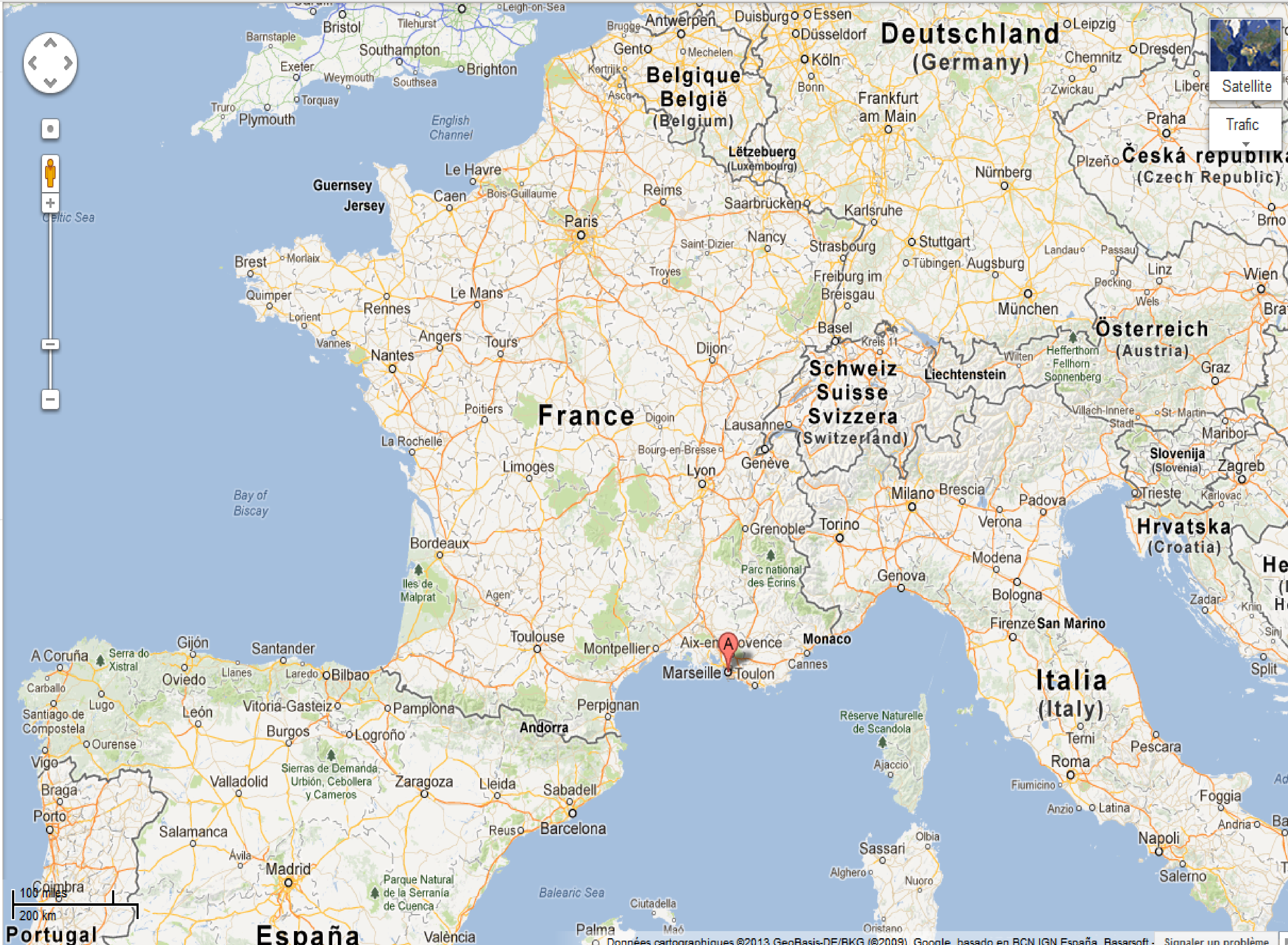
The basis of Distributional Learning for the inference of Non-Regular Languages

Rémi Eyraud
Aix-Marseille Université
QARMA team
LIS


Deutschland
(Germany)
France
Belgique
België
(Belgium)
Lëtzebuerg
(Luxembourg)
Schweiz
Suisse
Svizzera
(Switzerland)
Liechtenstein
Österreich
(Austria)
Česká republika
(Czech Republic)
Slovenija
(Slovenia)
Hrvatska
(Croatia)
Italia
(Italy)
San Marino
Monaco
Andorra
España
Portugal
Guernsey
Jersey
Celtic Sea
English Channel
Bay of Biscay
Balearic Sea
Bristol
Barnstaple
Tilehurst
Southampton
Brighton
Exeter
Weymouth
Southsea
Torquay
Truro
Plymouth
Leigh-on-Sea
Brugge
Antwerpen
Gent
Mechelen
Kortrijk
Ascq
Duisburg
Essen
Düsseldorf
Köln
Bonn
Frankfurt am Main
Leipzig
Dresden
Chemnitz
Zwickau
Praha
Plzeň
Nürnberg
Brno
Karlsruhe
Saarbrücken
Stuttgart
Tübingen
Augsburg
Landau
Passau
Linz
Wien
Pocking
Wels
München
Freiburg im Breisgau
Strasbourg
Basel
Kreis 11
Wilten
Hefferthorn - Fellhorn - Sonnenberg
Graz
Villach-Innere Stadt
St. Martin
Maribor
Zagreb
Trieste
Karlovac
Le Havre
Caen
Bois-Guillaume
Reims
Paris
Nancy
Saint-Dizier
Troyes
Brest
Morlaix
Quimper
Lorient
Vannes
Rennes
Le Mans
Angers
Tours
Nantes
Dijon
Poitiers
Digoin
Lausanne
Genève
Bourg-en-Bresse
La Rochelle
Limoges
Lyon
Milano
Brescia
Padova
Verona
Grenoble
Torino
Parc national des Ecrins
Bordeaux
Iles de Malprat
Modena
Genova
Bologna
Agen
Firenze
Toulouse
Montpellier
Aix-en-Provence
Marseille
Toulon
Cannes
A
Split
Zadar
Knin
Sinj
Gijón
Santander
A Coruña
Serra do Xistral
Oviedo
Llanes
Laredo
Bilbao
Carballo
Lugo
Santiago de Compostela
León
Vitoria-Gasteiz
Pamplona
Perpignan
Ourense
Burgos
Logroño
Vigo
Sierras de Demanda, Urbión, Cebollera y Cameros
Braga
Valladolid
Zaragoza
Lleida
Sabadell
Porto
Barcelona
Salamanca
Reus
Ávila
Madrid
Coimbra
Parque Natural de la Serranía de Cuenca
València
Palma
Ciutadella
Maó
Réserve Naturelle de Scandola
Ajaccio
Terni
Pescara
Roma
Fiumicino
Anzio
Latina
Foggia
Andria
Bari
Napoli
Salerno
Olbia
Sassari
Alghero
Nuoro
Oristano
Satellite
Trafic
100 miles
200 km
Données cartographiques ©2013 GeoBasis-DE/BKG (©2009), Google, basado en BCN IGN España, Basarsoft -
Signaler un problème

CANNES, FRANCE
pure médias
CANNES, FRANCE
Cannes, France Reuters
MMIT
LIVE CNN
NAS 33.02
© DR
FRANCE
Lille
Rennes
Paris
Strasbourg
Toulouse
Lyon
Cannes
CNN
DEFENDANT KILLED IN BAGHDAD
CHINA'S PRES. BEG
Bristol
Southampton
Brighton
Exeter
Weymouth
Southsea
Torquay
Barnstaple
Tilehurst
Leigh-on-Sea
Duisburg
Essen
Kortrijk
Bordeaux
Toulouse
Montpellier
Aix-en-Provence
Marseille
Toulon
Cannes
Monaco
Grenoble
Torino
Milano
Brescia
Padova
Verona
Trieste
Modena
Bologna
Genova
Firenze
San Marino
Hrvatska (Croatia)
Zadar
Split
Italia (Italy)
Terni
Roma
Pescara
Fiumicino
Anzio
Latina
Foggia
Andria
Bari
Napoli
Salerno
Réserve Naturelle de Scandola
Ajaccio
Sassari
Olbia
Alghero
Nuoro
Oristano
Parc national des Écrins
Iles de Malprat
Agen
Perpignan
Andorra
A Coruña
Serra do Xistral
Gijón
Santander
Oviedo
Llanes
Laredo
Bilbao
Carballo
Lugo
Santiago de Compostela
León
Vitoria-Gasteiz
Pamplona
Burgos
Logroño
Ourense
Vigo
Braga
Porto
Valladolid
Sierras de Demanda, Urbión, Cebollera y Cameros
Zaragoza
Lleida
Sabadell
Barcelona
Reus
Salamanca
Ávila
Madrid
Coimbra
Parque Natural de la Serranía de Cuenca
Balearic Sea
Ciutadella
Maó
Palma
València
Portugal
España
100 miles
200 km
Données cartographiques ©2013 GeoBasis-DE/BKG (©2009), Google, basado en BCN IGN España, Basarsoft - Signaler un problème

Outline

- **Introduction**
- Learning Substitutable languages
- Prime congruence classes
- Extension to tree and graph grammars
- A dual approach
- Conclusion

Some basic definitions (Formal Language Theory)

- An alphabet is a finite set of symbols. Ex.:
  - $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - $\Sigma = \{A, C, T, G\}$
  - Σ = set of Part-Of-Speech tags
  - $\Sigma = \{0, 1\}$, usually denoted $\{a, b\}$
- A word/string on an alphabet is a finite sequence of symbols
  - 000042 or 84 or ε
  - VBN-TL NNS-TL IN-TL NP-TL

Some basic definitions (Formal Language Theory)

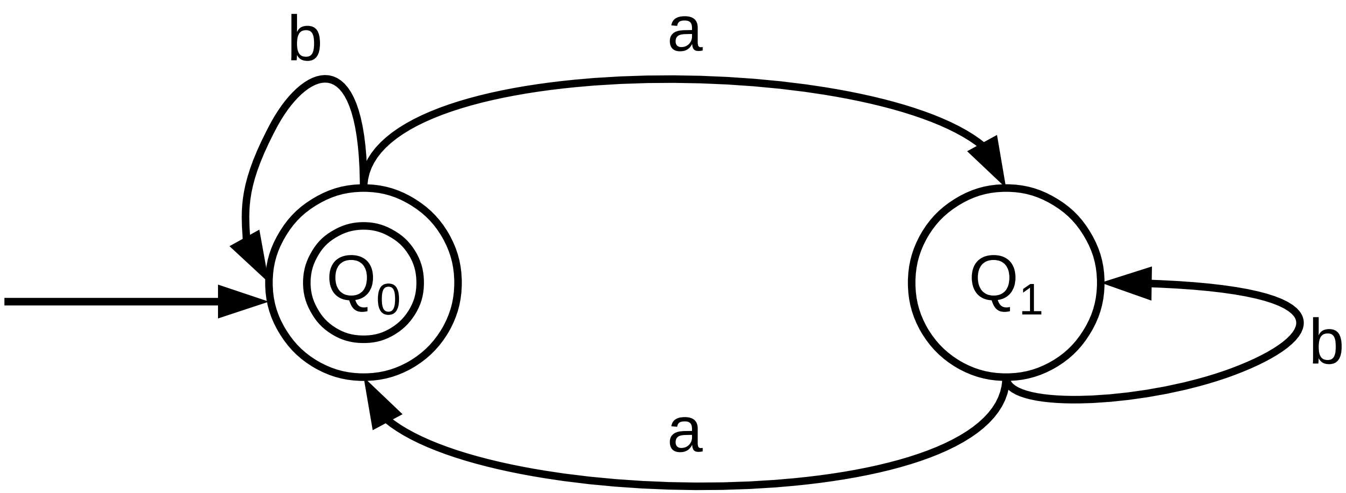
- A language over an alphabet is a (possibly non-finite) set of strings over this alphabet.
  - L = {10, 11, 12, 13, 14, 15, 16, 17, 18, 19}
  - L = all natural numbers
  - L = all DNA sequences that correspond to a gene
  - L = all POS tag sequences that correspond to an English sentence
  - $L = \{a^n b^n : n>0\}$

Some basic definitions (Formal Language Theory)

- If a language is infinite (or too big to be finitely represented), we need a finite representation to be able to handle it : we call this representation a **grammar**.
- Well-known grammar classes : the Chomsky hierrachy
  - Regular grammars: productions $A \rightarrow aB$ or $A \rightarrow \varepsilon$
  - Context-free grammars: $A \rightarrow BC$ or $A \rightarrow a$ (normal form)
  - Context sensitive grammars: $\alpha A \beta \rightarrow \alpha \gamma \beta$
- With each of these classes of grammars is associated a class of languages.

Some basic definitions (Formal Language Theory)

- DFA:



- Regular grammars:

  $Q_0 \rightarrow b\ Q_0,\ Q_0 \rightarrow a\ Q_1,\ Q_0 \rightarrow \varepsilon$

  $Q_1 \rightarrow b\ Q_1,\ Q_1 \rightarrow a\ Q_0$

Some basic definitions (Formal Language Theory)

- Context-free grammars:
  - The alphabet of the language : Σ
  - A finite set of variables (called non-terminals) : N
  - A set of context-free rules: $P \subset N \rightarrow (N \cup \Sigma)^*$
  - A special non-terminal (the axiom) : S
- The language represented by a context-free grammar is the set of strings over Σ that one can obtain from the axiom using the rules of P.

