	**0.72**	**0.84**	1	0.58	0.73
4,4-LS	1	0.7	0.82	1	0.6	0.75
5,5-LS	1	0.68	0.8	1	0.66	0.8
6,6-LS	1	0.6	0.75	**1**	**0.7**	**0.82**
7,7-LS	1	0.5	0.67	1	0.65	0.79
Prosite	1	0.6	0.75	1	0.8	0.89
SCFG	-	-	-	1	0.05	0.1
[Dyrka & Nebel,	-	-	-	0.1	1	0.18
2009]	1	1	1	0.79	0.65	0.71

Conclusion

- ReGLiS : (the ReG.*iS family : ReGiS, ReGCis, ReGLiS, ReGLCiS)
  - Bottom-up generalization from initial grammar
  - Efficient by dynamic programming on parsing graph
  - No parsing required, iterative
- Practical algorithm
  - faster inference
  - faster parsing (non redundant minimal grammar)
- Reduced grammar
  - Easier to read
  - Canonical form ! cf [Clark, 2013]

  → Polynomial dentification in the limit property (cf Remi's talk yesterday)
- Confirmation of good results on proteins (with some preprocessing but no statistical parameters)
- Another step towards practical application of (local-)substitutability

Perspectives

- Practical
  - Choice of initial classes : data-driven heuristics
  - Grammar weighting for biological sequences
  - Better understand why (local-)substitutability seems pertinent for biological sequences. . .
  - Prototype to efficient implementation ?
- Theoretical
  - Better understand/characterize interest of outer loop in generalization wrt *SGL*
  - What happens exactly when sample is not characteristic ?
  - Is it possible to ensure always returning a grammar in the target class ?

And attend Gaelle's thesis (Dec 2014)