Lattice based approaches

Learnable representations for languages

Alexander Clark

Department of Computer Science
Royal Holloway, University of London

August 2010
ESSLLI, 2010

Outline

This lecture

Take basic congruence based approaches and fix them.

- Start by considering a different CF approach
- Move to a mildly context-sensitive formalism to fix some efficiency issues.

Palindrome language

$L = \{\lambda, aa, aba, baab, \dots\}$

Inconvenient truth

$u \equiv_L v$ implies $u = v$

$[u] = \{u\}$

Proof

- Assume $|u| \leq |v|$
- $uu^R \in L$, so $vu^R \in L$
- So $v = uw$, and w is palindrome.
- w' is w with a changed to b and vice versa
- $uw'u^R \in L$, so $uww'u^R \in L$, so $|w| = 0$

Problem 2

CF language

$L = \{a^n b^n | n \geq 0\} \cup \{a^n b^{2n} | n \geq 0\}$

We need non-terminals that generate

- $a^n b^n$
- $a^n b^{2n}$
- ...

But *aabb* is not congruent to *aaabbb*!

Representational primitives

Strings

$[u] = \{v | v \equiv_L u\}$
The smallest possible sets
Gives us a partition of Σ^*

Contexts

$I[l, r] = \{v | lvr \in L\}$
These are the largest possible sets.
Overlap

Representational primitives

Strings

$[u] = \{v | v \equiv_L u\}$
The smallest possible sets
Gives us a partition of Σ^*

Contexts

$I[l, r] = \{v | lvr \in L\}$
These are the largest possible sets.
Overlap

Later we will look at:

Intersections

Given a set of contexts F
For every subset $C \subseteq F$
$C' = \{w | \forall (l, r) \in C, lwr \in L\}$
$\bigcap_{(l,r) \in C} I[l, r]$

Congruence based models

Primal

A priori rules

$[uv] \rightarrow [u][v]$
$[a] \rightarrow a$

Certain rules

$S \rightarrow [u]$

Defeasible rules

$[u] \rightarrow [v]$ iff $u \equiv_L v$

Outline

Crude approach

Single contexts

$I[l, r] = \{v | lvr \in L\}$

- $L = I[\lambda, \lambda]$

Finite set of contexts F that contains (λ, λ). Each context will be a non-terminal.

$\{a^n b^n | n \geq 0\}$

$I[a, b] = L$
$I[a, \lambda] = \{a^n b^{n+1} | n \geq 0\}$
$I[a, a] = \emptyset$
$I[a, abbb] = \{a\}$

Basic rules

Assume we have MQs

Lexical rules: certain

$[l, r] \to a$
Valid if $lar \in L$

The start symbol: a priori

$[\lambda, \lambda]$

Branching rules: defeasible

$[l_N, r_N] \to [l_P, r_P], [l_Q, r_Q]$
Valid only if $I[l_P, r_P] I[l_Q, r_Q] \subseteq I[l_N, r_N]$

Testing branching rules

Define a set of strings: K. Test whether:

$$(I[l_P, r_P] \cap K)(I[l_Q, r_q] \cap K) \subseteq I[l_N, r_N]$$

- Take every $u, v \in K$ such that $l_P u r_p, l_Q v r_q \in L$
- Test whether $l_N u v r_N \in L$

As K increases, the test gets harder.

Example

$L = \{a^n b^n | n \geq 0\}$

Suppose we have $(\lambda, b), (a, \lambda), (\lambda, \lambda)$.

- Consider $ab = a \circ b$
- $ab \in I[\lambda, \lambda]$
- $a \in I[\lambda, b]$
- $b \in I[a, \lambda]$

Example

$L = \{a^n b^n | n \geq 0\}$

Suppose we have $(\lambda, b), (a, \lambda), (\lambda, \lambda)$.

- Consider $ab = a \circ b$
- $ab \in I[\lambda, \lambda]$
- $a \in I[\lambda, b]$
- $b \in I[a, \lambda]$
- So maybe we have a rule $[\lambda, \lambda] \rightarrow [\lambda, b], [a, \lambda]$

Example

$L = \{a^n b^n | n \geq 0\}$

Suppose we have $(\lambda, b), (a, \lambda), (\lambda, \lambda)$.

