

# Learning context-free grammars

Colin de la Higuera  
University of Nantes



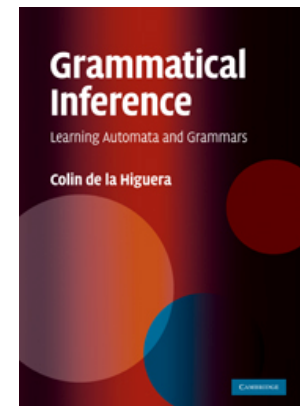


# Acknowledgements

- Laurent Miclet, Jose Oncina and Tim Oates for previous versions of these slides.
- Rafael Carrasco, Paco Casacuberta, Rémi Eyraud, Philippe Ezequel, Henning Fernau, Thierry Murgue, Franck Thollard, Enrique Vidal, Frédéric Tantini,...
- List is necessarily incomplete. Excuses to those that have been forgotten.

<http://pagesperso.lina.univ-nantes.fr/~cdlh/slides/>

Chapter 15





# Outline

1. Context-free grammars
2. Paradigms and theorems
3. Some heuristics
4. Applications
5. Conclusions

# 1. Context free grammars



# What is a context free grammar?



A 4-tuple  $(\Sigma, S, V, P)$  such that:

- $\Sigma$  is the alphabet
- $V$  is a finite set of non terminals
- $S$  is the start symbol
- $P \in V \times (V \cup \Sigma)^*$  is a finite set of rules

# Example



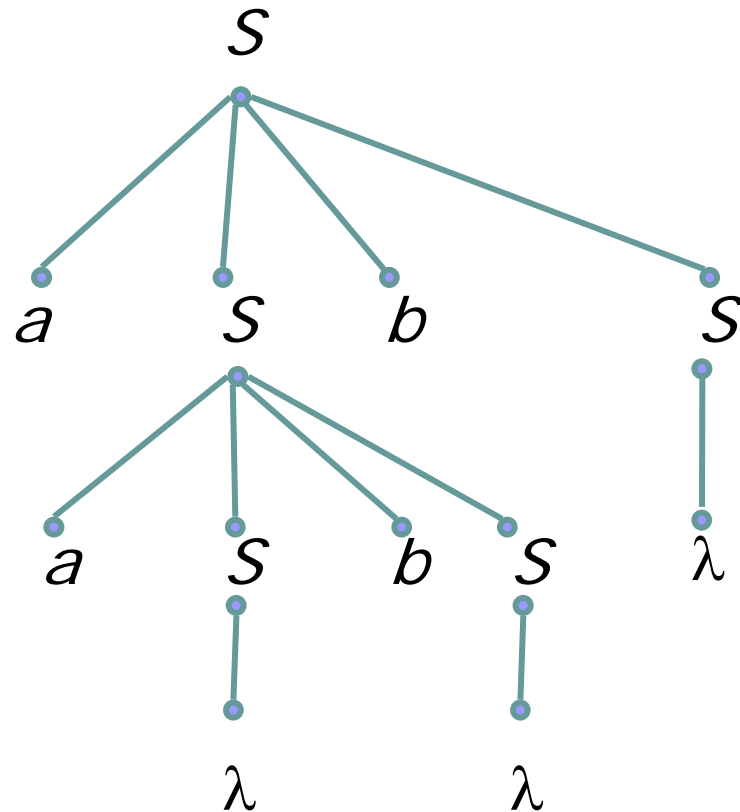
## The Dyck<sub>1</sub> grammar

- $(\Sigma, S, V, P)$
- $\Sigma = \{a, b\}$
- $V = \{S\}$
- $P = \{S \rightarrow aSbS, S \rightarrow \lambda\}$

# Derivations and derivation trees



$S \rightarrow aSbS$   
 $\rightarrow aaSbSbS$   
 $\rightarrow aabSbS$   
 $\rightarrow aabbS$   
 $\rightarrow aabb$



# Why learn context free grammars (CFG)?

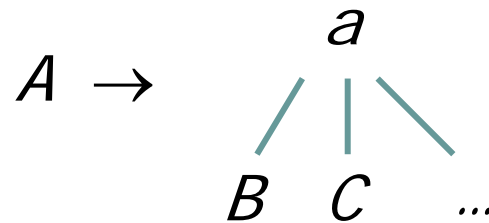


- More expressive than regular grammars: all regular languages are context-free
- next step up on the Chomsky hierarchy
- allows to define more precise and expressive structure



# Tree Grammars

Similar to *CFG* but the rules have the shape:

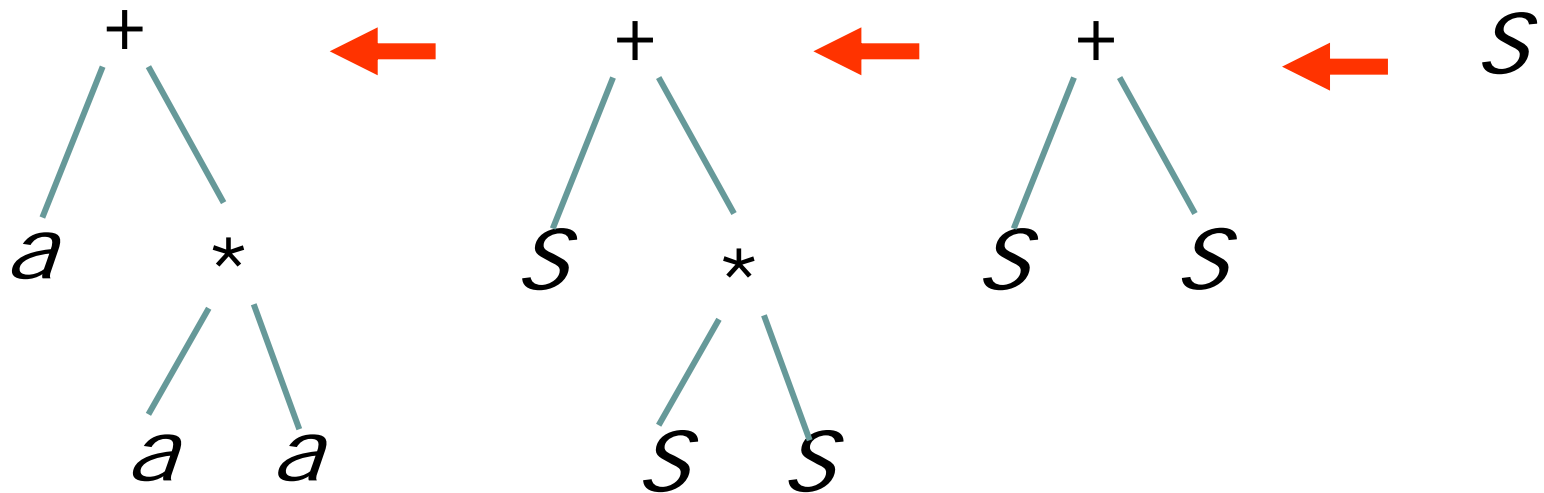




# Example

Let  $P = \{ S \rightarrow a,$

$S \rightarrow +, S \rightarrow * \}$



# Skeletons and tree grammars



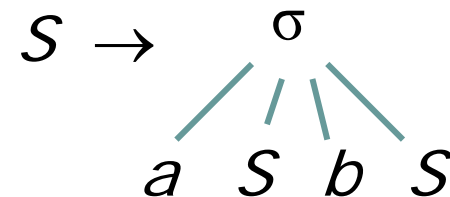
Any context free grammar can be transformed into a tree grammar that produces skeletons

