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

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

\[ x_1y_1z_1 \in L \land x_1y_2z_1 \in L \land x_2y_1z_2 \in L \Rightarrow x_2y_2z_2 \in L \]
Substitutable Languages

\( L \) is \textit{substitutable} [\textit{Clark & Eyraud, 2007}] if:

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

\[ x_1 y_1 z_1 \in L \land x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \iff x_2 y_2 z_2 \in L) \]
L is substitutable [Clark & Eyraud, 2007] if:

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

\[ x_1 y_1 z_1 \in L \land x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \iff x_2 y_2 z_2 \in L) \]

i.e. \([y_1] = [y_2]\)
Substitutable Languages

\( L \) is \textit{substitutable} \([\text{Clark \& Eyraud, 2007}]\) if:

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

\[ x_1 y_1 z_1 \in L \land x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \Leftrightarrow x_2 y_2 z_2 \in L) \]

i.e. \([y_1] = [y_2] \]

\( L \) is \textit{k, l-substitutable} \([\text{Yoshinaka, 2008}]\) if:

\[ x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1 v, uy_2 v \neq \lambda \]

\[ x_1 uy_1 v z_1 \in L \land x_1 uy_2 v z_1 \in L \Rightarrow (x_2 uy_1 v z_2 \in L \Leftrightarrow x_2 uy_2 v z_2 \in L) \]
**Substitutable Languages**

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

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

$$x_1y_1z_1 \in L \land x_1y_2z_1 \in L \Rightarrow (x_2y_1z_2 \in L \iff x_2y_2z_2 \in L)$$

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

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

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

$$x_1uy_1vz_1 \in L \land x_3uy_2vz_3 \in L \Rightarrow (x_2y_1z_2 \in L \iff x_2y_2z_2 \in L)$$

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

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

$$x_1uy_1vz_1 \in L \land x_1uy_2vz_1 \in L \Rightarrow (x_2uy_1vz_2 \in L \iff x_2uy_2vz_2 \in L)$$
L is **substitutable** [Clark & Eyraud, 2007] if:

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

\[ x_1 y_1 z_1 \in L \land x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \iff x_2 y_2 z_2 \in L) \]

i.e. \([y_1] = [y_2]\)

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

$x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1 v, uy_2 v \neq \lambda$

\[ x_1 u y_1 v z_1 \in L \land x_3 u y_2 v z_3 \in L \Rightarrow (x_2 y_1 z_2 \in L \iff x_2 y_2 z_2 \in L) \]

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

$x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1 v, uy_2 v \neq \lambda$

\[ x_1 u y_1 v z_1 \in L \land x_1 u y_2 v z_1 \in L \Rightarrow (x_2 u y_1 v z_2 \in L \iff x_2 u y_2 v z_2 \in L) \]

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

$x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1 v, uy_2 v \neq \lambda$

\[ x_1 u y_1 v z_1 \in L \land x_1 u y_2 v z_1 \in L \Rightarrow (x_2 u y_1 v z_2 \in L \iff x_2 u y_2 v z_2 \in L) \]
Substitutable Languages

L is substitutable [Clark & Eyraud, 2007] if:
\[ x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^* \land y_1, y_2 \neq \lambda: \]
\[ x_1 y_1 z_1 \in L \land x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \iff x_2 y_2 z_2 \in L) \]
i.e. \[ y_1 = y_2 \]

L is k, l-local substitutable [CGN, 2012] if:
\[ x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1 v, uy_2 v \neq \lambda \]
\[ x_1 uy_1 vz_1 \in L \land x_3 uy_2 vz_3 \in L \Rightarrow (x_2 y_1 z_2 \in L \iff x_2 y_2 z_2 \in L) \]

L is k, l-context-substitutable [Yoshinaka, 2008] if:
\[ x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1 v, uy_2 v \neq \lambda \]
\[ x_1 uy_1 vz_1 \in L \land x_3 uy_2 vz_3 \in L \Rightarrow (x_2 uy_1 vz_2 \in L \iff x_2 uy_2 vz_2 \in L) \]