Some basic definitions (Formal Language Theory)

- Context-free grammars:
  - The alphabet of the language : Σ
  - A finite set of variables (called non-terminals) : N
  - A set of context-free rules: $P \subset N \rightarrow (N \cup \Sigma)^*$
  - A special non-terminal (the axiom) : S
- Example: $\Sigma=\{a, b\}$, $N=\{S\}$, $P=\{S \rightarrow a\ S\ b,\ S \rightarrow a\ b\ \}$

What is the language of this grammar?

Some basic definitions (Formal Language Theory)

- Context-free grammars:
  - The alphabet of the language : Σ
  - A finite set of variables (called non-terminals) : N
  - A set of context-free rules: $P \subset N \rightarrow (N \cup \Sigma)^*$
  - A special non-terminal (the axiom) : S
- Example: $\Sigma=\{a, b\}$, $N=\{S\}$, $P=\{S \rightarrow a\ S\ b,\ S \rightarrow a\ b\ \}$

  What is the language of this grammar?

  - $\{a^n b^n : n>0\}$
  - $n = 3$: $S \rightarrow a\ S\ b \rightarrow a\ a\ S\ b\ b \rightarrow a\ a\ a\ b\ b\ b$

    We write $S \rightarrow^* aaabbb$

Grammatical Inference

- We are interested in algorithms that are able to learn a class of languages using a given class of grammars:
  - A learning algorithm is fed with data corresponding to an unknown language of the class,
  - It outputs a hypothesis (i.e. a grammar of the class) that is a representation of a language,
  - If the outputed grammar is an “acceptable” representation of the unknown language then the algorithm has learnt the language
  - If it can learn all languages of the class, then we say that the algorithm learns the class.

Grammatical Inference

- When can we say that an algorithm is able to learn?
  - Experimentally validate (e.g. Cross-validation)

Corpus		
a a a a a b b b b b	Yes	Learning sample
b a b b b a a b b	No	
a b b	No	
a a a a a a a a a a a b b b b b b b b b b b	Yes	
a b a b a b a b	No	
a b	Yes	test sample

  - Fulfills conditions of a formal definition of learning

Grammatical Inference

- When can we say that an algorithm is able to learn?
  - Practically validate (e.g. cross-validation)
  - Fulfills conditions of a formal definition of learning:
    - Probably Approximatively Correct (PAC) paradigm

The probability that the error rate of the output *h* of the agorithm is greater than a epsilon is less than a delta: $\Pr(\mathrm{error}(h) < \varepsilon) > 1 - \delta$

Grammatical Inference

- When can we say that an algorithm is able to learn?
  - Practically validate (e.g. cross-validation)
  - Fulfills conditions of a formal definition of learning:
    - Probably Approximatively Correct (PAC) paradigm
    - Identification in the limit

Input: an infinite (complete) sequence of data

Behaviour: for each new data a hypothesis is outputed

Learning: for each possible sequence, there exists a moment at which the algorithm **converges** to a hypothesis that is equivalent to the **target** grammar, and it never changes its hypothesis after.

Grammatical Inference

- What kind of data?
  - Examples (of sentences, DNA codes, bird songs, ...)
  - Example and counter-examples
  - Structured examples (trees, skeletons, ...)
  - Queries to an oracle
  - ...

Grammatical Inference

- Nice results for regular languages:
  - Efficient identification from positive and negative examples, RPNI [Oncina & Garcia, 92]
  - Identification from positive examples only of subclasses: reversible [Angluin, 82], locally testable, ...
  - PAC-learning of the whole class from positive examples (with restrictions on the distribution of examples) [Clark & Thollard, 02]
  - Learning of the whole class using membership and equivalence queries [Angluin, 87]

Grammatical Inference

- Nice results for regular grammars:
- Few postives results for context-free grammars (prior to the works presented today)
  - Efficient identification of small subclasses from postive and negative examples (reduction to the regular case)
  - Identification of a very restrictive subclass from positive examples (very simple grammars [Yolomori, 02])
  - The whole class from skeletons [Sakakibara, 92]
- No positive result for contex-sensitive grammars

Grammatical Inference

- Why are regular languages a success story?
  - Strong link between the representation and the structure of the language (residual, Nerode equivalence classes, ...).
- Slogan: “The structure of the representation should be based on the structure of the language, not something arbitrarily imposed on it from outside”
  - Identify some structure in the language
  - Show how that structure can be observed
  - Construct a representation based on that structure

Outline

- Introduction
- **Learning Substitutable languages**
- Prime congruence classes
- Extension to tree and graph grammars
- A dual approach
- Conclusion

Key idea

- Learn a structurally defined class of languages
- Use only examples of the language
- Rely on the notion of *Context:*

A string u appear in the context (l,r) in a string w if w=lur.

The set of all contexts of u in a language L is written $C_L(u) = \{ (l,r) : lur \text{ in } L\}$.

For instance, the substring ab appears in the context (a,b) in the word aabb.

Syntactic congruence

- Well-studied relation structuring languages
- *u* and *v* are syntactically congruent w.r.t. a language L iff for all *l*,*r* in Σ*, *lur* is in L iff *lvr* is in L ($u \equiv_L v$).
- In term of context, $u \equiv_L v$ iff $C_L(u) = C_L(v)$.
- [u] is congruence class of *u*, i.e. the set of all substrings *v* such that $u \equiv_L v$
- Example: L={bcb,baab,cb,aab}, $c \equiv_L aa$, $bcb \equiv_L baab$, [bcb] = [baab] ≠ [cb]

A weaker relation

- *The weak substitutability*:

  u and *v* are weakly substitutable w.r.t. a language L, iff there exist *l*,*r* in Σ^*, *lur* is in L iff *lvr* is in L

  Notation : $u \approx_L v$

- In term of set of contexts, $u \approx_L v$ if and only if $C_L(u) \cap C_L(v) \neq \emptyset$

Substitutability

The syntactic congruence is the most interesting: u and v always appear in the same context (then they can be generated by the same non-terminal, for instance).

But: from a finite set of examples, we can only observe the weak substitutability.

We can unify these notions in order to ensure the observability of the syntactic congruence

Substitutable languages

- A language L is substitutable iff for all u and v in Σ^*, $u \approx_L v$ implies $u \equiv_L v$, i.e. the weak substitutability implies the syntactic congruence.

- The sets of contexts of two substrings of words of L are either disjoint or identical.
- In other words:

  If lur, lvr and l'ur' are in L then l'vr' is in L.

Examples

- Σ^* is substitutable.
- $\{a^n \mid n>0\}$ is substitutable (all contexts of a substring have the form (a^k,a^l)).
- $\{wcw^R \mid w \text{ in } (a,b)^*\}$ is substitutable.
- $\{a^n cb^n \mid n>0\}$ is substitutable.
- $\{w : |w|_a = |w|_b \text{ and } |w|_c = |w|_d\}$ is substitutable.

- $\{a^n b^n \mid n>0\}$ is not substitutable: for instance, we have $a \approx_L aab$ but not $a \equiv_L aab$
- $\{a,aa\}$ is not substitutable (a and aa share the context $(\varepsilon,\varepsilon)$ but not the context (ε,a))

Algorithm: main ideas

- We want to compute the syntactic classes of the language from the examples, together with their mutual structure.
- We are going to use the fact that $[u][v] \subseteq [uv]$ by creating the rules $[uv] \rightarrow [u][v]$

- Algorithmic trick: the *Substitution graph*
  - each distinct substring of the learning sample is a node.
  - There is an edge between two nodes if they appear in the same context(s).

Running Example

LS={c;aca;bcb;abcba;aacaa} (palindrome with a center marked)

c bcba abcb

abcba aca acaa ac

bcb ca aaca

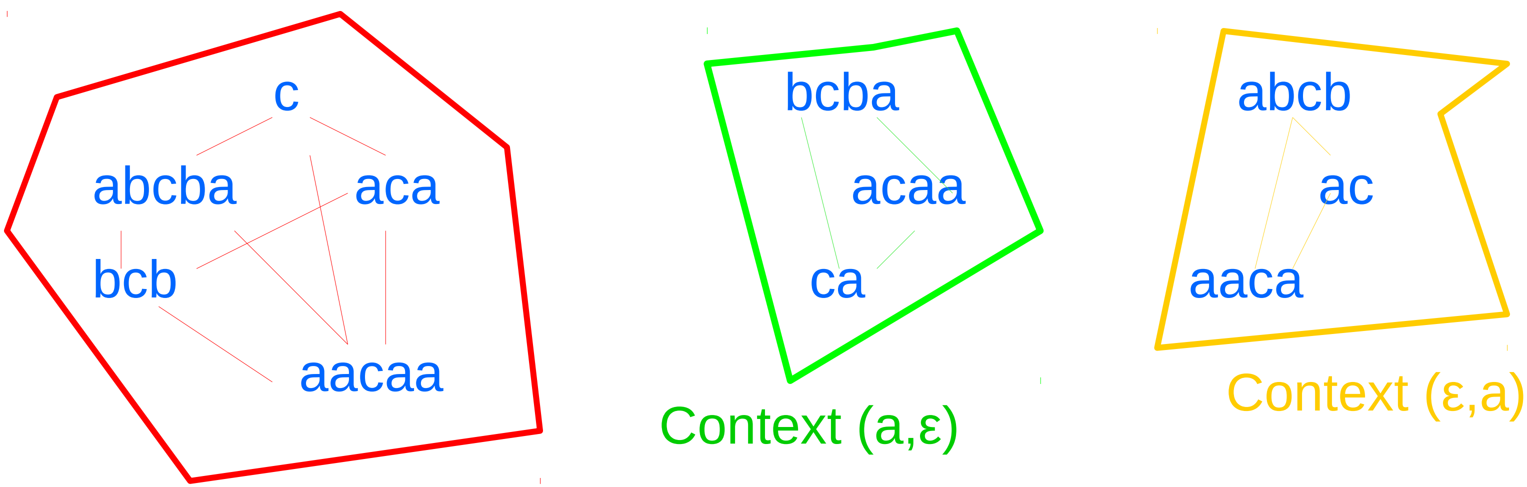
aacaa

a b ab abcb …

bc ba aac caa

Running example

LS={c;aca;bcb;abcba;aacaa}



Empty context (ε,ε)

a b ab abcb …

bc ba aac caa

First step: create a non terminal for each component.

[c] (=[aca]=[abcba]=[bcb]=[aacaa])

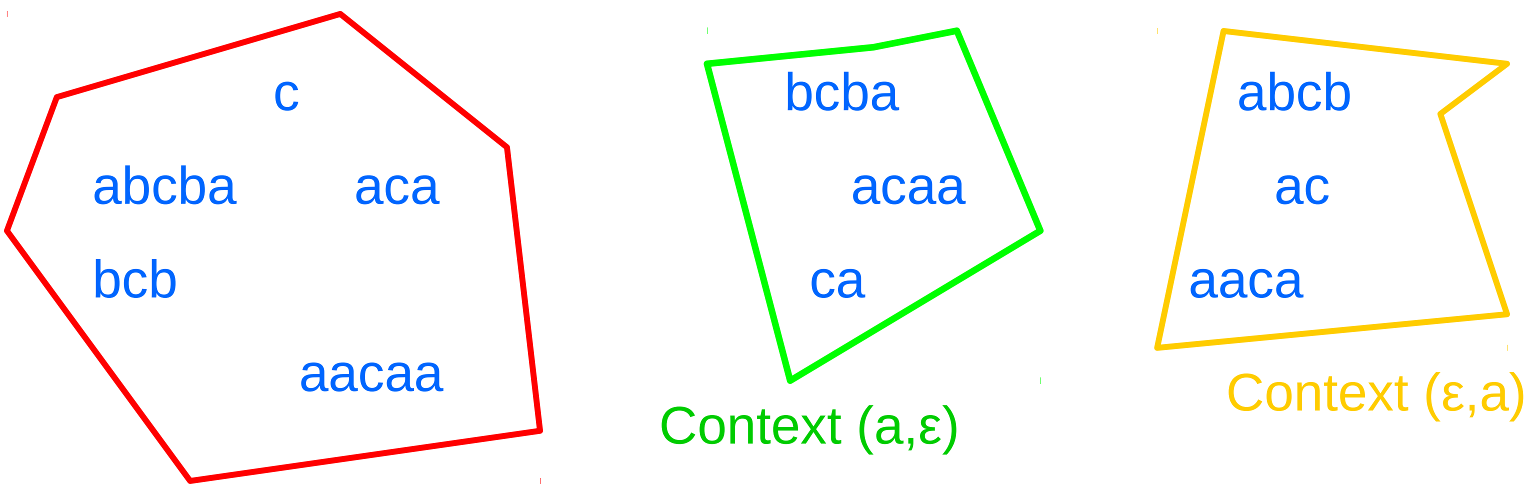
[ca] (=[bcba]=[acaa])

[ac] (=[abcb]=[aaca])

But also: [ab], [abcb], [bc], [aac], [ba]…

Running example

LS={c;aca;bcb;abcba;aacaa}



Context (ε,a)

Context (a,ε)

Empty context (ε,ε)

a b ab abcb …

bc ba aac caa

Next step: create the rules for the letters of the alphabet.