A tree automaton:

$$S \rightarrow aSbS$$



$$S \rightarrow \lambda$$



$$S \rightarrow \sigma$$



# Theory

It is made harder by the hardness of various problems over  $CF$  grammars:

- Expansiveness
- Ambiguity
- Undecidability of the equivalence problem



# Expansiveness

$$T_0 \rightarrow T_1 T_1^+ \dots$$

$$T_1 \rightarrow T_2 T_2^+ \dots$$

⋮

$$T_n \rightarrow a$$

$$L_a(T_0) = \{a^{2^m}\}$$

String  $a^{2^n}$  is  
probable but  
very long.  
What about  
complexity?



# Ambiguity

- $S \rightarrow S^*S, S \rightarrow a, S \rightarrow b$
- Where does  $a^*b^*b$  come from?
- Do we really want to learn languages?

# Equivalence problem revisited



- When trying to learn a grammar, are we not attempting to find some normal form?
- Does that not seem difficult when the equivalence problem is undecidable?

# Practical issues: parsing



- CYK, complexity:  $O(n^3)$
- Earley, complexity:  $O(n^3)$
- Valiant, complexity:  $O(n^{2.81})$

# 2. Paradigms and results





# Identification in the limit

- The examples are provided in sequence
- For each new example the learning algorithm must provide a hypothesis
- Success if the sequence of hypotheses converges



# Learning from positives examples

- It is impossible to identify in the limit any super-finite class of language from positive examples only (Gold, 67)
- A super-finite class of languages includes:
  - all the finite languages and
  - at least an infinite language

# What can we do?



- Use of some additional help:
  - negative data
  - access to oracles
  - knowledge of the structure
  - belief there is a structured distribution
- Avoid super-finite classes of languages
- Combinations of both ideas

# Can we learn in the limit context-free languages from ...

*Complexity does not matter!*





## ... positive examples?

NO (Gold, 67)

- the class of context free languages is super-finite

# ... positive and negative examples?



YES (Gold, 67)

- by an enumeration procedure:
  1. order all the *CFG* in a list
  2. search the list and return the first grammar consistent with the data

Complexity is  $O(|V| |P| |V|)$



## ... skeletons of positive examples?

NO, as a consequence of (Gold, 67)

- the class of the tree languages that represent skeletons is super-finite
- YES, (Sakakibara, 92)
  - if the skeletons come from a reversible context free grammar (normal form)



# Important!

- In the first case we want to identify a grammar that matches a set of trees
- In the second case we will need trees that conform to the unknown grammar
  
- Crucial question... what are we learning?  
Grammars or languages?

# Reversible context free languages



- Deterministic bottom-up / top-down
- $A \rightarrow \alpha$  and  $B \rightarrow \alpha \Rightarrow A=B$
- $A \rightarrow \alpha B \beta$  and  $A \rightarrow \alpha C \beta \Rightarrow B=C$
- Algorithm
  - Build the grammar that only accepts the sample
  - Merge pair of non terminals that violate some of the previous rules



## ... queries?

Not well known!

- The most used queries are:
  - Membership queries
  - Equivalence queries

*Note that an equivalence query might be non computable.*

Queries are usually introduced to deal with complexity issues...

# Polynomial identification

There are several definitions:

- (Pitt, 89) and (Yokomori, 91)
  - Polynomial update time
  - Polynomial number of hypothesis changes
  - Polynomial number of implicit prediction errors (Yokomori)
- In polynomial time and data (cdlh, 97)
  - Polynomial update time
  - Polynomial characteristic sample

# Can we learn polynomially in the limit context free languages from ...



# ... positive and negative examples?



NO,

- if usual cryptographic rules apply  
(Pitt and Warmuth, 88)

In the *polynomial time and data* framework, context-free and linear\* languages are not identifiable (cdlh, 97)

\* the rules have the shape:  $A \rightarrow vBw, A \rightarrow v$



## ... positive skeletons?

YES,

- provided that the grammar is written in *reversible* normal form (Sakakibara, 92)
- even though the regular languages are not identifiable from positive skeletons!!!

# ... positive and negative skeletons?



YES,

- it is a special case of learning tree grammars from positive and negative examples  
(García & Oncina, 93 )

... positive skeletons and negative examples?



YES

- with a slight modification of the previous algorithm

... the knowledge that the distribution is given by a stochastic context-free grammar?



- There is not even a sensible definition of what this can be
- The number of examples should be very large in order to have information about the existence of a rule with a very low probability

# ... And if we have the grammar also?



Not known!

There are some heuristics...

- Expectation maximization
- Inside-Outside



... queries?

YES,

- provided the grammar is written in reversible normal form

(Sakakibara, 90)

NO,

in general

**Can we learn polynomially  
in the limit some  
subclasses of context-free  
languages from ...**



## ... positive examples?

- **Subclasses** of even linear languages  
 (Takada, 88), (Sempere & García, 94),  
 (Mäkinen, 96)
  - Rules with shape  $A \rightarrow aBb + a + \lambda$
  - The trick is to transform  $A \rightarrow aBb$  into  $A \rightarrow [ab]B$ , then we have a regular language
- Very simple grammars (Yokomori, 91)
  - Rules with shape  $A \rightarrow a + aB + aBC$
  - Globally deterministic in "a"

# ... positive and negative examples?



- Even Linear Languages

(Takada, 88), (Mäkinen, 96), (Sempere & García, 94)

- Same trick as in the previous slide

- Linear Deterministic Languages

(de la Higuera & Oncina, 02)

- Rules of shape  $A \rightarrow aBv + \lambda$
- $A \rightarrow aBv$  rules deterministic in  $a$



## ... positive skeletons?

Some classes transposed from regular languages to tree languages and then to context free

- $k$ -testable tree languages (Knuutila, 93)
- (Fernau, 02)
- (Ishizaka, 89)
- (Yokomori, 91)



# ... a distribution?

## Stochastic Deterministic Linear Languages

(de la Higuera & Oncina, 03)

- Identification of the structure
- Polynomial update time



# ... Queries?

## Simple Deterministic Languages

(Ishizaka, 89)

- Grammar:
  - rules with shape  $A \rightarrow a + aB + aBC$
  - deterministic in "d"
- Queries:
  - membership queries
  - extended equivalence queries

# Can we PAC learn context free languages from ...



... positive and negative examples?