L is k, l-local-context substitutable [CGN, 2012] if:
\[ x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1 v, uy_2 v \neq \lambda \]
\[ x_1 uy_1 vz_1 \in L \land x_3 uy_2 vz_3 \in L \Rightarrow (x_2 uy_1 vz_2 \in L \iff x_2 uy_2 vz_2 \in L) \]
Substitutable Languages

**L is zero-substitutable** [Clark & Eyraud, 2007] if:
\[ x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, y_1, y_2 \neq \lambda : \]
\[ x_1 y_1 z_1 \in L \land x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \iff x_2 y_2 z_2 \in L) \]
i.e. \( [y_1] = [y_2] \)

**L is \( k, l \)-local substitutable** [CGN, 2012] if:
\[ x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1 v, uy_2 v \neq \lambda : \]
\[ x_1 uy_1 v z_1 \in L \land x_3 uy_2 v z_3 \in L \Rightarrow (x_2 y_1 z_2 \in L \iff x_2 y_2 z_2 \in L) \]

**L is \( k, l \)-context-substitutable** [Yoshinaka, 2008] if:
\[ x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1 v, uy_2 v \neq \lambda : \]
\[ x_1 uy_1 v z_1 \in L \land x_1 uy_2 v z_1 \in L \Rightarrow (x_2 uy_1 v z_2 \in L \iff x_2 uy_2 v z_2 \in L) \]

**L is \( k, l \)-local-context substitutable** [CGN, 2012] if:
\[ x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1 v, uy_2 v \neq \lambda : \]
\[ x_1 uy_1 v z_1 \in L \land x_3 uy_2 v z_3 \in L \Rightarrow (x_2 uy_1 v z_2 \in L \iff x_2 uy_2 v z_2 \in L) \]
Substitutable Languages

\(L\) is zero-substitutable [Clark & Eyraud, 2007] if:

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

\[x_1 y_1 z_1 \in L \land x_1 y_2 z_1 \in L \implies (x_2 y_1 z_2 \in L \iff x_2 y_2 z_2 \in L)\]

i.e. \([y_1] = [y_2]\

\(L\) is \(k, l\)-local substitutable [CGN, 2012] if:

\[x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1 v, uy_2 v \neq \lambda:\]

\[x_1 u y_1 v z_1 \in L \land x_3 u y_2 v z_3 \in L \implies (x_2 y_1 z_2 \in L \iff x_2 y_2 z_2 \in L)\]

i.e. \([y_1] = [y_2]\

\(L\) is \(k, l\)-context-substitutable [Yoshinaka, 2008] if:

\[x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1 v, uy_2 v \neq \lambda:\]

\[x_1 u y_1 v z_1 \in L \land x_1 u y_2 v z_1 \in L \implies (x_2 u y_1 v z_2 \in L \iff x_2 u y_2 v z_2 \in L)\]
Substitutable Languages

L is zero-substitutable [Clark & Eyraud, 2007] if:
\[x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, y_1, y_2 \neq \lambda:\]
\[x_1 y_1 z_1 \in L \land x_1 y_2 z_1 \in L \Rightarrow (x_2 y_1 z_2 \in L \iff x_2 y_2 z_2 \in L)\]
\[\text{i.e. } [y_1] = [y_2]\]

L is \(k, l\)-local substitutable [CGN, 2012] if:
\[x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1 v, uy_2 v \neq \lambda:\]
\[x_1 uy_1 v z_1 \in L \land x_3 uy_2 v z_3 \in L \Rightarrow (x_2 uy_1 v z_2 \in L \iff x_2 uy_2 v z_2 \in L)\]
\[\text{i.e. } [uy_1 v] = [uy_2 v]\]