- Consider $ab = a \circ b$
- $ab \in I[\lambda, \lambda]$
- $a \in I[\lambda, b]$
- $b \in I[a, \lambda]$
- So maybe we have a rule $[\lambda, \lambda] \rightarrow [\lambda, b], [a, \lambda]$
- But consider $aab \circ abb$

Example

$L = \{a^n b^n | n \geq 0\}$

Suppose we have $(\lambda, abb), (a, \lambda), (\lambda, \lambda)$.

- Consider $ab = a \circ b$
- $ab \in I[\lambda, \lambda]$
- $a \in I[\lambda, abb]$
- $b \in I[a, \lambda]$

Example

$L = \{a^n b^n | n \geq 0\}$

Suppose we have $(\lambda, abb), (a, \lambda), (\lambda, \lambda)$.

- Consider $ab = a \circ b$
- $ab \in I[\lambda, \lambda]$
- $a \in I[\lambda, abb]$
- $b \in I[a, \lambda]$
- So maybe we have a rule $[\lambda, \lambda] \rightarrow [\lambda, abb], [a, \lambda]$

Example

$L = \{a^n b^n | n \geq 0\}$

Suppose we have $(\lambda, abb), (a, \lambda), (\lambda, \lambda)$.

- Consider $ab = a \circ b$
- $ab \in I[\lambda, \lambda]$
- $a \in I[\lambda, abb]$
- $b \in I[a, \lambda]$
- So maybe we have a rule $[\lambda, \lambda] \rightarrow [\lambda, abb], [a, \lambda]$
- $I[\lambda, abb] = \{a\}$ – so valid.

Not a partition

In congruence class methods, we have a partition, so there is a unique N, $N \rightarrow P, Q$.
Multiple overlapping rules.

- $[\lambda, \lambda] \rightarrow [\lambda, abb], [a, \lambda]$
- $[\lambda, \lambda] \rightarrow [\lambda, abb], [aab, \lambda]$
- $[\lambda, \lambda] \rightarrow [\lambda, a], [aab, \lambda]$

Not a partition

In congruence class methods, we have a partition, so there is a unique N, $N \to P, Q$.
Multiple overlapping rules.

- $[\lambda, \lambda] \to [\lambda, abb], [a, \lambda]$
- $[\lambda, \lambda] \to [\lambda, abb], [aab, \lambda]$
- $[\lambda, \lambda] \to [\lambda, a], [aab, \lambda]$
- Unary rules: $[\lambda, a] \to [\lambda, abb]$

Algorithm idea

- Increase K as much as possible – $Sub(\{w_1, \ldots, w_n\})$.
- Eventually this will remove all incorrect rules
- When we undergenerate, increase F to get more rules
- $F = Con(\{w_1, \ldots, w_n\})$.

Algorithm

Data: Input alphabet Σ

Result: A sequence of CFGs $G_1, G_2, \ldots$

$K \leftarrow \Sigma \cup \{\lambda\}$, $F \leftarrow \{(\lambda, \lambda)\}$, $E = \{\}$;
$G = \texttt{Make}(K, F)$;
repeat
    $E \leftarrow E \cup \{w\}$;
    $K \leftarrow \mathit{Sub}(E)$;
    if *there is some* $w \in E$ *that is not in* $L(G)$ **then**
        $F \leftarrow \mathit{Con}(E)$;
        $G \leftarrow \texttt{Make}(K, F)$;
    end
    else
        $G \leftarrow \texttt{Make}(K,F)$;
    end
    Output G;
until w ;

Algorithm 1: CFG learning algorithm

Example

$L = \{a^n b^n | n \geq 0\}$

Start:

- $F = \{(\lambda, \lambda)\}$
- $K = \{a, b\lambda\}$

Grammar

- $[\lambda, \lambda] \rightarrow \lambda$
- $[\lambda, \lambda] \rightarrow [\lambda, \lambda], [\lambda, \lambda]$

$L(G_0) = \{\lambda\}$

Example

$L = \{a^n b^n | n \geq 0\}$

Step 1: receive example *ab*, not in current L.