NO,

- If usual cryptographic rules apply:  
(Kearns & Valiant, 94)



... positive examples?

NO,

- a consequence of the previous result



## ... positive skeletons?

NO,

- because regular languages cannot be learned ...

# ... positive skeletons and negative examples?



probably NO,

- if usual cryptographic rules apply

It should be a direct consequence of  
(Kearns & Valiant, 94)

# 3 “Pragmatic” Learning

Many different ideas:

Incremental learning

*MDL* principle

Genetic/evolutionary algorithms

Reversing the parser

Tree automata learning

Merging





## 3.1 SEQUITUR

(<http://sequitur.info/>)

(Neville Manning & Witten, 97)

Idea: construct a *CF* grammar from a very long string  $w$ , such that  $L(G)=\{w\}$

- No generalization
- Linear time (+/-)
- Good compression rates



# Principle

The grammar with respect to the string:


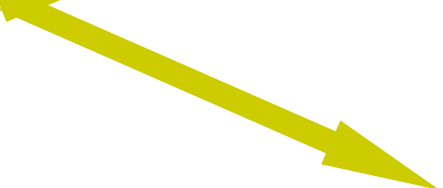
- Each rule has to be used at least twice
- There can be no sub-string of length 2 that appears twice



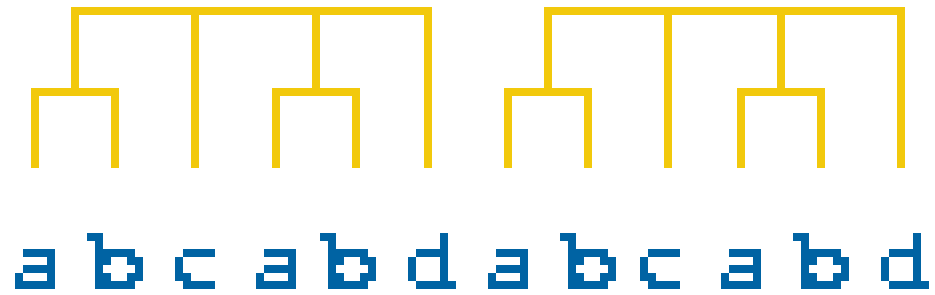
# Examples

$S \rightarrow abcdbc$    $S \rightarrow aAdA$   
 $A \rightarrow bc$

---

$S \rightarrow aabaaab$    $S \rightarrow AaA$   
 $A \rightarrow aab$   
  $S \rightarrow AbAab$   
 $A \rightarrow aa$

*abcabdabcabd*





In the beginning, God created the heavens and the earth.

And the earth was without form, and void; and darkness was upon the face of the deep. And the Spirit of God moved upon the face of the waters.

And God said, Let there be light: and there was light.

And God saw the light, that it was good: and God divided the light from the darkness.

And God called the light Day, and the darkness he called Night. And the evening and the morning were the first day.

And God said, Let there be a firmament in the midst of the waters, and let it divide the waters from the waters.

And God made the firmament, and divided the waters which were under the firmament from the waters which were above the firmament: and it was so.

And God called the firmament Heaven. And the evening and the morning were the second day.



In the beginning, God created the heavens and the earth. And the earth was without form, and void; and darkness was upon the face of the deep. And the Spirit of God moved upon the face of the waters. And God said, Let there be light: and there was light. And God saw the light, that it was good: and God divided the light



# Sequitur options

- appending a symbol to rule  $S$
- using an existing rule
- creating a new rule
- and deleting a rule



# Results

On text:

- 2.82 *bpc*
- *compress* 3.46 *bpc*
- *gzip* 3.25 *bpc*
- PPMC 2.52 *bpc*



## 3.2 Using a simplicity bias

(Langley & Stromsten, 00)

Based on algorithm **GRIDS** (Wolff, 82)

Main characteristics:

- *MDL* principle
- Not characterizable
- Not tested on large benchmarks

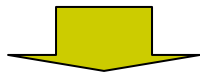


# Two learning operators

Creation of non terminals and rules

$NP \rightarrow ART \textit{ADJ} NOUN$

$NP \rightarrow ART \textit{ADJ} \textit{ADJ} NOUN$



$NP \rightarrow ART AP1$

$NP \rightarrow ART \textit{ADJ} AP1$

$AP1 \rightarrow \textit{ADJ} NOUN$

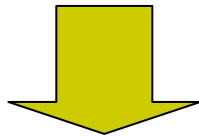
# Merging two non terminals

*NP* → *ART AP1*

*NP* → *ART AP2*

*AP1* → *ADJ NOUN*

*AP2* → *ADJ AP1*



*NP* → *ART AP1*

*AP1* → *ADJ NOUN*

*AP1* → *ADJ AP1*



- Scoring function: *MDL principle*:

$$|G| + \sum_{w \in T} |d(w)|$$

- Algorithm:
  - find best **merge** that improves current grammar
  - if no such merge exists, find best **creation**
  - halt when no improvement



# Results

- On subsets of English grammars (15 rules, 8 non terminals, 9 terminals): 120 sentences to converge
- on  $(ab)^*$ : all (15) strings of length  $\leq 30$
- on  $Dyck_1$ : all (65) strings of length  $\leq 12$



## 3.3 Context free grammar induction with genetic/evolutionary algorithms

- (Wyard, 91)
- (Dupont, 94)
- (Kammeyer & Belew, 96)
- (Sakakibara & Kondo, 99)
- (Sakakibara & Muramatsu, 00)



# Main issue

- Encoding a context free grammar as a string such that after crossovers and mutations the string is still a grammar...
- Some ideas:
  - Fill up with junk *dna*: (Kammeyer & Belew, 96)
  - A grammar is a partition. Encode the partition



$\{1,2,6\} \{3\} \{4,5\} \{7,9\} \{8\}$   $\longrightarrow$  112331454



$\{1,2,6\} \{3\} \{4\} \{5,7,9\} \{8\}$   $\longleftarrow$  112341454



## 3.4 Reversible CFGs

**Definition:** A context-free grammar is *reversible* if the following two conditions hold.



# The First Condition

## *Context-free grammars*

If there exist productions of the form  
 $A \rightarrow aBb$  and  $A \rightarrow aCb$  then  $B = C$ .