L is \(k, l\)-context-substitutable [Yoshinaka, 2008] if:
\[x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1 v, uy_2 v \neq \lambda:\]
\[x_1 uy_1 v z_1 \in L \land x_1 uy_2 v z_1 \in L \Rightarrow (x_2 uy_1 v z_2 \in L \iff x_2 uy_2 v z_2 \in L)\]
\[\text{i.e. } [uy_1 v] = [uy_2 v]\]

L is \(k, l\)-local-context substitutable [CGN, 2012] if:
\[x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1 v, uy_2 v \neq \lambda:\]
\[x_1 uy_1 v z_1 \in L \land x_3 uy_2 v z_3 \in L \Rightarrow (x_2 uy_1 v z_2 \in L \iff x_2 uy_2 v z_2 \in L)\]
\[\text{i.e. } [uy_1 v] = [uy_2 v]\]
“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.
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.

...
Adaptation of SGL algorithm \cite{Clark&Eyraud,2007}:
**SGL_{LS}** (Substitution Graph Learner for Local Substitutable languages)

**Input:** Set of sequences $K$ on alphabet $\Sigma$, int $k$, int $l$

**Output:** Grammar $G = \langle \Sigma, N_K, S_K, P_K \rangle$

/* Partition $\text{Sub}(K)$ in Local Substitutability classes */
$C_K \leftarrow \text{LS\_classes}(K, k, l)$ /* $\forall y \in \text{Sub}(K), C_K(y) = C \in C_K : y \in C$ */

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

for $C \in C_K$ do
    /* A non-terminal for each substitutability class */
    $N_K \leftarrow N_K \cup \{[C]\}$

    /* Productions rules for each substring in the class */
    for $y \in C$ do
        if $|y| > 1$ then
            /* Branching rules: a 'CNF' rule for each split */
            for $y_1 \in \Sigma^+, y_2 \in \Sigma^+: y_1y_2 = y$ do
                $P_K \leftarrow P_K \cup \{[C] \rightarrow [C_K(y_1)][C_K(y_2)]\}$
        else
            /* Terminal rule */
            $P_K \leftarrow P_K \cup \{[C] \rightarrow y\}$

    $S_K \leftarrow [C_K(k)] : k \in K$

return $\langle \Sigma, N_K, S_K, P_K \rangle$
Local substitutability classes

\[
\text{LS\_classes()}
\]

**Input:** Set of sequences \( K \) on alphabet \( \Sigma \), int \( k \), int \( l \)

**Output:** \( k, l \) local substitutability classes on \( K \)

/* Build substitutability graph on substrings */

\[
V \leftarrow \{ y \in \Sigma^+: y \in \text{Sub}(K) \}
\]

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

/* Return connected components of graph */

\[
\text{return Connected\_components}(\langle V, E \rangle)
\]

where \( \text{Sub}(K) \) denotes the set of substrings of \( K \)

- This is the unique difference with \( SGL \)!
- To learn other substitutable classes, change \( E \) and \( V \) initialization.
**SGL\textsubscript{LS}**

(Substitution Graph Learner for Local Substitutable languages)

**Input:** Set of sequences $K$ on alphabet $\Sigma$, int $k$, int $l$

**Output:** Grammar $G = \langle \Sigma, N_K, S_K, P_K \rangle$

/* Partition $\text{Sub}(K)$ in Local Substitutability classes */

$C_K \leftarrow \text{LS\_classes}(K, k, l)$ /* $\forall y \in \text{Sub}(K), C_K(y) = C \in C_K : y \in C$ */