- $K = \{\lambda, a, b, ab\}$
- $F = \{(\lambda, \lambda), (a, \lambda), (\lambda, b), (a, b)\}$

Lexical rules

- $[\lambda, \lambda] \rightarrow \lambda$
- $[\lambda, b] \rightarrow a$ and $[a, \lambda] \rightarrow b$

64 possible branching rules
But $[a, b]$ the same as $[\lambda, \lambda]$ so 27

Example

$L = \{a^n b^n | n \geq 0\}$

Step 1: receive example *ab*, not in current L.

- $K = \{\lambda, a, b, ab\}$
- $F = \{(\lambda, \lambda), (a, \lambda), (\lambda, b), (a, b)\}$

- $I[\lambda, \lambda] \cap K = \{\lambda, ab\}$
- $I[a, \lambda] \cap K = \{b\}$
- $I[\lambda, b] \cap K = \{a\}$
- $I[a, b] \cap K = \{\lambda, ab\}$

Example

$L = \{a^n b^n | n \geq 0\}$

Group 1

$[\lambda, \lambda] \to X, Y.$

	$[\lambda, \lambda]$	$[a, \lambda]$	$[\lambda, b]$
$[\lambda, \lambda]$	$ab \circ ab$	$\lambda \circ b$	$\lambda \circ a$
$[a, \lambda]$	$b \circ \lambda$	$b \circ b$	$b \circ a$
$[\lambda, b]$	$a \circ \lambda$	YES?	$a \circ a$

$$(I[\lambda, b] \cap K)(I[a, \lambda] \cap K) = \{a\}\{b\} = \{ab\} \subseteq I[\lambda, \lambda]$$

(Wrong!)

Example

$L = \{a^n b^n | n \geq 0\}$

Group 2

$[a, \lambda] \rightarrow X, Y.$

	$[\lambda, \lambda]$	$[a, \lambda]$	$[\lambda, b]$
$[\lambda, \lambda]$	$\lambda \circ \lambda$	YES?	$\lambda \circ a$
$[a, \lambda]$	$b \circ ab$	$b \circ b$	$b \circ a$
$[\lambda, b]$	$a \circ \lambda$	$a \circ b$	$a \circ a$

Example

$L = \{a^n b^n | n \geq 0\}$

Group 3

$[\lambda, b] \to X, Y.$

	$[\lambda, \lambda]$	$[a, \lambda]$	$[\lambda, b]$
$[\lambda, \lambda]$	$\lambda \circ \lambda$	$\lambda \circ b$	$ab \circ a$
$[a, \lambda]$	$b \circ ab$	$b \circ b$	$b \circ a$
$[\lambda, b]$	YES?	$a \circ b$	$a \circ a$

Example

$L = \{a^n b^n | n \geq 0\}$

Resulting grammar has rules:

- $[\lambda, \lambda] \rightarrow [\lambda, b], [a, \lambda]$
- $[a, \lambda] \rightarrow [\lambda, \lambda], [a, \lambda]$
- $[\lambda, b] \rightarrow [\lambda, b], [\lambda, \lambda]$

These are all incorrect.
Lexical rules:

- $[\lambda, \lambda] \rightarrow \lambda$
- $[\lambda, b] \rightarrow a$ and $[a, \lambda] \rightarrow b$

$L(G_1) = \{\lambda, ab, aabb, aababb \dots\}$

Example

$L = \{a^n b^n | n \geq 0\}$

- Next positive datum: *aabb*
- K includes *aab* and *abb* which knocks out these rules.
- Increase F to include $(\lambda, abb), (aab, \lambda)$

Example

$L = \{a^n b^n | n \geq 0\}$

- Next positive datum: *aabb*
- K includes *aab* and *abb* which knocks out these rules.
- Increase F to include $(\lambda, abb), (aab, \lambda)$
- Correct rules:
  - $[\lambda, \lambda] \to [\lambda, b][aab, \lambda]$
  - $[\lambda, \lambda] \to [\lambda, abb][aab, \lambda]$
  - $[\lambda, \lambda] \to [\lambda, abb][a, \lambda]$
  - $[\lambda, b] \to [\lambda, abb][\lambda, \lambda]$ etc.