## *Regular grammars*

If there exist productions of the  
form  $A \rightarrow aB$  and  $A \rightarrow aC$  then  $B = C$ .



## The Second Condition

### *Context-free grammars*

If there exist productions of the form  
 $A \rightarrow a$  and  $B \rightarrow a$  then  $A = B$

### *Regular grammars*

If there exist productions of the form  
 $A \rightarrow a$  and  $B \rightarrow a$  and there exists a  
string  $v$  that is a  $k$ -leader of both  $A$   
and  $B$  then  $A = B$



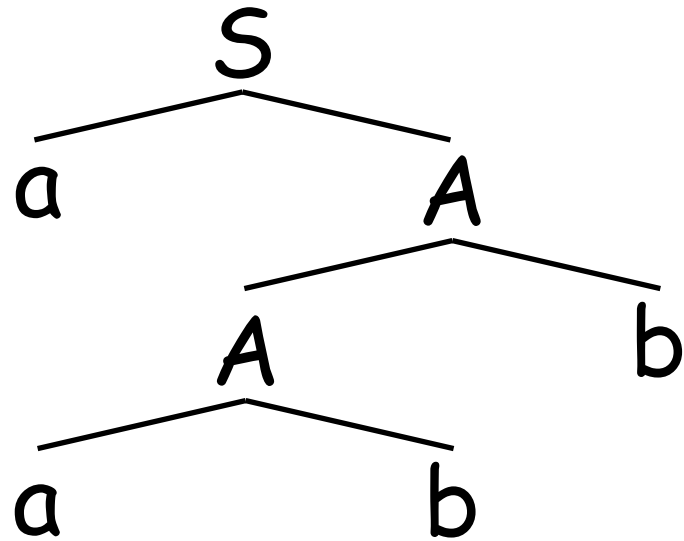
# Sakakibara's K-RI Algorithm

**Given:** a sample of strings  $S$  in the language of some reversible context-free grammar and their unlabeled derivation trees