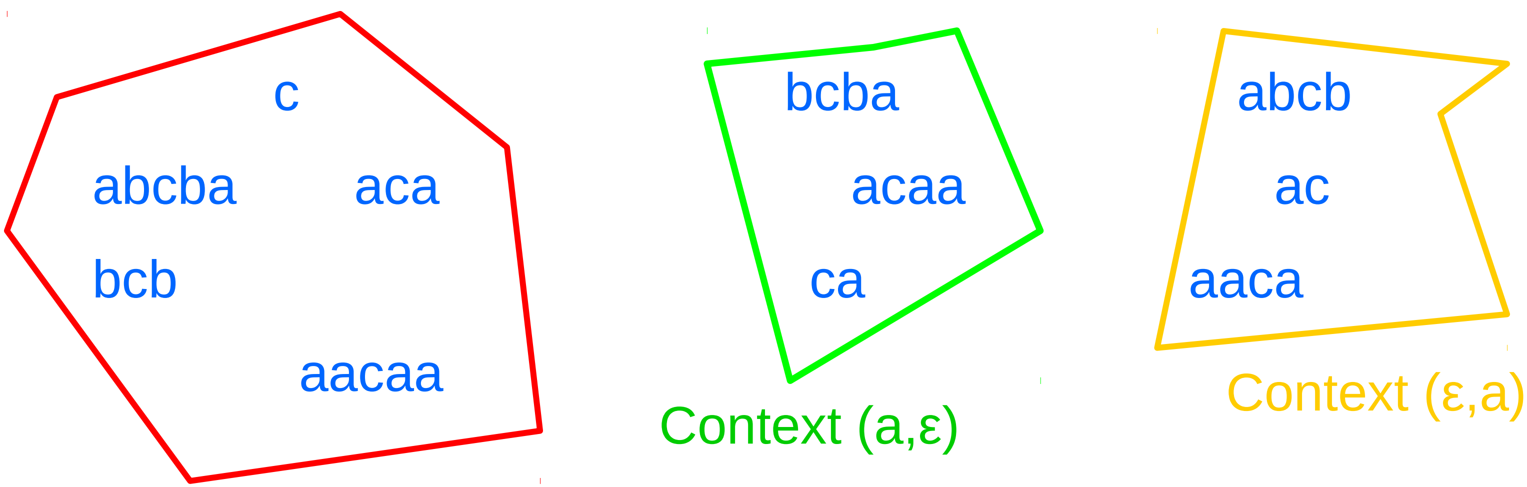
[a] → a

[b] → b

[c] → c

Running example

LS={c;aca;bcb;abcba;aacaa}



Empty context (ε,ε)

a b ab abcb …

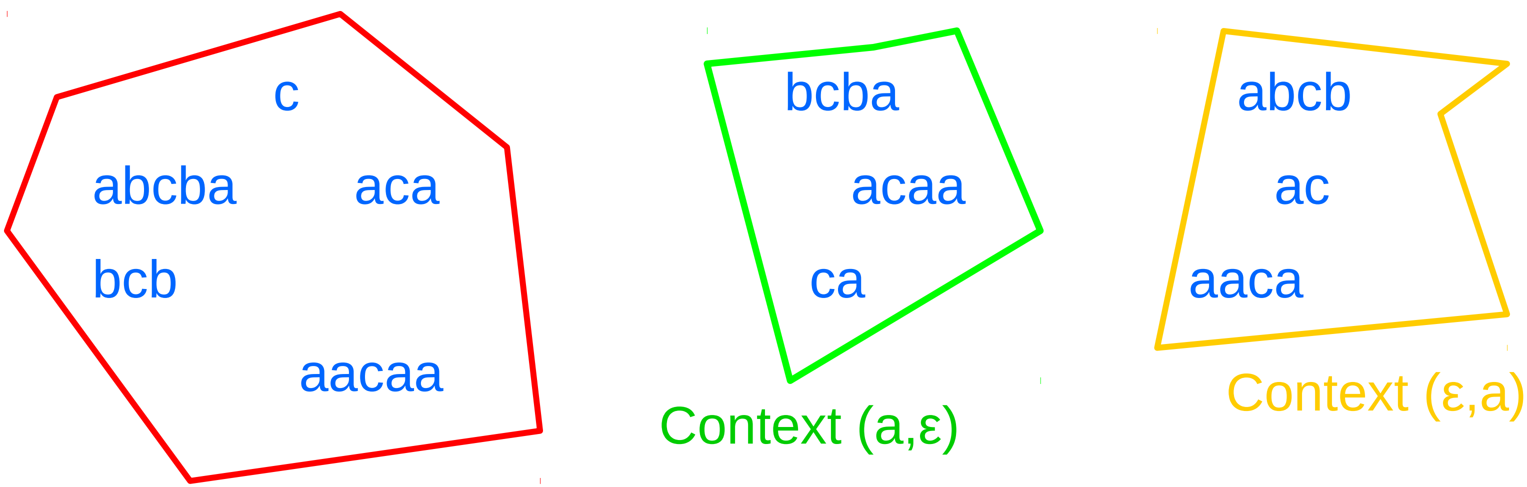
bc ba aac caa

Third step: create the rules for each component.

[w] → [u][v] when uv is a member of the component of w.

Running example

LS={c;aca;bcb;abcba;aacaa}



Empty context (ε,ε)

a b ab abcb …

bc ba aac caa

Component (ε,ε):

[c] → [a][bcba], [c] → [ab][cba], c → [abc][ba], [c] → [abcb][a]

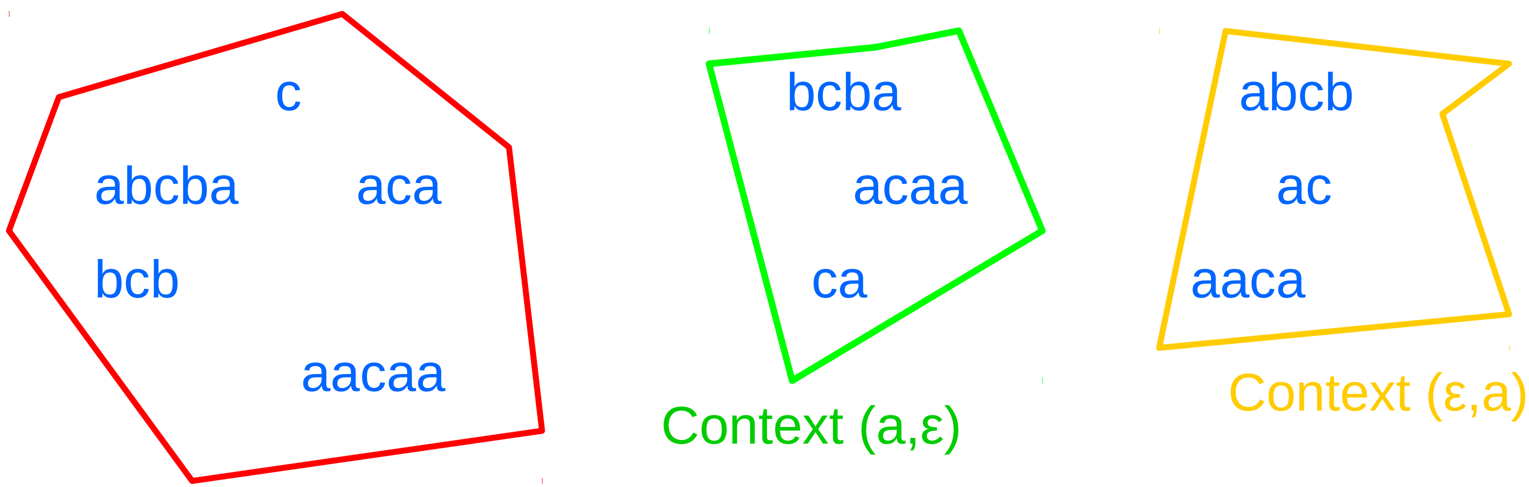
[c] → [a][ca], [c] → [ac][a]

[c] → [b][cb], [c] → [bc][b]

[c] → [a][acaa], [c] → [aa][caa], [c] → [aac][aa], [c] → [aaca][a]

Running example

LS={c;aca;bcb;abcba;aacaa}



Empty context (ε,ε)

a b ab abcb …

bc ba aac caa

Component (ε,ε):

[c] → [a] [ca], [c] → [ab] [cba], [c] → [abc] [ba], [c] → [ac] [a]

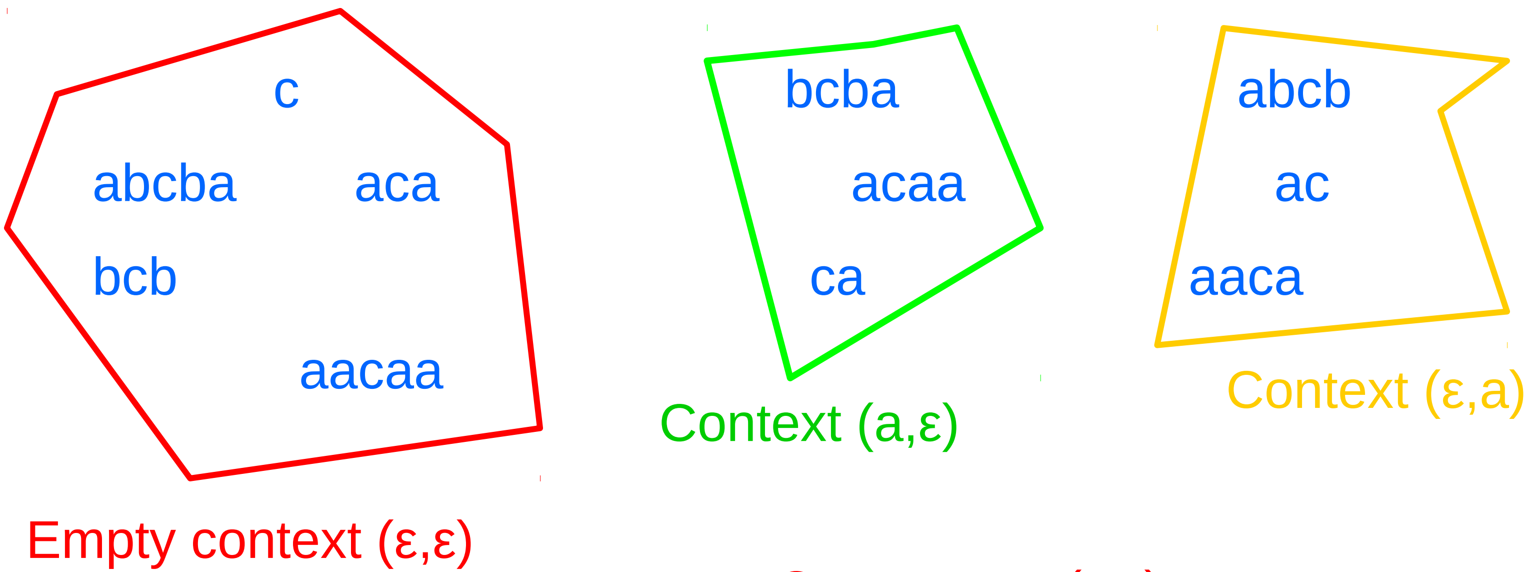
[c] → [a] [ca], [c] → [ac] [a]

[c] → [b] [cb], [c] → [bc] [b]

[c] → [a] [ca], [c] → [aa] [caa], [c] → [aac] [aa], [c] → [ac] [a]

Running example

LS={c;aca;bcb;abcba;aacaa}



a b ab abcb …

bc ba aac caa

Component (ε,ε):