/* Build grammar */

$N_K \leftarrow \emptyset$, $P_K \leftarrow \emptyset$

for $C \in C_K$ do

/* A non-terminal for each substitutability class */

$N_K \leftarrow N_K \cup \{[C]\}$

/* Productions rules for each substring in the class */

for $y \in C$ do

if $|y| > 1$ then

/* Branching rules: a 'CNF' rule for each split */

for $y_1 \in \Sigma^+, y_2 \in \Sigma^+: y_1y_2 = y$ do

$P_K \leftarrow P_K \cup \{[C] \rightarrow [C_K(y_1)][C_K(y_2)]\}$

else

/* Terminal rule */

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

end

end

$S_K \leftarrow [C_K(k)]: k \in K$

return $\langle \Sigma, N_K, S_K, P_K \rangle$
Resulting grammar:

\[
\begin{align*}
X_{47} & \rightarrow \text{'yesterday'} \\
X_{46} & \rightarrow X_3X_{43}|X_{11}X_{19}|X_{23}X_{13} \\
X_{45} & \rightarrow X_{20}X_2 \\
X_{44} & \rightarrow X_{20}X_1|X_{45}X_{29}|X_{34}X_{16}|X_9X_{15} \\
X_{43} & \rightarrow X_{20}X_{19}|X_{45}X_{13} \\
X_{42} & \rightarrow \text{'tomorrow'} \\
X_{41} & \rightarrow X_{42}X_{15} \\
X_{40} & \rightarrow X_{30}X_{46}|X_{21}X_{43}|X_{39}X_{19}|X_{25}X_{13}|X_{21}X_{34}|X_8X_{13} \\
N_0 & \rightarrow X_{30}X_{24}|X_{21}X_{31}|X_{10}X_{32}|X_{36}X_{17}|X_{26}X_{15}|X_{39}X_1 \\
& \quad |X_{25}X_{29}|X_{40}X_{41}|X_{21}X_{44}|X_8X_{29}|X_{40}X_{16}|X_{33}X_{15} \\
X_{29} & \rightarrow X_{13}X_{16}|X_4X_{15}|X_{13}X_{41}|X_{38}X_{15} \\
X_{28} & \rightarrow X_{27}X_{47} \\
X_{25} & \rightarrow X_{30}X_{23}|X_{21}X_{45}|X_9X_2 \\
X_{24} & \rightarrow X_3X_{31}|X_{18}X_{32}|X_{37}X_{17}|X_{14}X_{15}|X_{11}X_1|X_{23}X_{29}|X_{46}X_{41} \\
X_{27} & \rightarrow \text{'here'} \\
X_{26} & \rightarrow X_{30}X_{14}|X_{21}X_{12}|X_{10}X_{28}|X_{36}X_{47}|X_9X_{35}|X_{25}X_{38}|X_{40}X_{42} \\
X_{21} & \rightarrow \text{'He'}|X_{30}X_3 \\
X_{20} & \rightarrow \text{'will'} \\
X_{23} & \rightarrow X_3X_{45}|X_{11}X_2 \\
X_{22} & \rightarrow X_6X_{27} \\
X_8 & \rightarrow X_{21}X_{45}|X_9X_2 \\
X_9 & \rightarrow X_{20}X_5|X_{45}X_4|X_{34}X_{42} \\
X_2 & \rightarrow \text{'be'} \\
X_3 & \rightarrow \text{'Major'} \\
\end{align*}
\]
Limitations

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

⇒ 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

⇒ 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

⇒ Parsing time and learning time problems

A solution

Reduce the grammar during the inference

1. About a day for one experiment on a simple set of proteins
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 $\alpha : |\alpha| < |\beta| \land \alpha \Rightarrow \beta$, 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 \(N \rightarrow \alpha = \alpha_1 \ldots \alpha_{|\alpha|}\) implies

\([N] = [\alpha_1]\ldots[\alpha_{|\alpha|}]\).

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 \(\forall y_1y_2 \in [y], [y] \not\subset [y_1][y_2]\).