**Identify:** the smallest reversible context-free language containing  $S$

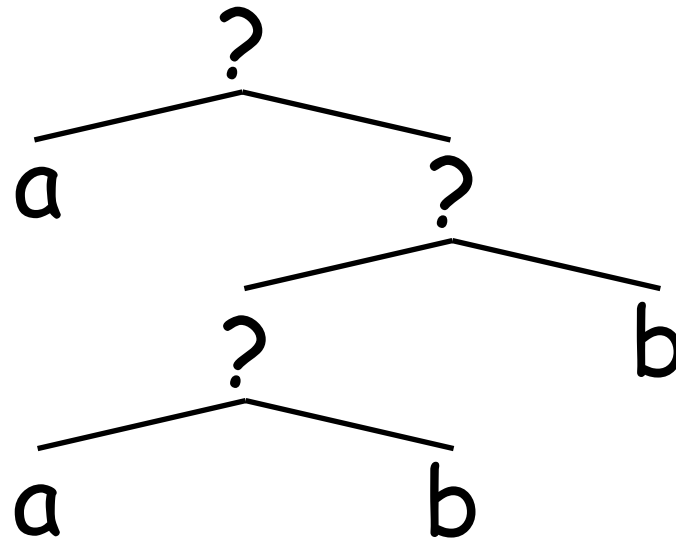


# Labeled Derivation Trees



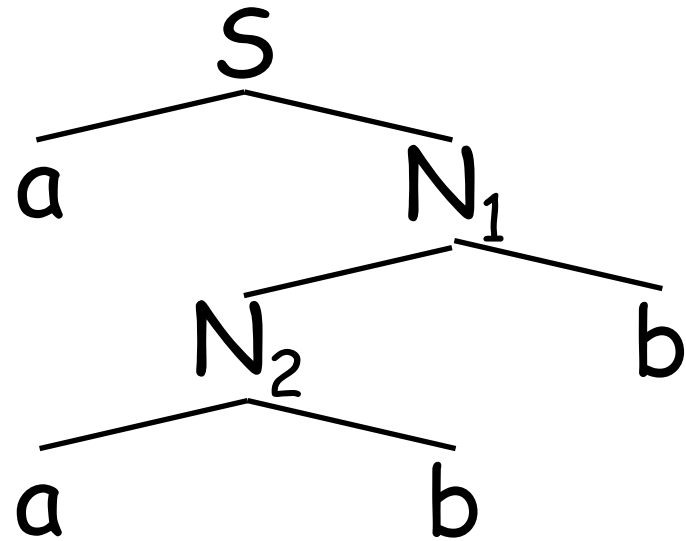


# Unlabeled Derivation Trees



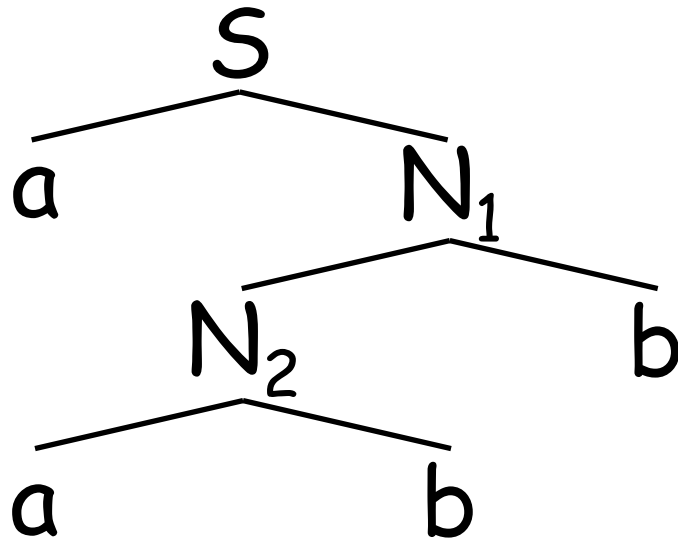


# Labelling UDTs





# Converting to Productions



$$S \rightarrow a N_1$$

$$N_1 \rightarrow N_2 b$$

$$N_2 \rightarrow a b$$

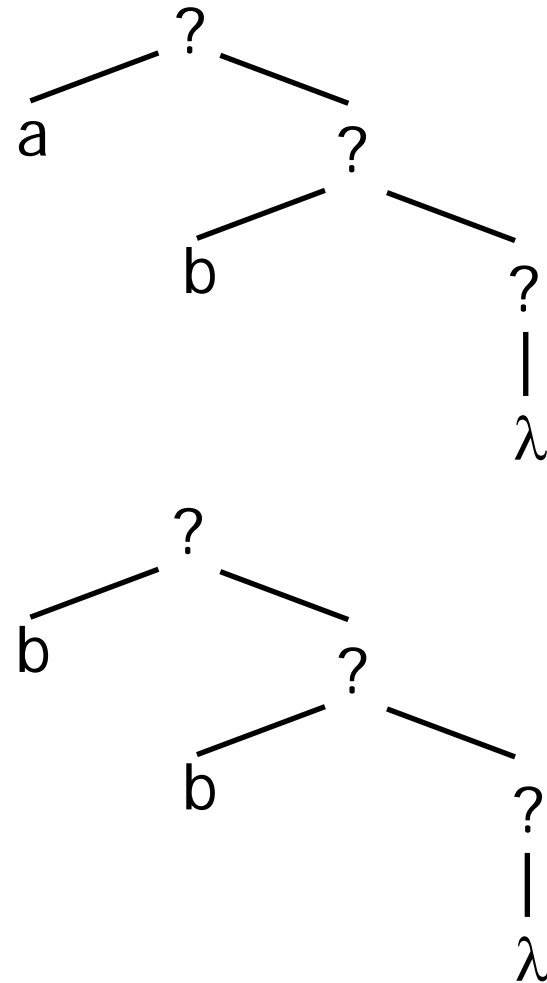
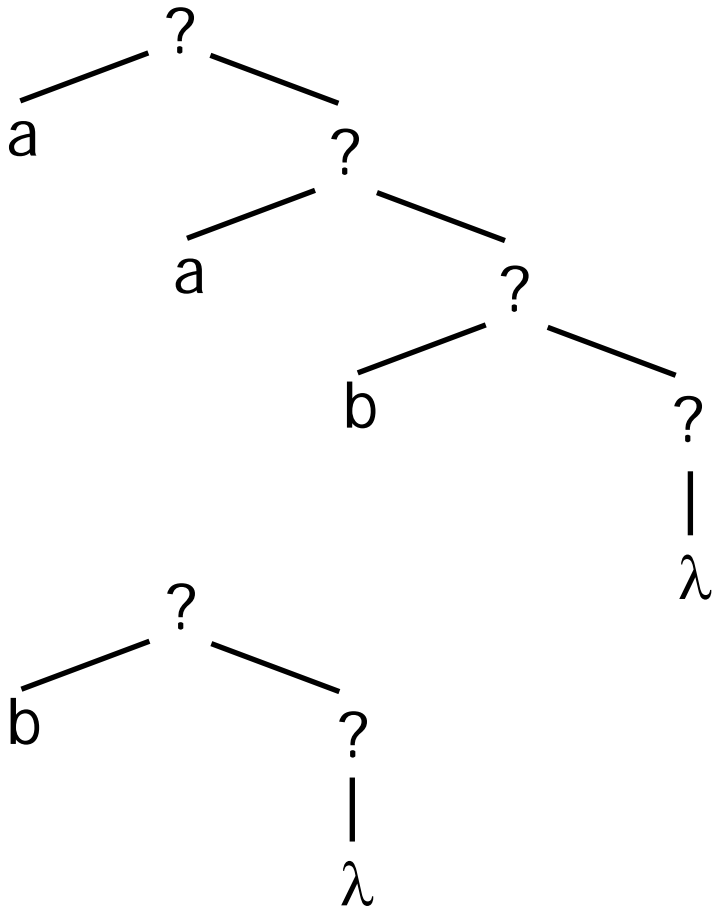
# Sakakibara's RC Algorithm

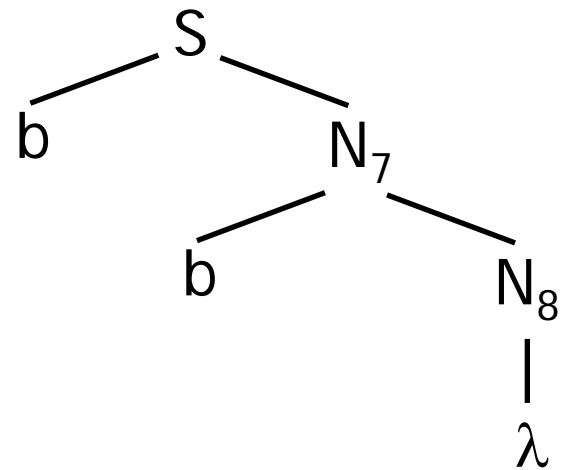
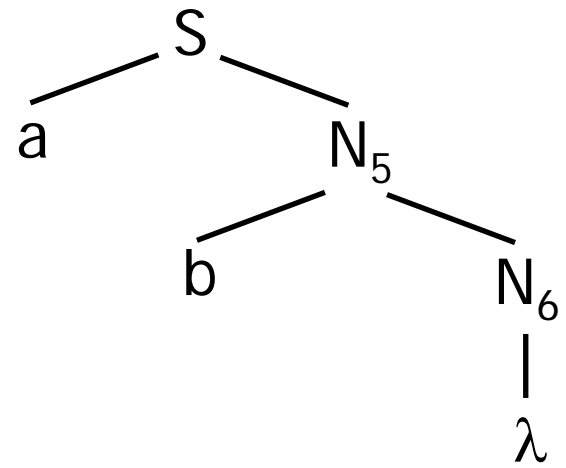
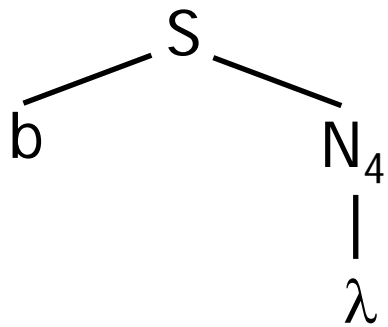
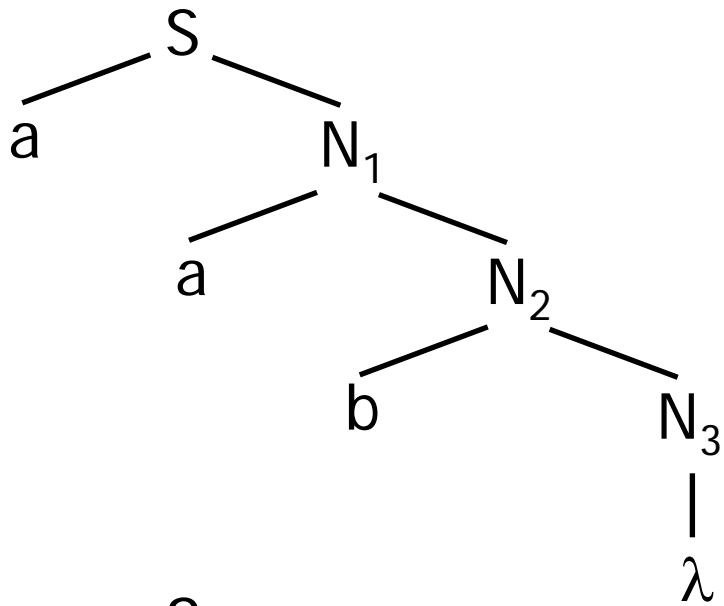