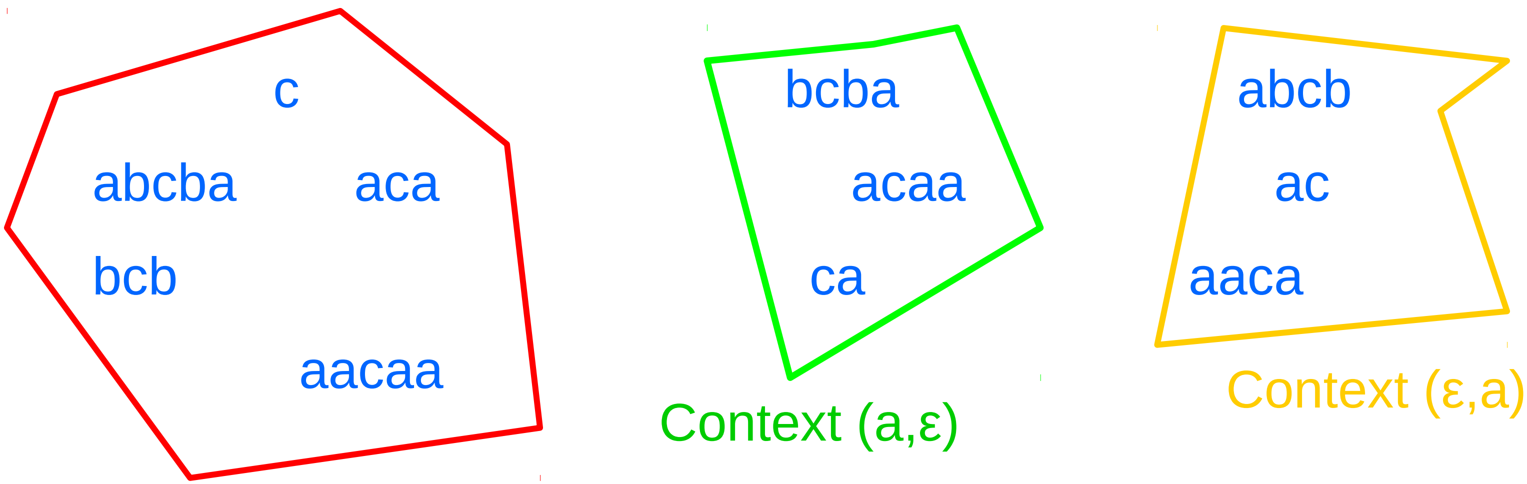
[c]→[a] [ca], [c]→[ac] [a],

[c]→[ab] [cba], [c]→[abc] [ba], [c]→[b] [cb],

[c]→[bc] [b], [c]→[aa] [caa], [c]→[aac] [aa],

Running example

LS={c;aca;bcb;abcba;aacaa}



Empty context (ε,ε)

a b ab abcb …

bc ba aac caa

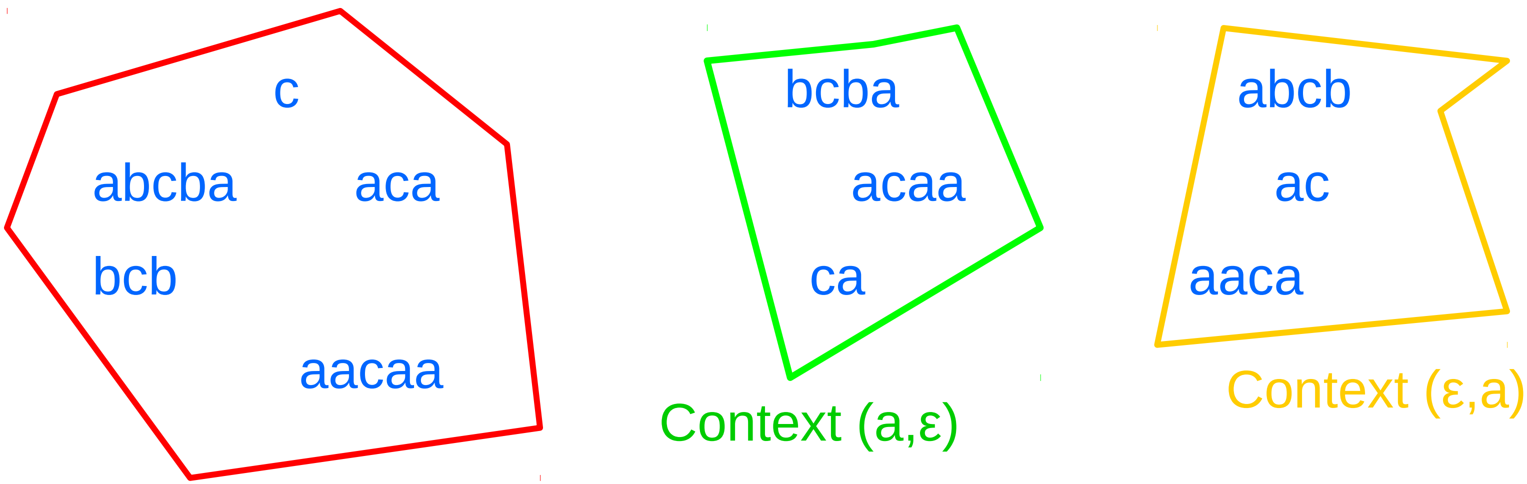
Component (a,ε): (final result)

[ca] → [c] [a], [ca] → [ac] [aa]

[ca] → [b] [cba], [ca] → [bc] [ba], [ca] → [a] [caa]

Running example

LS={c;aca;bcb;abcba;aacaa}



Empty context (ε,ε)

a b ab abcb …

bc ba aac caa

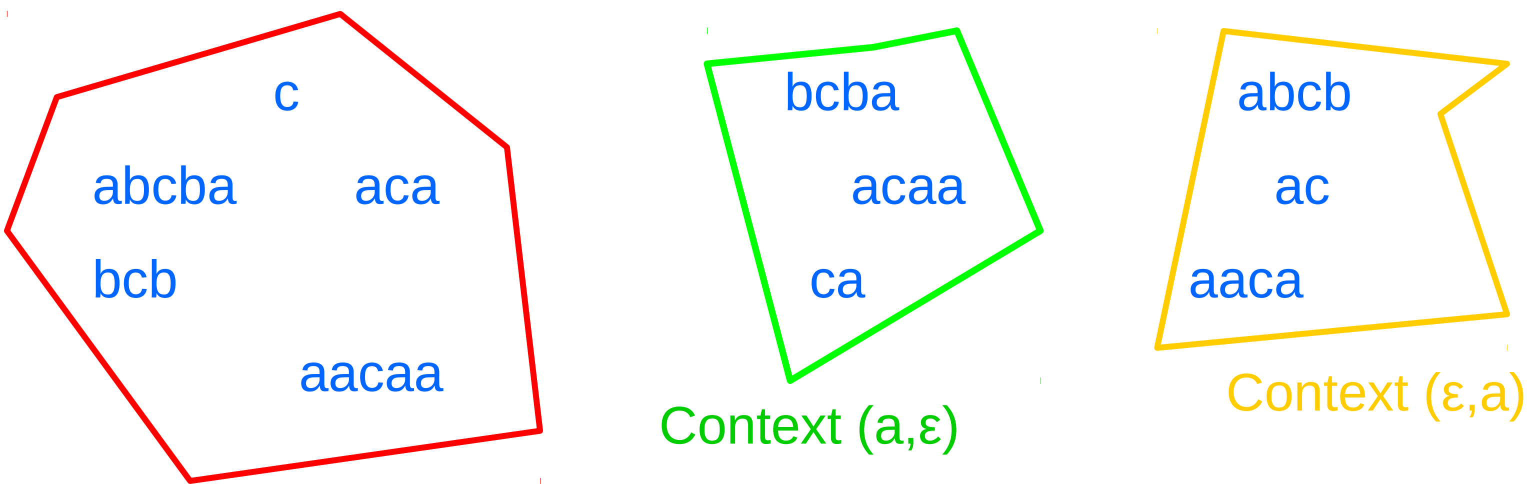
Component (ε,a): (final result)

[ac] → [a] [c], [ac] → [aa] [ca]

[ac] → [ab] [cb], [ac] → [abc] [b], [ac] → [aac] [a]

Running example

LS={c;aca;bcb;abcba;aacaa}



Empty context (ε,ε)

a b ab abcb …

bc ba aac caa

Components of one string:

[ab] → [a][b], [aac] → [a][ac], [aac] → [aa][c], [aa] → [a][a] …

Running example

LS={c;aca;bcb;abcba;aacaa}

Outputed grammar: G=< {a,b,c}, V, P, [c] >

where P = {[a] → a, [b] → b,

[c] → [a][ca] | [ac][a] | c, [ca] → [c][a], [ac] → [a][c],

[c] → [ab][cba] | [abc][ba] | [b][cb] | [bc][b] | [aa][caa] | [aac][aa]

[ca] → [ac][aa] | [b][cba] | [bc][ba] | [a][caa]

[ac] → [aa][ca] | [ab][cb] | [abc][b] | [aac] [a]

[ab] → [a][b]

[aac] → [a][ac] | [aa][c]

[aa] → [a][a]

… }

We can show that this grammar generates the language of palindromes with a center marked.

Learning Result

- The algorithm identifies polynomially in the limit the class of context-free substitutable languages.
- The polynomial bounds are on
  - Computation: it takes a polynomial time in the size of the learning sample to run the algorithm.
  - Data: for each substitutable language, there exists a characteristic sample whose *cardinality* is polynomial in the size of the target.

Substitutable context-free and 0-reversible regular languages

- A regular language is 0-reversible if whenever *uw* and *vw* are in the language then *ux* is in the language iff *vx* is in the language [Angluin, 82].
- Substitutable languages are the (context-free) exact analogue of 0-reversible languages (regular).

Direct Extensions

- This work [Clark & Eyraud, 05, 07] generated different extensions
  - Identification of k-l substitutable languages [Yoshinaka, 08]
  - PAC-learning of unambigous NTS languages [Clark, 06], of subclasses of CFG [Shibata&Yoshinaka, 16]
  - Local substitutable languages [Coste, Garet & Nicolas, 12]
  - Substitutable tree languages [Kasprzik & Yoshinaka, 11]
  - Substitutable graph languages [Eyraud, Janodet, Oates, 12&16]
  - Identification with the help of an oracle of congruential context-free laguages [Clark, 10], conjunctive grammars [Yoshinaka, 15], Parallel Multiple CF grammars [Clark & Yoshinaka, 14]
- Heuristics have preceded theory [Harris, 54], [Brill et al., 90] [Adriaans, 99], [van Zaanen, 00] [Klein & Manning, 02], ...

Substitutable Languages and Natural Languages

- Natural languages are obviously more complex, but substitutable ones can give nice insights into some linguistic disputes.
- Ex.: Auxiliary fronting in polar questions

Sentences like '*Is the man who is hungry ordering dinner?*' are unlikely to be present in a child environment.

However, it has been shown that children are quickly able to identify it as correct and reject the wrong sentence '*Is the man who hungry is ordering dinner?*'

Used as a clue in defence of the theory of innate knowledge of native language.

A (very) simple example

S = { the man who is hungry asked a beer.

the man asked a beer.

the man is hungry.

the man is ordering dinner.

is the man hungry? }

(+) is the man who is hungry ordering dinner?

(-) is the man who hungry is ordering dinner?

- Our algorithm can identifies correct structure from incorrect one without any example on this particular structure in the sample.

Outline

- Introduction
- Learning Substitutable languages
- **Prime congruence classes**
- Extension to tree and graph grammars
- A dual approach
- Conclusion

Congruencial classes

- Recall: Given a language L, [u] is the congruence class of *u*, that is the set of all substrings *v* such that $u \equiv_L v$
- If the language is not regular : an infinite number of congruence classes
- However, substitutable (and others) can be represented by a finite number of them

Prime congruence classes

- Two particular congruence classes:
  - The unit: [ε]
  - The zero: 0 = {u : for all (l,r), lur is not in L}
- A congruence class X is **prime** if it is non-zero and non-unit and for any two congruence classes Y, Z such that X = Y · Z then either Y or Z is the unit.
  If a non-zero non-unit congruence class is not prime then we say it is composite.

Prime congruence classes

- Example: $L=\{a^n cb^n : n \geq 0\}$
  - $[\varepsilon]=\{\varepsilon\}$ and $0=\Sigma^* ba \Sigma^*$: not prime by definition,
  - L : prime (because $L=[c]=[aca]\neq[a][ca]=\{a^n cb^n : n \geq \mathbf{1}\}$)
  - $[a] = \{a\}$ and $[b] =\{b\}$: both prime
  - $[a^i]=\{a^i\}=[a][a^{i-1}]$ with $i>1$: not prime (same for b)
  - $[a^i c] = \{a^{i+j} cb^j : j \geq 0\}$ and $[cb^i] = \{a^j cb^{j+i} : j \geq 0\}$: not prime
- Note: $L = \{ab\}$ has 5 congruence classes: $[a]$, $[b]$, $[ab]$, $[\varepsilon]$ and 0. The first 4 are all singleton sets. $[a]$ and $[b]$ are prime but $[ab] = \{ab\} = [a][b]$, and so L is not prime.

L_{sc}

- L_{sc} : the set of all languages which are substitutable, non-empty, do not contain ε, and have a finite number of prime congruence classes.
- Example: $\{a^n c b^n : n \geq 0\}$
- Counter-example: $L=\{c^i b a^i b : i>0\} \cup \{c^i d e^i d : i>0\}$
  - CF and substitutable
  - But for all i, $C_L(ba^ib) = \{(c^i, \varepsilon)\} = C_L(de^id)$
  - Infinite number of classes $[ba^ib]=[de^id]=\{ba^ib, de^id\}$, each of which is prime.

Prime decomposition

- A prime decomposition of a congruence class X is a finite sequence of one or more prime congruence classes $\alpha = \langle X_1 , \ldots , X_k \rangle$ such that $X = X_1 X_2 \ldots X_k$
- Lemma: Every non-zero non-unit congruence class of a language in L_{sc} has a unique prime factorisation

Correct rules

- Correct production: $[\bar{\alpha}] \rightarrow \alpha$ where α is a sequence of at least 2 primes and $[\bar{\alpha}]$ is a prime congruence class.

  A correct lexical production is one of the form $[a] \rightarrow a$ where $a \in \Sigma$, and $[a]$ is prime.

- Ex: $L = \{a\ n\ cb\ n \mid n \geq 0\}$. Primes: [a], [c], [b].

  The correct lexical productions are the three obvious ones $[a] \rightarrow a$, $[b] \rightarrow b$ and $[c] \rightarrow c$.

  The only correct productions have [c] on the left hand side, and are $[c] \rightarrow [a][c][b]$, $[c] \rightarrow [a][a][c][b][b]$ and so on.