Sufficient test since for syntactic congruence, we have \([y_1y_2] \supseteq [y_1][y_2]\)
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 : $C_K$

**Output**: Set of substitutability classes satisfying primality test in $C_K$ : $\mathcal{P}$

$\mathcal{P} \leftarrow \emptyset$

for $C \in C_K$ do

| If $(\forall C' \in C_K : \forall y \in C, \exists y' \in C', \exists v \in \Sigma^+, y = y'v)$
| and $(\forall C' \in C_K : \forall y \in C, \exists y' \in C', \exists u \in \Sigma^+, y = uy')$
| $\mathcal{P} \leftarrow \mathcal{P} \cup C$

Primality test in $K$ not in $L$!

but works well in practice and if the sample is informative enough.
On the example
On the example

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

F. Coste (Dyliss, Inria)
ReG.*iS
ICGI'14 16 / 26
On the example

- General Major was here yesterday morning
- He will be gone tomorrow evening
- General Major will be there tomorrow morning
- General Major went here yesterday morning
- He will be gone tomorrow evening
- He was here yesterday
- went here yesterday
- will be there tomorrow
- morning
- evening
- there
- was
- gone
- went
- General Major
ReGLiS Part1  (Learning Reduced Grammar by $k, l$-Local Substitutability) simplified !

**Input**: Set of sequences $K$ on alphabet $\Sigma$, int $k$, int $l$

**Output**: Grammar $G = \langle \Sigma, N_K, S_K, P_K \rangle$

/* Prime substitutability classes on $K$ */

$C_K \leftarrow \text{Primes}(\text{LS_classes}(K, k, l))$

/* Build initial grammar */

$N_K \leftarrow \emptyset, P_K \leftarrow \emptyset$

for $C \in C_K$ do

/* A non-terminal for each $K$-prime */

$N_K \leftarrow N_K \cup \{\llbracket C \rrbracket\}$

/* A direct production for each substring in the class */

for $y \in C$ do

$P_K \leftarrow P_K \cup \{\llbracket C \rrbracket \rightarrow y\}$

/* So far, we have built the initial grammar */
ReGLiS Part1  (Learning Reduced Grammar by $k, l$-Local Substitutability) simplified!

**Input:** Set of sequences $K$ on alphabet $\Sigma$, int $k$, int $l$

**Output:** Grammar $G = \langle \Sigma, N_K, S_K, P_K \rangle$

/* Prime substitutability classes on $K$ */

$C_K \leftarrow \text{Primes}(\text{LS_classes}(K, k, l))$

/* Build initial grammar */

$N_K \leftarrow \emptyset, P_K \leftarrow \emptyset$

**for** $C \in C_K$ **do**

/* A non-terminal for each $K$-prime */

$N_K \leftarrow N_K \cup \{[C]\}$

/* A direct production for each substring in the class */

**for** $y \in C$ **do**

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

/* So far, we have built the initial grammar */
Initial 'bottom' grammar

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

- 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)
  \[
  \frac{N_1 \rightarrow \ldots \beta \ldots, \ N_2 \Rightarrow \ast \beta}{N_1 \rightarrow \ldots N_2 \ldots, \ N_2 \Rightarrow \ast \beta}
  \]
- 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:

![Parsing Graph](image-url)
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:

\[ N \rightarrow aN_1e \]
Example

Let us consider that so far the grammar is such that

\[ P = \{ \ldots ; N \rightarrow abcde | \ldots ; N_1 \rightarrow bcd | \ldots ; N_2 \rightarrow cde | \ldots ; N_3 \rightarrow ab | \ldots ; N_4 \rightarrow de | \ldots ; \ldots \} \]

(where \( a, b, c, d, e \) is a terminal or non-terminal symbols)

The parsing graph for \( N \rightarrow abcde \) is then:

![Parsing Graph](image)