$G \leftarrow$  empty context-free grammar  
for each UDT in sample  
    assign non-terminal  $N$  to root node  
    assign unique NT names to all other nodes  
    convert to list of productions and add to  $G$   
while  $G$  violates either condition for  
    reversibility  
    merge any pair of non-terminals causing a  
    violation  
return  $G$



# An Example







$$S \rightarrow a N_1$$

$$N_1 \rightarrow a N_2$$

$$N_2 \rightarrow b N_3$$

$$N_3 \rightarrow \lambda$$

$$S \rightarrow b N_4$$

$$N_4 \rightarrow \lambda$$

$$S \rightarrow a N_5$$

$$N_5 \rightarrow b N_6$$

$$N_6 \rightarrow \lambda$$

$$S \rightarrow b N_7$$

$$N_7 \rightarrow b N_8$$

$$N_8 \rightarrow \lambda$$



$$S \rightarrow a N_1$$

$$N_1 \rightarrow a N_2$$

$$N_2 \rightarrow b N_3$$

$$N_3 \rightarrow \lambda$$

$$S \rightarrow b N_4$$

$$N_4 \rightarrow \lambda$$

$$S \rightarrow a N_5$$

$$N_5 \rightarrow b N_6$$

$$N_6 \rightarrow \lambda$$

$$S \rightarrow b N_7$$

$$N_7 \rightarrow b N_8$$

$$N_8 \rightarrow \lambda$$



$$S \rightarrow a N_1$$

$$N_1 \rightarrow a N_2$$

$$N_2 \rightarrow b N_3$$

$$N_3 \rightarrow \lambda$$

$$S \rightarrow b N_3$$

$$S \rightarrow a N_5$$

$$N_5 \rightarrow b N_3$$

$$S \rightarrow b N_7$$

$$N_7 \rightarrow b N_3$$



$$S \rightarrow a N_1$$

$$N_1 \rightarrow a N_2$$

$$N_2 \rightarrow b N_3$$

$$N_3 \rightarrow \lambda$$

$$S \rightarrow b N_3$$

$$S \rightarrow a N_5$$

$$N_5 \rightarrow b N_3$$

$$S \rightarrow b N_7$$

$$N_7 \rightarrow b N_3$$



$$S \rightarrow a N_1$$

$$N_1 \rightarrow a N_2$$

$$N_2 \rightarrow b N_3$$

$$N_3 \rightarrow \lambda$$

$$S \rightarrow a N_5$$

$$N_5 \rightarrow b N_3$$

$$S \rightarrow b N_3$$

$$N_3 \rightarrow b N_3$$



$$S \rightarrow a S$$

$$S \rightarrow b N_3$$

$$N_3 \rightarrow b N_3$$

$$N_3 \rightarrow \lambda$$

$$\mathbf{L(G)} = a^*b^+$$



$$S \rightarrow a S$$

$$S \rightarrow b N_3$$

$$N_3 \rightarrow b N_3$$

$$N_3 \rightarrow \lambda$$

$$\mathbf{L}(G) = a^* b^+$$



$$S \rightarrow a S$$

$$S \rightarrow b S$$

$$S \rightarrow \lambda$$

$$\mathbf{L}(\mathcal{G}) = \{a, b\}^*$$



## Claim 1

Given a set of strings  $S$  from a zero-reversible regular language,  $RC(S) = K\text{-RL}(S)$

# *K*-Reversible Context-Free Grammars



**Definition:** The *k-ancestors* of non-terminal  $A$  are the non-terminals that can derive a string containing  $A$  in exactly  $k$  steps

**Definition:** The *k-contexts* of non-terminal  $A$  are the strings that can be derived from the  $k$ -ancestors of  $A$  in  $k$  steps augmented with their unlabeled derivation trees



# ***K-Reversible Context-Free Grammars***

**Definition:** A context-free grammar is *k-reversible* if the following two conditions hold

- 1) If there exist productions of the form  $A \rightarrow aBb$  and  $A \rightarrow aCb$  then  $B = C$
- 2) If there exist productions of the form  $A \rightarrow a$  and  $B \rightarrow a$  and there exists a string  $b$  that is a  $k$ -context of both  $A$  and  $B$  then  $A = B$



# The KRCFG Algorithm

$G$  = empty context-free grammar

for each UDT in sample

    assign non-terminal  $S$  to root node

    assign unique NT names to all other nodes

    convert to list of productions and add to  $G$

while  $G$  violates either condition for  $k$ -  
    reversibility

    merge any pair of non-terminals causing a  
    violation

return  $G$



# Analysis

**Theorem 1:** KRCFG performs the least amount of generalization (i.e. merging) required to ensure that the grammar it returns is  $k$ -reversible

**Complexity:**  $O(m^{k+c_1} * n^{k+c_2})$

- $m$  = number of productions in original grammar
- $n$  = number of non-terminals in original grammar
- $c_1, c_2$  are small