Too long rules

- We say that a sequence of primes α is pleonastic (too long) if $\alpha = \gamma\beta\delta$ for some γ, β, δ, which are sequences of primes, such that $|\gamma| + |\delta| > 0$, $[\beta]$ is a prime, and $|\beta| > 1$
- A rule is too long if its right handside is too long. It is valid otherwise

Canonical grammar

- Lemma: If $L \in L_{sc}$ then there are a finite number of valid productions
- Let L in L_{sc}. Its (unique) canonical grammar $G_*(L)$:
  - Non-terminals: the prime congruence classes of L, together with an additional symbol S (the start symbol).
  - Productions:
    - the single production containing the start symbol: $S \rightarrow \alpha(L)$, where $\alpha(L)$ be the unique prime decomposition of L.
    - All valid productions
    - The production $[a] \rightarrow a$, for each terminal symbol *a* that occurs in the language
- Theorem: the language of $G_*(L)$ is L

Learning result

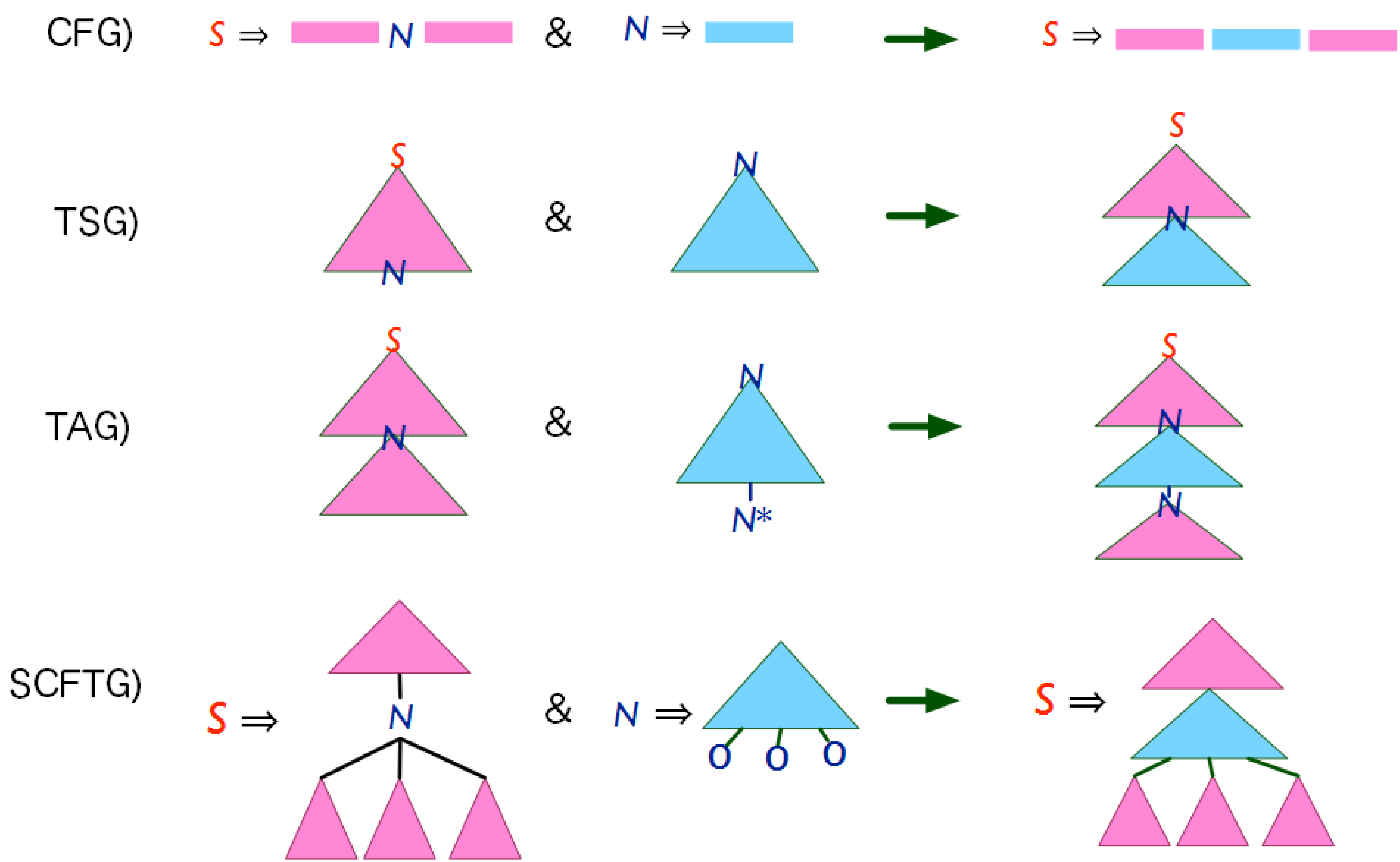
- The previous algorithm can be adapted to output only canonical grammars: it identifies in the limit the class L_{sc}
- Moreover, it learns exactly the target grammar (=language + structure): strong learning
- Corrolary: it learns trees from strings!

Outline

- Introduction
- Learning Substitutable languages
- Prime congruence classes
- **Extension to tree and graph grammars**
- A dual approach
- Conclusion

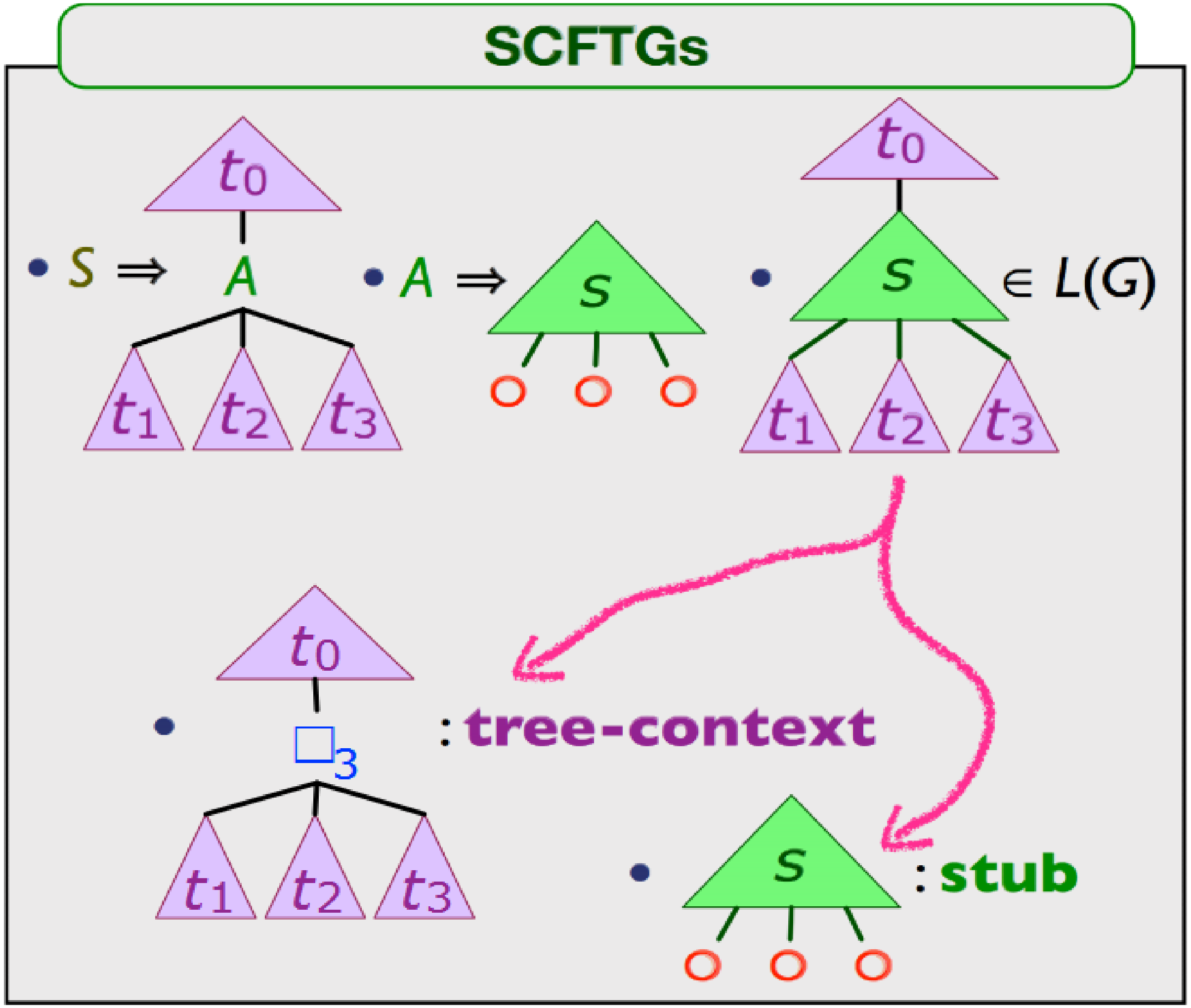
Extension to tree

- Instead of strings the data are trees
- The grammars are Simple Context Free Tree Grammar



Extension to tree

- All we need is the transposition of the notions of context, subtree and a gluing mechanism.



Distributional learning of tree languages

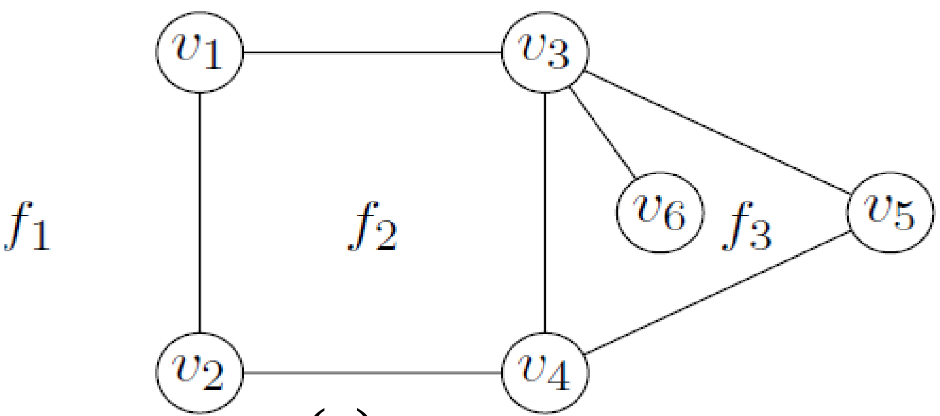
- [Kasprzik & Yoshinaka, 11]:
  - r-substitutable context-free tree languages are efficiently identifiable in the limit from positive tree examples.
  - r-SCFTG with p-finite environment are identifiable from positive presentation using a membership oracle.
  - The algorithms are simple adaptation to trees of the ones for the strings.

Extension to graphs

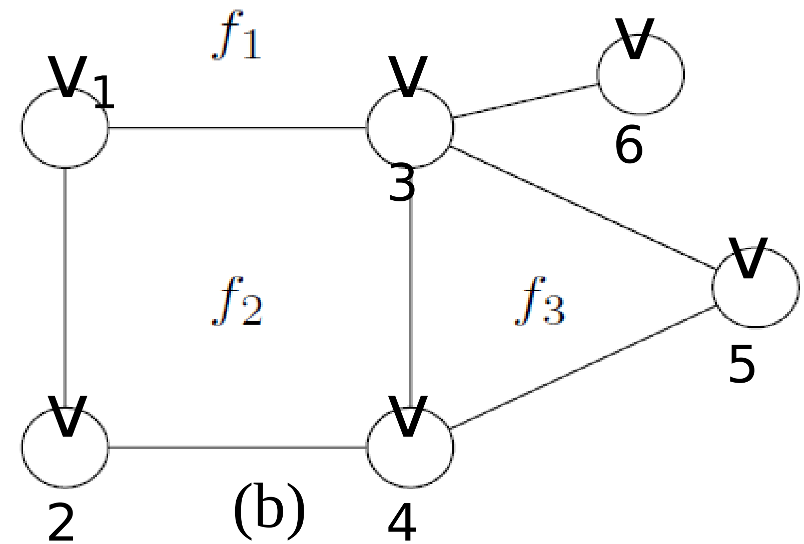
- Again, we need to define what a context is, what a subgraph is, and how to glu them together.
- But we also need to restrict ourselves as general graphs are too complex:
  - We need tractable isomorphism (and sub-isomorphisme)
  - We need a grammar formalism where it is polynomially doable to test whether a given graph is in the language
- Plane graphs are good candidate: polynomial decidable sub-isomorphism.