Can be reduced! (025 strict subsequence 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
\[ P = \{ \ldots ; N \rightarrow abcde | \ldots ; N_1 \rightarrow bcd | \ldots ; N_2 \rightarrow cde | \ldots ; N_3 \rightarrow ab | \ldots ; N_4 \rightarrow de | \ldots ; \ldots \} \]
(where \( a, b, c, d, e \) is a terminal or non-terminal symbols)
The parsing graph for \( N \rightarrow abcde \) is then:

\[ N \rightarrow aN_1e | N_3N_2 \]
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:

\[ N \rightarrow aN_1e \mid N_3N_2 \]

Implemented by dynamic programming on vertices in function

\texttt{Non\_redundant\_rhs()}
/* Generalization */

repeat
  
  $P \leftarrow P_K \cup \emptyset$
  
  /* Branching rules */
  
  for $(\llbracket C \rrbracket \rightarrow \alpha) \in P$ ordered by increasing $|\alpha|$ do
    
    $PG \leftarrow \text{Build}_\text{parsing}_\text{graph}(\alpha, P)$
    
    for $\beta \in \text{Non}_\text{redundant}_\text{rhs}(PG)$ do
      
      $P_K \leftarrow P_K \cup (\llbracket C \rrbracket \rightarrow \beta)$
  
until $P_K = P$

$S_K \leftarrow \llbracket C_K(k) \rrbracket : k \in K$

return $\langle \Sigma, N_K, S_K, P_K \rangle$

---

**Build_parsing_graph**

**Input**: Sequence $\alpha$, Set of rules $P$

**Output**: Parsing graph $\langle V, E \rangle$

$V \leftarrow \{i \in [0, |\alpha|]\}$ /* vertices */; $E \leftarrow \emptyset$ /* labeled directed edges */

for $i \in V$ do

  for $j \in V : i < j$ and $(i, j) \neq (0, |\alpha|)$ do

    if $\exists (\llbracket C \rrbracket \rightarrow \alpha[i + 1, j]) \in P$ then

      $E \leftarrow E \cup (i, j, \llbracket C \rrbracket)$

return $\langle V, E \rangle$
/* Generalization */
repeat

\[ P \leftarrow P_K; P_K \leftarrow \emptyset \]

/ * Branching rules */

for \((\llbracket C \rrbracket \rightarrow \alpha) \in P\) ordered by increasing \(|\alpha|\) do

\[ PG \leftarrow \text{Build\_parsing\_graph}(\alpha, P) \]

for \(\beta \in \text{Non\_redundant\_rhs}(PG)\) do

\[ P_K \leftarrow P_K \cup (\llbracket C \rrbracket \rightarrow \beta) \]

until \(P_K = P\)

\[ S_K \leftarrow \llbracket C_K(k)\rrbracket: k \in K \]
return \(\langle \Sigma, N_K, S_K, P_K \rangle\)

---

**Build\_parsing\_graph**

**Input:** Sequence \(\alpha\), Set of rules \(P\)

**Output:** Parsing graph \(\langle V, E \rangle\)

\[ V \leftarrow \{i \in [0, |\alpha|]\} \text{ / * vertices / *}; E \leftarrow \emptyset \text{ / * labeled directed edges / *} \]

for \(i \in V\) do

for \(j \in V: i < j\) and \((i,j) \neq (0, |\alpha|)\) do

if \(\exists (\llbracket C \rrbracket \rightarrow \alpha[i + 1, j]) \in P\) then

\[ E \leftarrow E \cup (i,j, \llbracket C \rrbracket) \]

return \(\langle V, E \rangle\)
Final Grammar

\[ S \rightarrow X_3 \ X_4 \ X_2 \]
\[ X_1 \rightarrow \text{was} \ | \ \text{went} \]
\[ X_2 \rightarrow \text{morning} \ | \ \text{evening} \]
\[ X_3 \rightarrow \text{He} \ | \ \text{General Major} \]
\[ X_4 \rightarrow \text{will be } X_5 \ \text{tomorrow} \ | \ X_1 \ \text{here yesterday} \]
\[ X_5 \rightarrow \text{there} \ | \ \text{gone} \]
Complexity