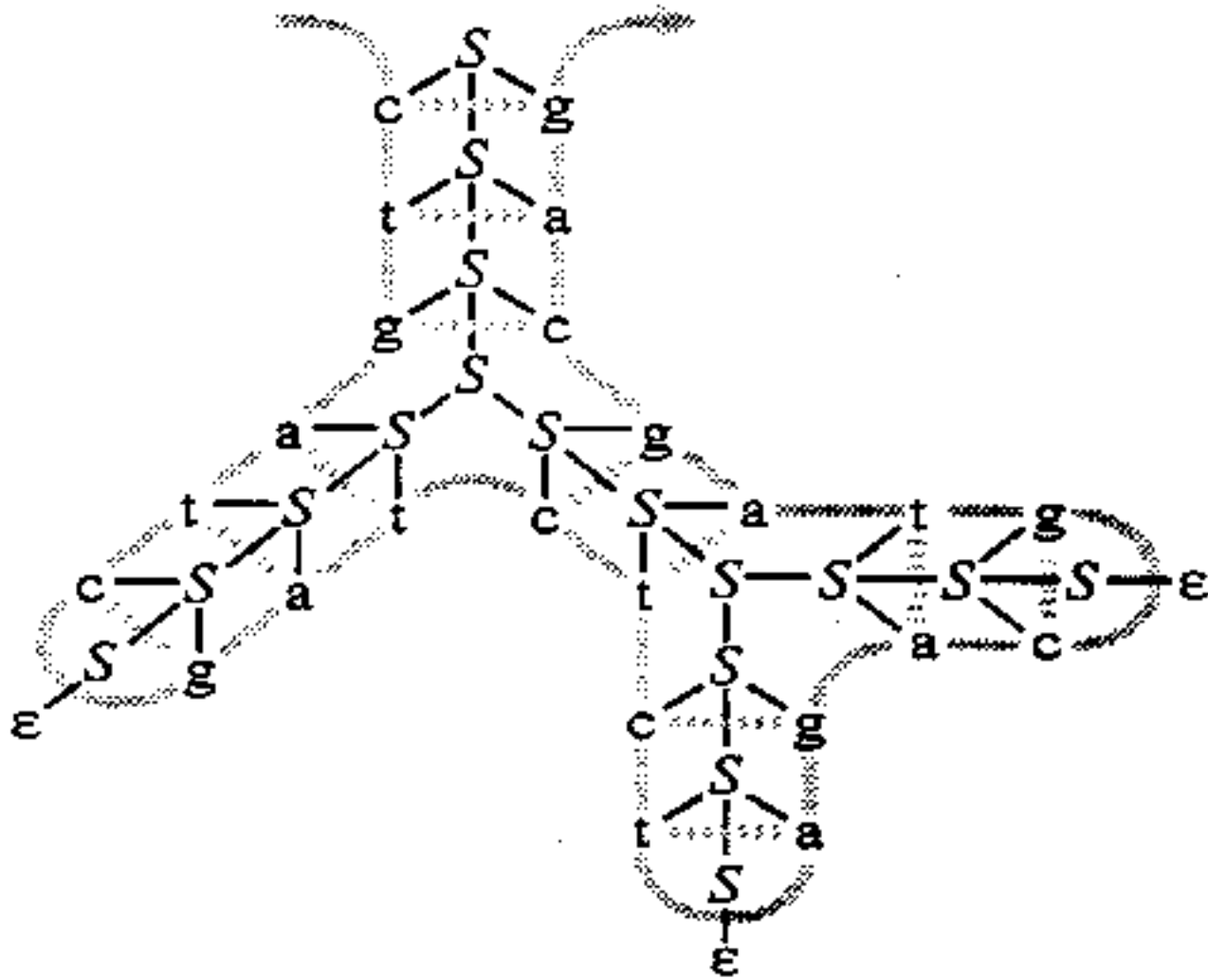
# 4 Applications

- Computational Biology
- Program synthesis, *ILP*, compiler construction
- Language models, speech & *NLP*
- Document structure, *XML*

# 4.1 Secondary structure predictions



- **Why:** find the secondary structure
- **Concept:** a *CF* grammar
- **Data:** long tagged strings over a small alphabet: (*RNA*)
- **Difficulties:**
  - only positive data : restrict to a subclass of *CF* grammars, or use stochastic *CF* grammars
- **Bibliography:** Sakakibara *et al.* 94, Abe & Mamitsuka 94



# Combining stochastic *CFGs* and *n*-grams over *RNA* sequences (Salvador & Benedi 2002)



- *CFGs* to learn the structure and long term dependencies
- bigrams for the local relations (non structured part)
- Sakakibara's algorithm (minimum reversible consistent *CFG*)
- Probability estimation (inside-outside)

## 4.2 Inductive logic programming



- **Why:** learn recursive programs
- **Concept:** tree automata and grammars
- **Input:** a transformation of examples and background knowledge into strings (*SLD* refutations, or terms)

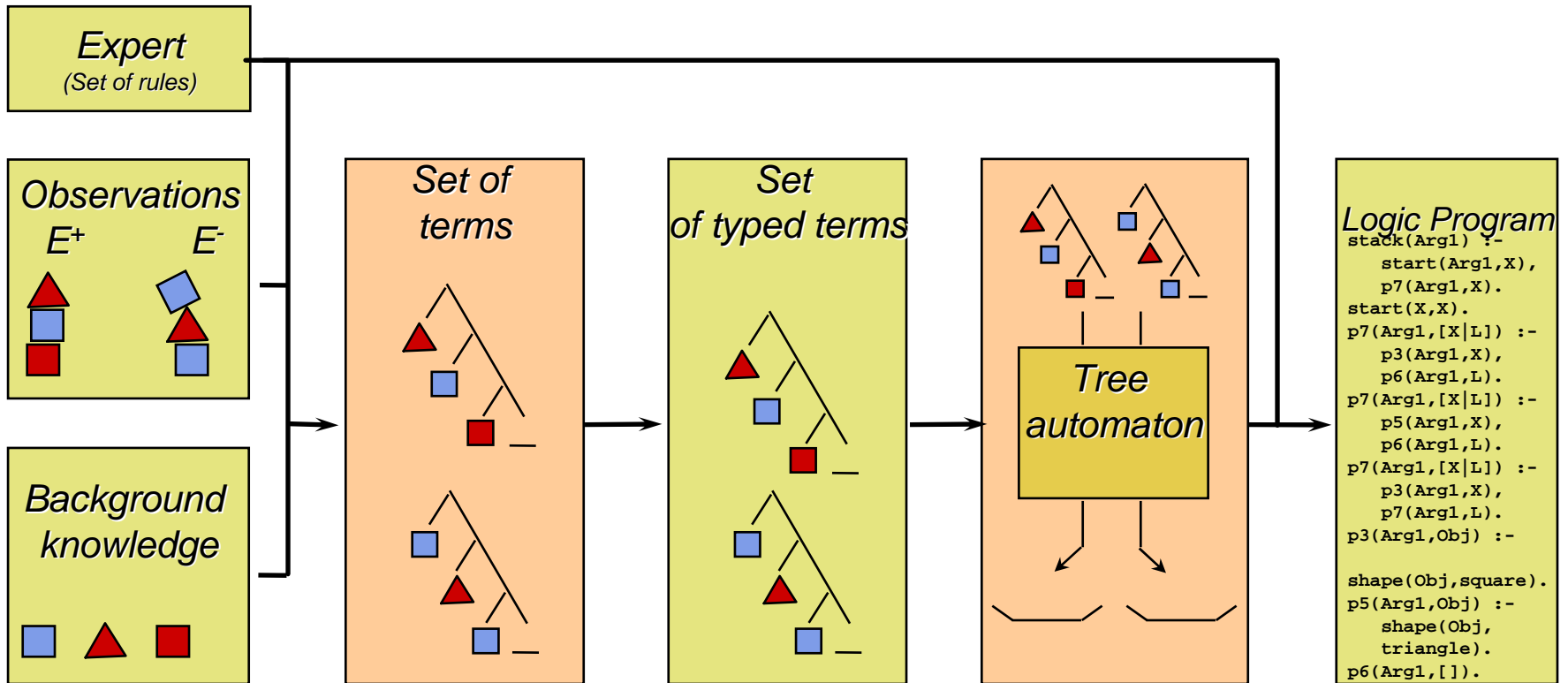


- **Difficulties:**
  - getting the first order information into strings/trees
  - regular grammars are very restricted
  - numeric data
  - post-transformation into a logic program
- **Bibliography:** Merlin, *GIFT* (Böstrom, 95 & 96, Bernard & cdlh, 99)