Plane Graph

- Embeddings of planar graph in the plan:



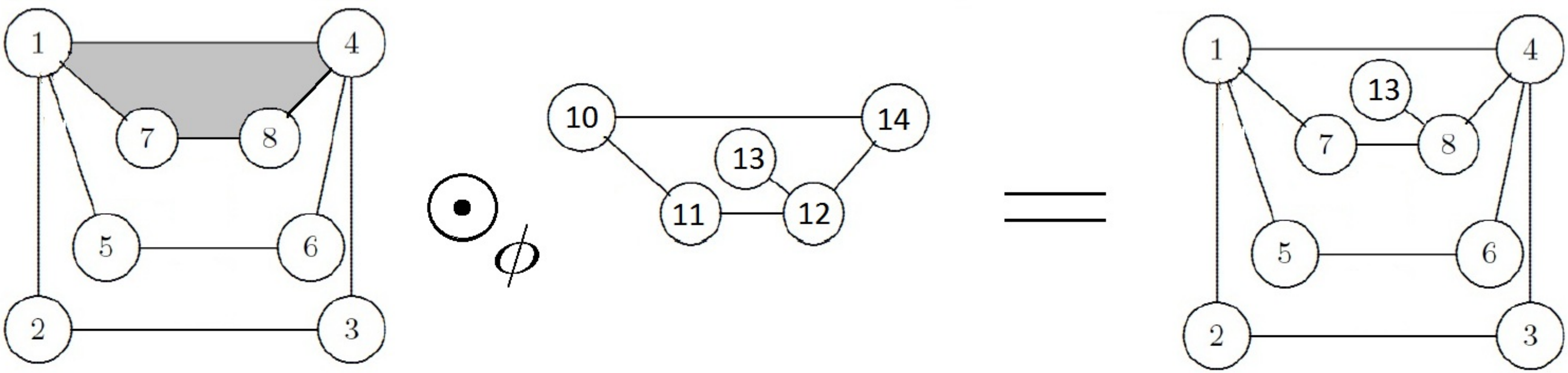
(a)



(b)

- Class of isotopy

Substitutability



- In a substitutable plane graph language, whenever two plane graphs appear in the same context once, they share the same set of contexts.

Learning result

- [Eyraud, Janodet, Oates, 16]: Substitutable plane graph languages are identifiable in the limit from examples of the language

  - Promising research
  - Languages are close under isomorphism.
  - First non-trivial class of graph grammars to be learnable

  - Algorithm is not efficient
  - Hard to extend to more complex kind of graphs

Outline

- Introduction
- Learning Substitutable languages
- Extension to tree and graph grammars
- **A dual approach**
- Conclusion

Distributional Learning

- A new familly of approaches to handle grammar induction:
  - "Observe, model, exploit the relation between substrings and contexts"
- Primal (ex: substitutable): a non-terminal represent a (set of) string that appear in the same contexts: if [u] $\rightarrow^*$ v then $C_L(u) = C_L(v)$
- Dual: a non-terminal represent a (set of) context and generates only strings that appear in this context: if [(l,r)] $\rightarrow^*$ u then $u \in \{w : lwr \text{ in } L\} = S_L(l,r)$

An example

- $L = \{a^n b^n : n \geq 0\}$
- Observation table:

	(ε,ε)	(a,ε)	(ε,b)	(a,b)
ε	yes	no	no	yes
a	no	no	yes	no
b	no	yes	no	no
ab	yes	no	no	yes
aab	no	no	yes	no
abb	no	yes	no	no
aabb	yes	no	no	yes

An example

- $L = \{a^n b^n : n > 0\}$
- Observation table:

	(ε,ε)	(a,ε)	(ε,b)	(a,b)
ε	1	0	0	1
a	0	0	1	0
b	0	1	0	0
ab	1	0	0	1
aab	0	0	1	0
abb	0	1	0	0
aabb	1	0	0	1

An example

- L = $\{a^n b^n : n>0\}$
- Observation table:

	(ε,ε)	(a,ε)	(ε,b)	(a,b)
ε	1	0	0	1
a	0	0	1	0
b	0	1	0	0
ab	1	0	0	1
aab	0	0	1	0
abb	0	1	0	0
aabb	1	0	0	1

- Primal: similar lines (or similar parts of different lines) **may** correspond to the same non-terminal

An example

- $L = \{a^n b^n : n > 0\}$
- Observation table:

	(ε,ε)	(a,ε)	(ε,b)	(a,b)
ε	1	0	0	1
a	0	0	1	0
b	0	1	0	0
ab	1	0	0	1
aab	0	0	1	0
abb	0	1	0	0
aabb	1	0	0	1

- Dual: similar columns **may** correspond to the same non-terminal

An example

- $L = \{a^n b^n : n > 0\}$ but we only "see" the sample S={ab, aabb}
- Observation table:

	(ε,ε)	(a,ε)	(ε,b)	(a,b)
ε	?	?	?	1
a	?	?	1	?
b	?	1	?	?
ab	1	?	?	1
aab	?	?	1	?
abb	?	1	?	?
aabb	1	?	?	?

- Ask an oracle for the missing information

An example

- $L = \{a^n b^n : n>0\}$ but we only “see” the sample S={ab, aabb}
- Real observation table:

	(ε,ε)	(a,ε)	(ε,b)	(a,b)	(aa,ε)	(aab,ε)	(ε,bb)	(ε,abb)	(aa,b)	(a,bb)
ε				+						
a			+					+		+
b		+				+			+	
ab	+			+						
aa							+			
bb					+					
aab			+							
abb		+								
aabb	+									

An example

- L = $\{a^n b^n : n>0\}$ but we only “see” the sample S={ab, aabb}
- Real observation table:

Context set

Kernel

	(ε,ε)	(a,ε)	(ε,b)	(a,b)	(aa,ε)	(aab,ε)	(ε,bb)	(ε,abb)	(aa,b)	(a,bb)
ε				+						
a			+					+		+
b		+				+			+	
ab	+			+						
aa							+			
bb					+					
aab			+							
abb		+								
aabb	+									

Learning principle

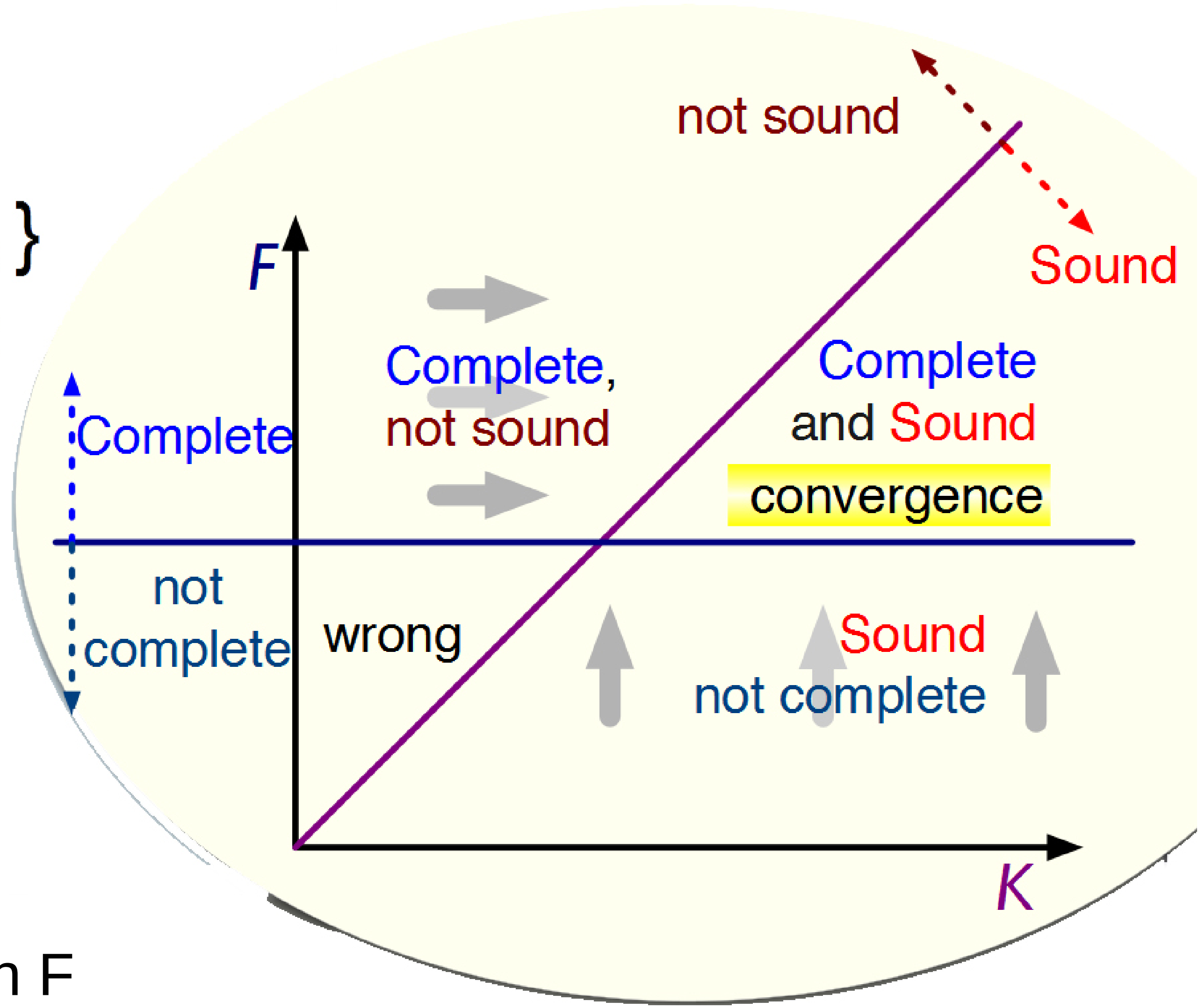
- Problem: if the language is not regular, there exists a non-finite number of syntactic congruence classes.
- What we need is to restrict ourselves to classes of languages that are representable by finitely many congruence classes
- Then we may observe these classes in the completed observable table constructed from a finite sample:
  - Either by considering similar rows (primal)
  - Or by considering similar column (dual)

Finite Context Property (dual)

- A CFG $G = < \Sigma, N, S, P >$ has the (one) Finite Context Property iff every $A \in N$ admits a characterizing context (l,r) such that $S_L(l,r)=\{v: N \rightarrow^* v\}$
- Examples: all regular languages, parenthesis languages, ...
- With enough data, one column of the observation table corresponds exactly to each non-terminal of the target grammar.
  - Adding new columns add new non-terminals (thus new rules)
  - Adding new lines remove incorrect rules

Learning in the dual

Let $D := K := F := \emptyset$;
For $n = 1,2,3,\ldots$
  let $D := \{u_1, u_2, \ldots, u_n\}$
  If $D \not\subseteq L(G(F,K))$ then
    let $F :=$ Con(D)
  End if
  let $K :=$ Sub(D)
  output $G(F,K)$
End for



Grammar creation:
Non-terminals: [(l,r)], (l,r) in F
Rules: [(l,r)] → a if *lar* in L
[(l,r)] → [(l_1,r_1)][(l_2,r_2)] if for all w_1, w_2 in K s.t. $l_1w_1r_1$ and $l_2w_2r_2$ in L we have lw_1w_2r in L

Learning results

- [Clark, 10 ; Yoshinaka, 11] 1-FCP (and 1-FKP) classes are identifiable in the limit from examples using a membership query with
  - An update time polynomial in the size of the sample
  - An number of queries polynomial in the size of the target grammar.
- Extended to k-FCP (and k-Finite Kernel Property and k-Finite Distributional Property where each non-terminal has either a characteristic context or characteristic string).
- Extended to other classes of grammars: multiple CFG and parallel CFG.
- Extended to PAC-learning results

Outline

- Introduction
- Learning Substitutable languages
- Extension to tree and graph grammars
- A dual approach
- **Conclusion**

Overall results (in 2012...)

		String formalisms		Tree form.	Graph form.
		CFG	MCFG	SCFTG	PGG
Primal	Substitutable	Clark & Eyraud '05,'07	Yoshinaka '09	Kasprzik & Yoshinaka '11	Eyraud, Janodet & Oates '12
	Congruential	Clark '10	Yoshinaka & Clark '10		
	Finite Kernel Property	Yoshinaka'11	Yoshinaka'10	Kasprzik & Yoshinaka '11	
Dual	Context-deterministic	Shirakawa & Yokomori '93			
	Finite Context Property	Clark, Eyraud & Habrard '08 (CBFG) Clark'10		Kasprzik & Yoshinaka '11	

Next steps

- Restrictions rely on a class of grammars (to make sure the language is representable by a finite number of congruence classes).
- Use of a membership oracle (for the more complex classes).
- Not practical that way but nice proof of concept.

MISC. citations

- John Myhill, 1950, commenting on Bar-Hillel

I shall call a system regular if the following holds for all expressions μ,ν and all wffs φ,ψ each of which contains an occurrence of ν: If the result of writing μ for some occurrence of ν in φ is a wff, so is the result of writing μ for any occurrence of ν in ψ. Nearly all formal systems so far constructed are regular; ordinary word-languages are conspicuously not so.