- Complexity: $\mathcal{O}(\max(l^3, l \cdot t))$
  - $l$: 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)

<table>
<thead>
<tr>
<th>Subst.</th>
<th>Zinc finger</th>
<th></th>
<th></th>
<th>MPI phos.</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
<td>F-measure</td>
<td>Precision</td>
<td>Recall</td>
<td>F-measure</td>
</tr>
<tr>
<td>Subst.</td>
<td>1</td>
<td>0.1</td>
<td>0.36</td>
<td>1</td>
<td>0.15</td>
<td>0.26</td>
</tr>
<tr>
<td>3,3-LS</td>
<td>1</td>
<td>0.2</td>
<td>0.33</td>
<td>1</td>
<td>0.5</td>
<td>0.67</td>
</tr>
<tr>
<td>4,4-LS</td>
<td>1</td>
<td>0.25</td>
<td>0.4</td>
<td>1</td>
<td>0.6</td>
<td>0.75</td>
</tr>
<tr>
<td>5,5-LS</td>
<td>1</td>
<td>0.33</td>
<td>0.5</td>
<td>1</td>
<td>0.67</td>
<td>0.8</td>
</tr>
<tr>
<td>6,6-LS</td>
<td>1</td>
<td>0.5</td>
<td>0.67</td>
<td>1</td>
<td>0.62</td>
<td>0.77</td>
</tr>
<tr>
<td>7,7-LS</td>
<td><strong>1</strong></td>
<td><strong>0.55</strong></td>
<td><strong>0.7</strong></td>
<td>1</td>
<td>0.53</td>
<td>0.69</td>
</tr>
<tr>
<td>SCFG</td>
<td>1</td>
<td>0.1</td>
<td>0.18</td>
<td>1</td>
<td>0.3</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td>0.15</td>
<td>1</td>
<td>0.26</td>
<td>0.5</td>
<td>1</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>0.75</td>
<td>0.87</td>
<td>0.85</td>
<td>0.98</td>
<td>0.89</td>
<td>0.93</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>PS00219</th>
<th></th>
<th></th>
<th>PS00063</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
<td>F-measure</td>
<td>Precision</td>
<td>Recall</td>
<td>F-measure</td>
</tr>
<tr>
<td>Subst.</td>
<td>1</td>
<td>0.2</td>
<td>0.33</td>
<td>1</td>
<td>0.23</td>
<td>0.37</td>
</tr>
<tr>
<td>3,3-LS</td>
<td><strong>1</strong></td>
<td><strong>0.72</strong></td>
<td><strong>0.84</strong></td>
<td>1</td>
<td>0.58</td>
<td>0.73</td>
</tr>
<tr>
<td>4,4-LS</td>
<td>1</td>
<td>0.7</td>
<td>0.82</td>
<td>1</td>
<td>0.6</td>
<td>0.75</td>
</tr>
<tr>
<td>5,5-LS</td>
<td>1</td>
<td>0.68</td>
<td>0.8</td>
<td>1</td>
<td>0.66</td>
<td>0.8</td>
</tr>
<tr>
<td>6,6-LS</td>
<td>1</td>
<td>0.6</td>
<td>0.75</td>
<td><strong>1</strong></td>
<td>0.7</td>
<td><strong>0.82</strong></td>
</tr>
<tr>
<td>7,7-LS</td>
<td>1</td>
<td>0.5</td>
<td>0.67</td>
<td>1</td>
<td>0.65</td>
<td>0.79</td>
</tr>
<tr>
<td>Prosite</td>
<td>1</td>
<td>0.6</td>
<td>0.75</td>
<td>1</td>
<td>0.8</td>
<td>0.89</td>
</tr>
<tr>
<td>SCFG</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0.05</td>
<td>0.1</td>
</tr>
</tbody>
</table>

[Dyrka & Nebel, 2009]
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)