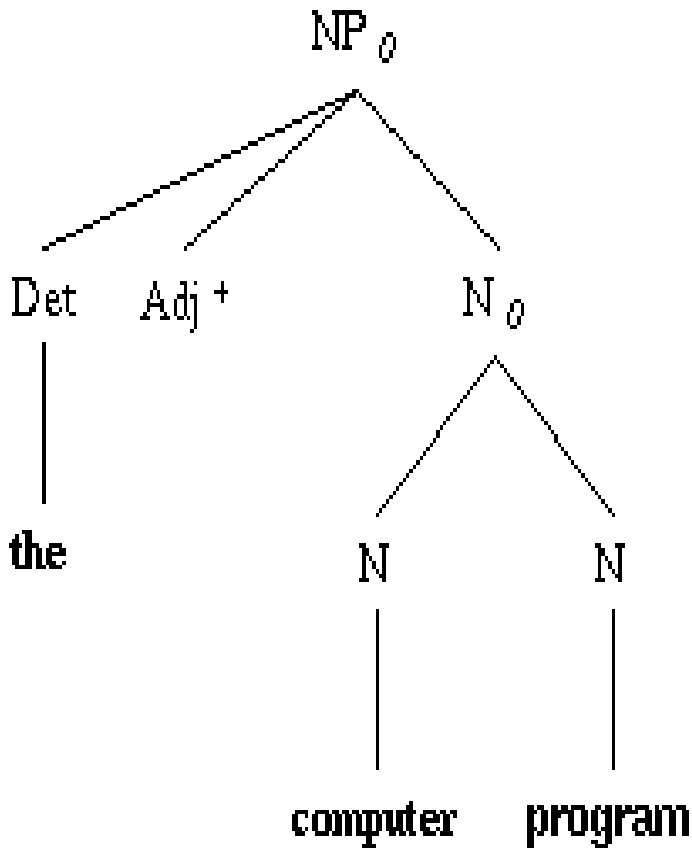


# GIFT

## architecture of the system



# 4.3 Natural Language Processing



[NP {subs 0}  
[Det [{bold the}]]  
[Adj {sup s 8 +}]  
[N {norm12 N} {subs 0}  
[N [{bold computer}]]  
[N [{sans program}]]]]

# System EMILE

(version 4.1.6, 2003) Marco Vervoort / Pieter Adriaans



## *Entity Modeling Intelligent Learning Engine*

- A context/expression pair is a sentence split into 3 parts: John (makes) tea.
  - **makes** is an expression
  - **John (.) tea** is a context.
- Identifying contexts, expressions is what EMILE is about.
- How? Through clustering algorithms



# An example

the fox jumped. the dog jumped.

the quick brown fox jumped.

the lazy dog jumped.

the fox jumped over the dog.

the dog jumped over the fox.

the quick brown fox jumped over the dog.

the lazy dog jumped over the fox.

the fox jumped over the lazy dog.

the dog jumped over the quick brown fox.

the lazy dog jumped over the quick brown fox.



# Result of Emile

- [0]→[18] dog jumped .
- [0]→the [4] jumped .
- [0]→[18] dog jumped over the [4] .
- [0]→the [4] jumped over [18] dog .
- [4]→fox
- [4]→quick brown [4]
- [18]→the
- [18]→the lazy



# System ABL

(van Zaanen, 00...)

- Uses alignments for grammar construction
- System for unsupervised learning

# 4.4 Structured documents: XML



Extract XML schema (Chiidlovski 200x)

```
<book>
  <part>
    <chapter>
      <sect1/>
      <sect1>
        <orderedlist numeration="arabic">
          <listitem/>
          <f:fragbody/>
        </orderedlist>
      </sect1>
    </chapter>
  </part>
</book>
```

# DTD



Copied from <http://www.vervet.com/>

```
<!DOCTYPE NEWSPAPER [  
<!ELEMENT NEWSPAPER (ARTICLE+)>  
<!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>  
<!ELEMENT HEADLINE (#PCDATA)>  
<!ELEMENT BYLINE (#PCDATA)> <!ELEMENT LEAD (#PCDATA)>  
<!ELEMENT BODY (#PCDATA)>  
<!ELEMENT NOTES (#PCDATA)>  
<!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>  
<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>  
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>  
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>  
<!ENTITY NEWSPAPER "Vervet Logic Times">  
<!ENTITY PUBLISHER "Vervet Logic Press">  
<!ENTITY COPYRIGHT "Copyright 1998 Vervet Logic Press">]>
```



## 5 Conclusion

- Theoretical hardness of the polynomial time learning issues
- determinism and linearity seem to play a strong part
- algorithms and heuristics are based on very clever ideas
- not enough comparable work



# Perspectives

- Tasks
  - Benchmarks
  - Prototypes
  - Clearly identifiable open problems
- Bottlenecks**
-

# Benchmarks, some features that one expects...



- Small/large/very large alphabets (2, <20, x0 000)
- All grammars/simple grammars
- Languages or grammars (normal forms?)
- Size
  - of data set
  - of grammars
- No help (only positive data)/some help:
  - Skeletons
  - Partial structure
  - Distribution
- Noise/no noise
- Recognition/tolerance



# Prototypes

- Avoid having a repetition of the *DFA*/stochastic *DFA* situation: no fixed *RPNI/Alergia* around
- distribution of implementations is a necessity
- distributing your algorithm means extra references!!!



# Open problems (1)

- Limits of learning from polynomial data?  
Comparison between models
- A plausible model for polynomial identification with probability 1 or something related to this...
- Find a problem solvable on strings for *DFA* but not solvable on skeletons for *CFGs*/tree automata



## Open problems (2)

- Provide an algorithm for the case of learning a stochastic  $CF$  grammar from strings
- Integrate the categorical grammars into the picture
- Learn deterministic linear grammars (*i.e.* one turn deterministic push-down automata)