- Noam Chomsky review of Greenberg (1959)

Let us say that two units A and B are substitutable1 if there are expressions X and Y such that XAY and XBY are sentences of L; substitutable2 if whenever XAY is a sentence of L then so is XBY and whenever XBY is a sentence of L so is XAY (i.e. A and B are completely mutually substitutable). These are the simplest and basic notions. (footnote 3. They are discussed by R. Carnap in The Logical Syntax of Language, 1934)

String Rewriting Rule

- Introduced in 1914 by Alex Thue.
- A string rewriting rule replaces a substring of a string by another substring.
- Example: ab $\rightarrow$ ε

This rule replaces substings ab by ε, i.e. it erases substrings ab.

a<u>ab</u>bab $\rightarrow$ ab<u>ab</u> $\rightarrow$ <u>ab</u> $\rightarrow$ ε

Running Example

- LS={c;aca;bcb;abcba;aacaa} (palindromes with a center marked)

c bcba abcb

abcba aca acaa ac

bcb ca aaca

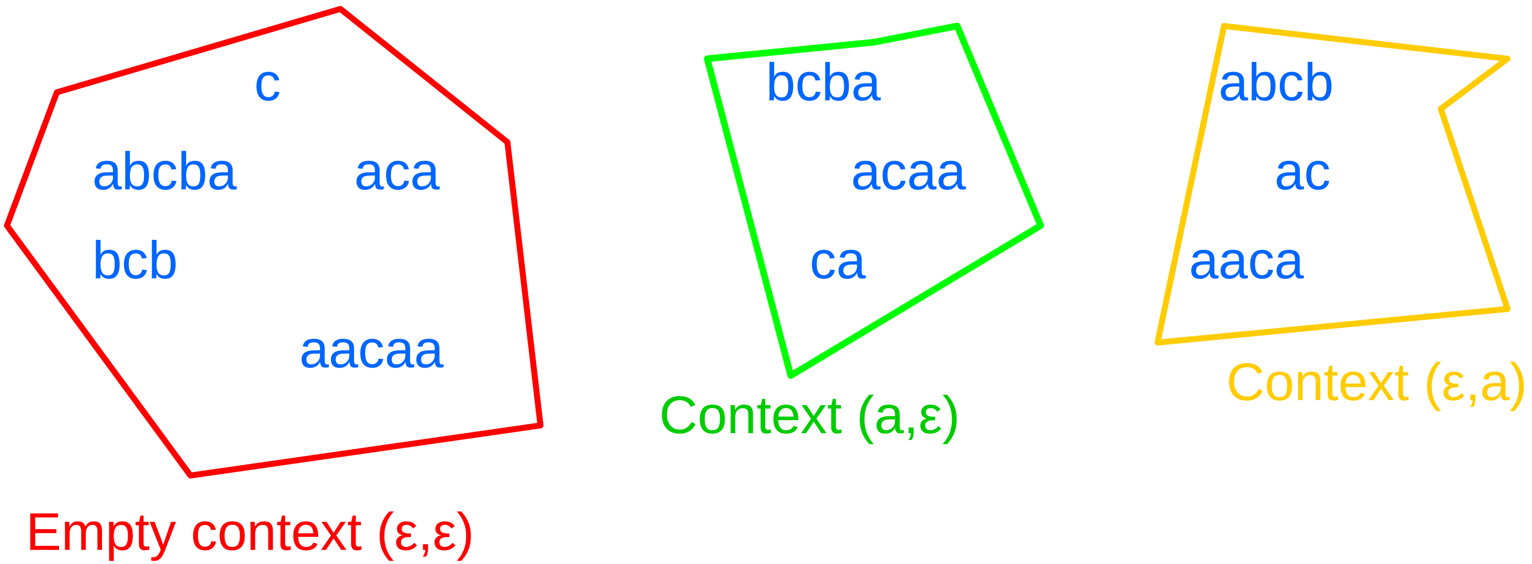
aacaa

a b ab abcb …

bc ba aac caa

Running Example

- LS={c;aca;bcb;abcba;aacaa}

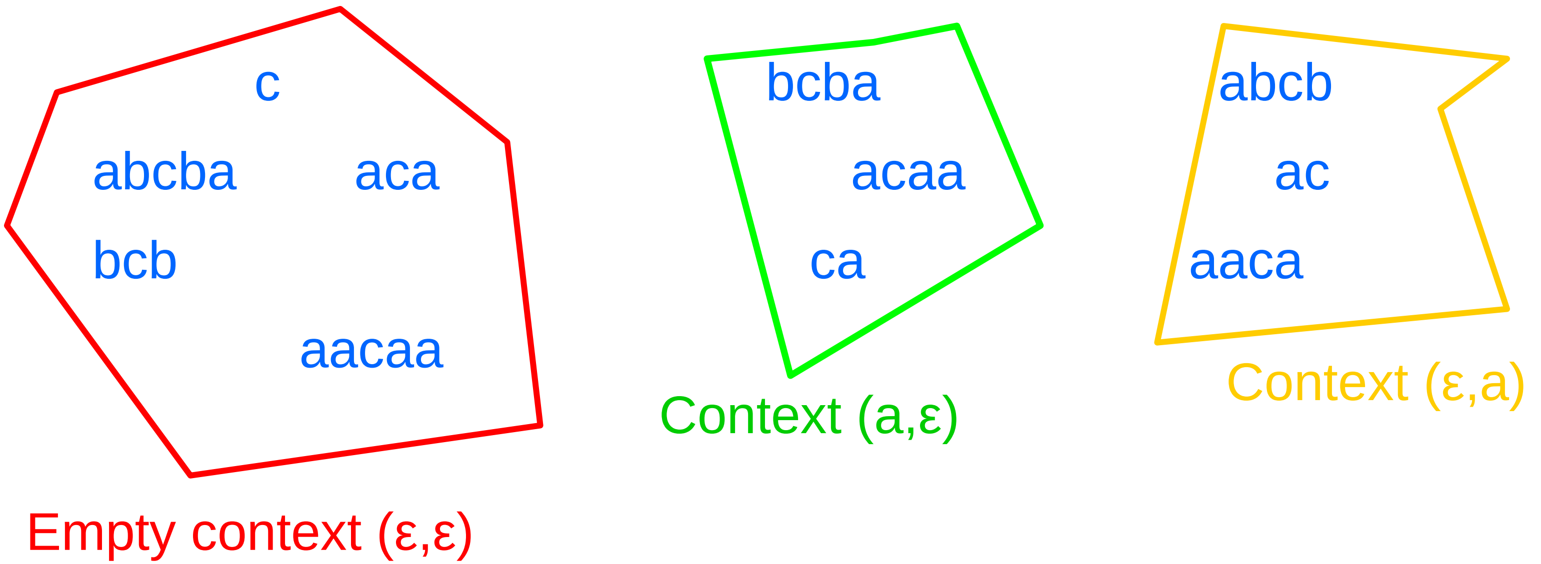


a b ab abcb …

bc ba aac caa

Running Example

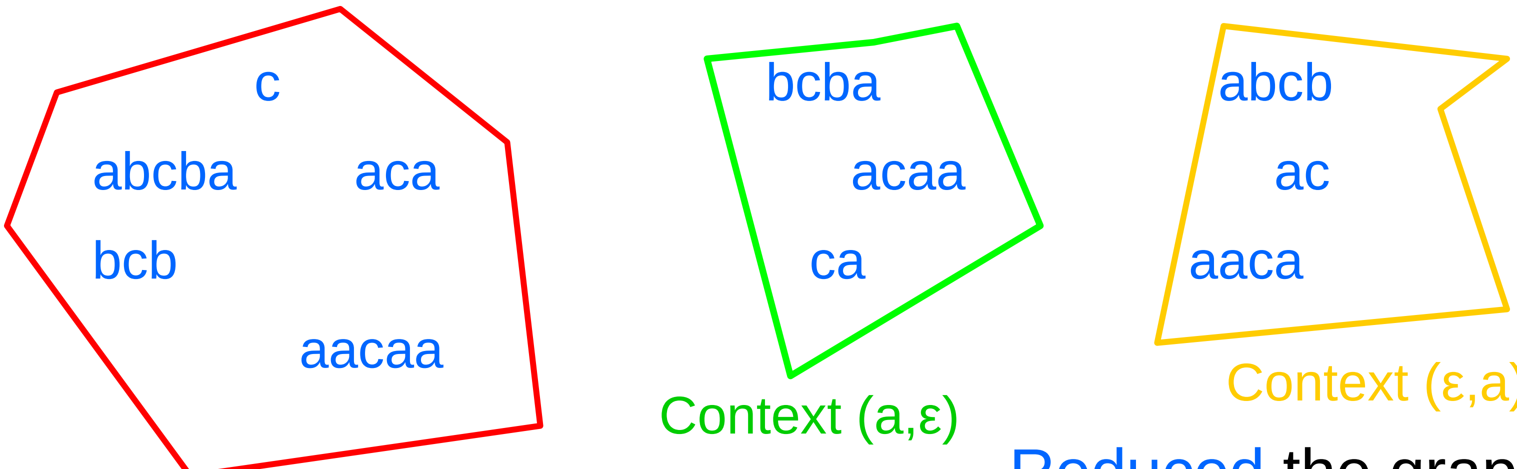
- LS={c;aca;bcb;abcba;aacaa}



Erase component made of a unique element.

Running Example

- LS={c;aca;bcb;abcba;aacaa}



Empty context (ε,ε)

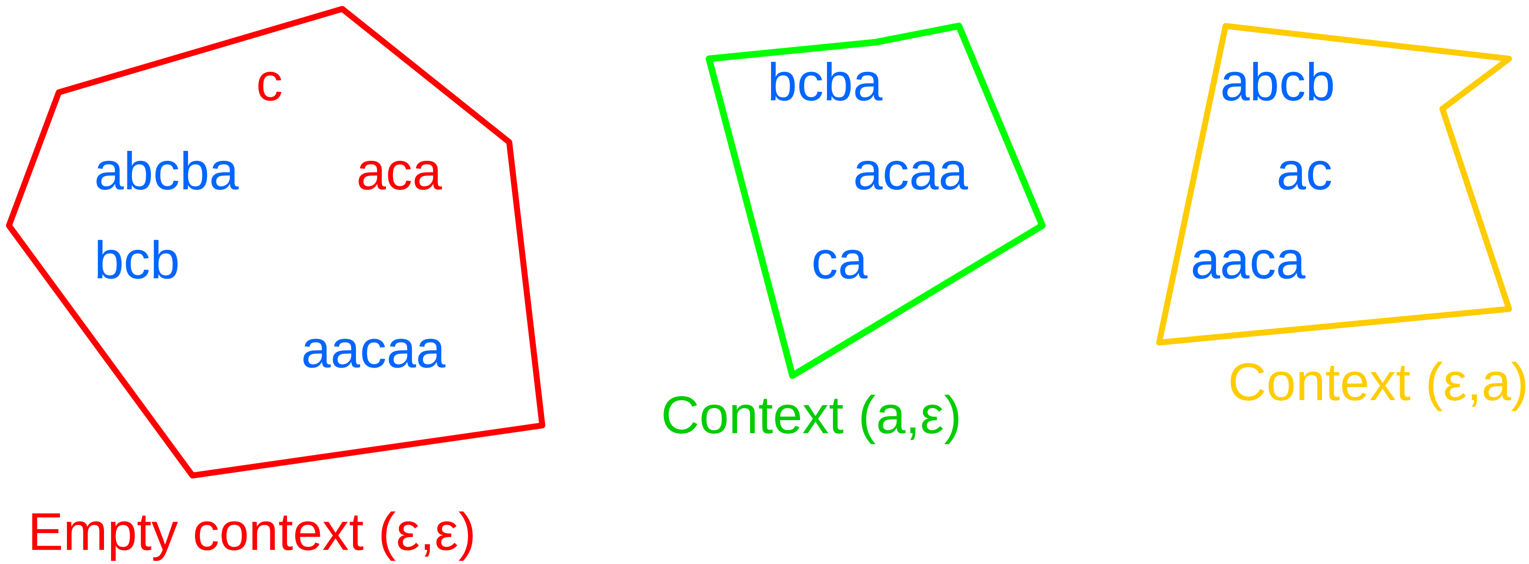
Reduced the graph:

If u and v are in the same component and u > v

Then replace every lur by lvr

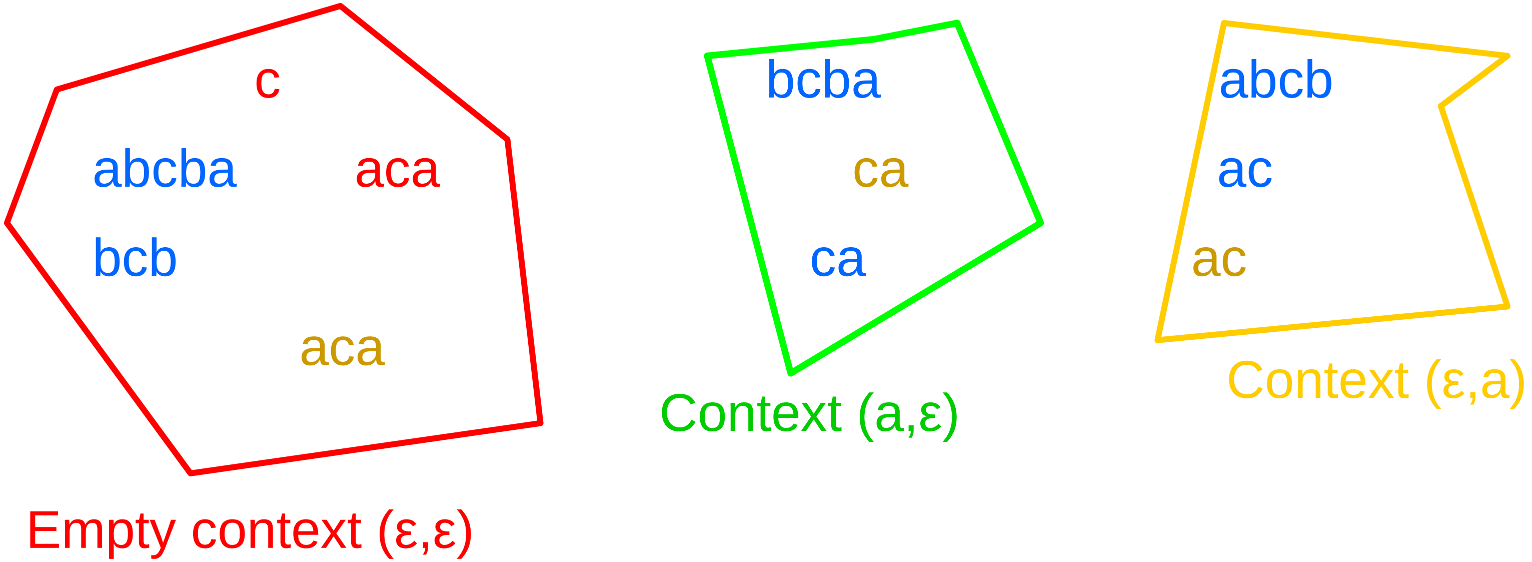
Running Example

- LS={c;aca;bcb;abcba;aacaa}



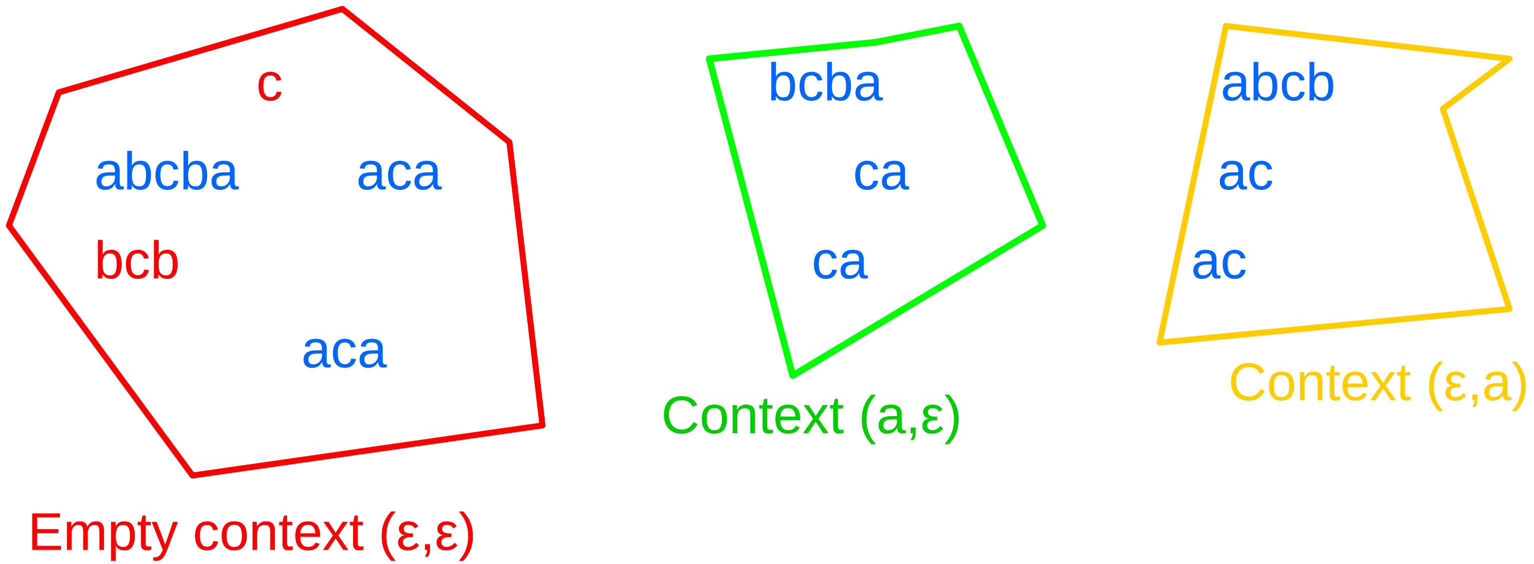
Running Example

- LS={c;aca;bcb;abcba;aacaa}



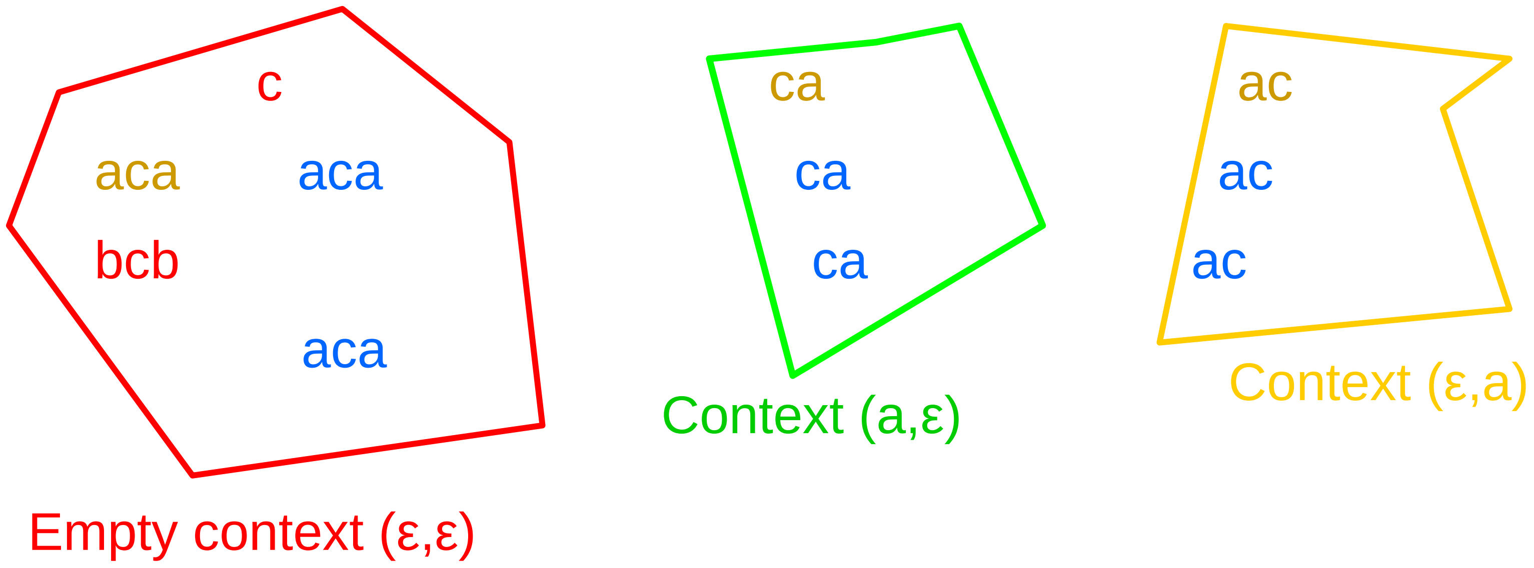
Running Example

- LS={c;aca;bcb;abcba;aacaa}



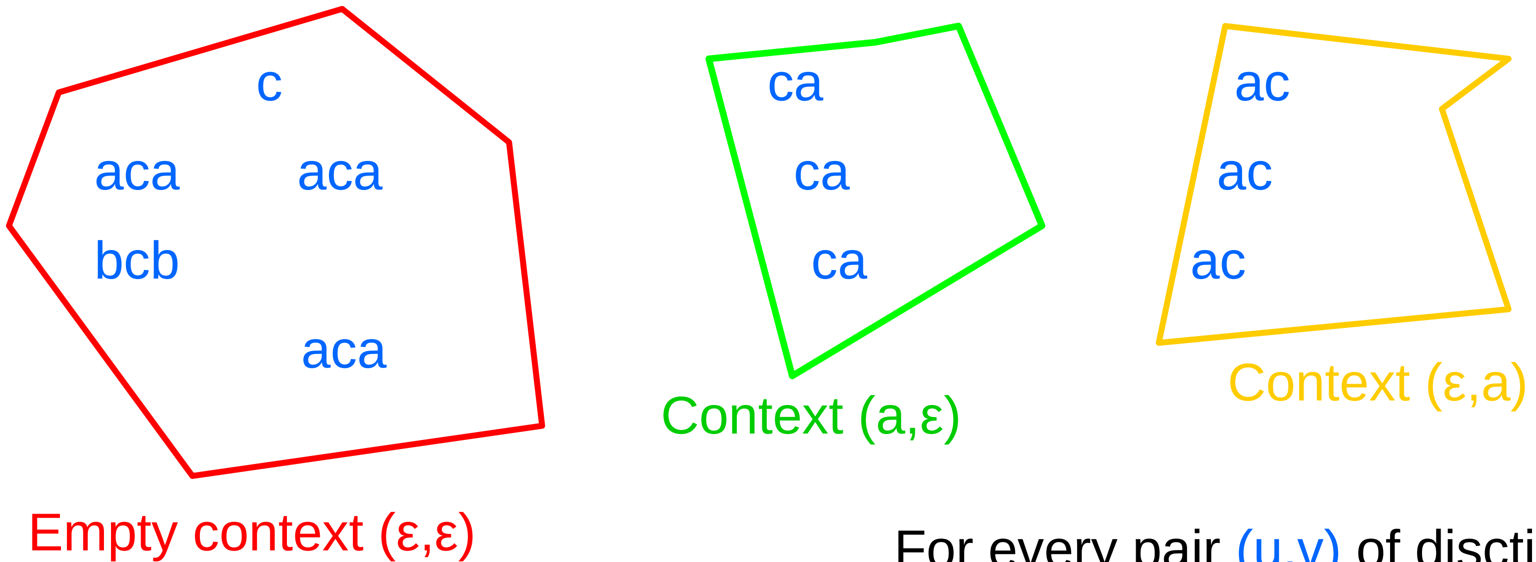
Running Example

- LS={c;aca;bcb;abcba;aacaa}



Running Example

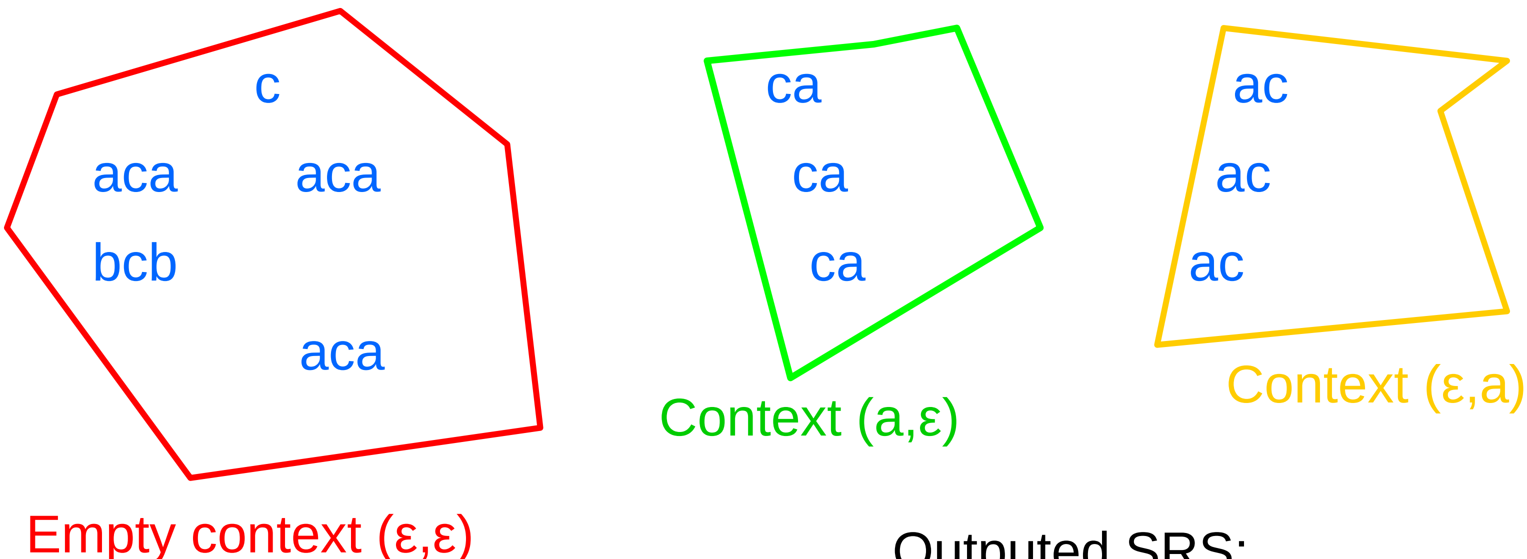
- LS={c;aca;bcb;abcba;aacaa}



For every pair (u,v) of disctinct substrings in the same component, creat a rule u → v (u > v).

Running Example

- LS={c;aca;bcb;abcba;aacaa}



Outputed SRS:

< {aca → c; bcb → c}, {c} >

Learning in the Primal

Let $D := K := F := \emptyset$;

For $n = 1,2,3,\ldots$

  let $D := \{u_1, u_2, \ldots, u_n\}$

  If $D \not\subseteq L(\mathrm{G}(K,F))$ then

    let $K := \mathrm{Sub}(D)$

  End if

  let $F := \mathrm{Con}(D)$

  output $\mathrm{G}(K,F)$

End for

Finite Kernel Property (primal)

- A CFG $G = < \Sigma, N, S, P >$ has the (one) Finite Kernel Property iff every $A \in N$ admits a characterizing string u such that $C_L(u)=\{(l,r):\exists v, N\rightarrow^* v, lvr\in L(G)\}$
- Examples: all regular languages, parenthesis languages, ...
- With enough data, one line of the observation table corresponds exactly to each non-terminal of the target grammar.
  - Adding new lines add new non-terminals (thus new rules)
  - Adding new columns remove incorrect rules