Example

$$L_{nd} = \{a^n b^n c^m | n, m \geq 0\} \cup \{a^m b^n c^n | n, m \geq 0\}$$

$L(G, N)$	F_N
λ	$(aaabb, bccc)$
a	$(\lambda, abbccc)$
b	$(aaab, bccc)$
c	$(aaabbc, \lambda)$
c^*	(c, λ)
a^*	(λ, a)
$\{a^n b^n \| n \geq 0\}$	(a, b)
$\{a^n b^{n+1} \| n \geq 0\}$	(aa, b)
$\{b^n c^n \| n \geq 0\}$	(b, c)
$\{b^n c^{n+1} \| n \geq 0\}$	(bb, c)
L_{nd}	(λ, λ)

Classes are too large

Intermediate model

Fix a constant $f = 1, 2, \ldots$

Consider all subsets of F of at most f contexts

Polynomial size $|F|^f$

Very crude: doesn't really solve the problem.

Special case $f = 2$

Palindrome language

- $(\lambda, ba), (\lambda, aa)$ defines $\{a\}$
- $[(\lambda, \lambda)] \to [(\lambda, ba), (\lambda, aa)], [(a, \lambda)]$

$a^n b^n \cup a^n b^{2n}$

- $(\lambda, \lambda), (a, b)$ define $a^n b^n$
- $(\lambda, \lambda), (a, bb)$ define $a^n b^{2n}$

Outline

Introduction

Dual CFG model

- Single context
- Multiple contexts

Lattice approaches

- Linguistic justification
- Theory
- Formal concept analysis

Syntactic Concept Lattice

Background

- We can try all possible generalisations defined by any possible set of features
- use lattice theory
- Mildly context-sensitive approach

Sets of strings and contexts

Ambiguity

Simplified example

Suppose $N \stackrel{*}{\Rightarrow} u$, $M \stackrel{*}{\Rightarrow} v$
And $N \stackrel{*}{\Rightarrow} w$ $M \stackrel{*}{\Rightarrow} w$

Then we might have $C_L(w) = C_L(u) \cup C_L(v)$.

Motivation I

"can" is ambiguous:

- A can of beans
- I can see that

Motivation I

"can" is ambiguous:

- A can of beans
- I can see that

Distribution of "can" will be (roughly) the **union** of the distribution of count nouns and distribution of auxiliary verbs.

Motivation II

"may" is ambiguous:

- I like to go to the beach in may
- I may go to the beach

Motivation II

“may” is ambiguous:

- I like to go to the beach in may
- I may go to the beach

Distribution of auxiliary verbs is (roughly) the **intersection** of the distribution of “may” and “can”.

Rulon Wells

Immediate Constituents, Language 1947

It is easy to define a focus-class embracing a large variety of sequence classes but characterized by only a few environments; it is also easy to define one characterized by a great many environments in which all its members occur but on the other hand poor in the number of diverse sequence-classes that it embraces. What is difficult, but far more important than either of the easy tasks, is to define focus-classes rich both in the number of environments chracterizing them and at the same time in the diversity of sequence classes that they embrace.

- Concepts high up in the lattice have a few contexts, but lots of strings
- Concepts low down have a larger number of contexts, but only a few strings.

Key points

- We are dealing with sets of strings, sets of contexts
- These will overlap in very complex ways
- We may have words whose distributions are combinations of distributions of more primitive elements.
- The appropriate structure is a lattice.

Partially ordered sets

Definition

A set with a relation $\leq$ that is

transitive $X \leq Y \leq Z$ means $X \leq Z$

reflexive $X \leq X$

antisymmetric $X \leq Y$ and $Y \leq X$ means $X = Y$

Examples

- numbers with normal partial order
- sets with $X \leq Y$ iff $X \subseteq Y$
- sets with $X \leq Y$ iff $Y \subseteq X$

Lattice

Not all posets are lattices.
Two operations:

- $\vee$, least upper bound, join
- $\wedge$, greatest lower bound meet

Basic axioms

$$X \wedge X = X \quad (1)$$

$$X \wedge Y = Y \wedge X \quad (2)$$

$$(X \wedge Y) \wedge Z = X \wedge (Y \wedge Z) \quad (3)$$

$$X \vee (X \wedge Y) = X \quad (4)$$

$$(5)$$

Complete lattice has $\top$ and $\bot$, that are identities for the two operations.

General theory

Formal concept analysis:

- Set of strings
- Set of contexts

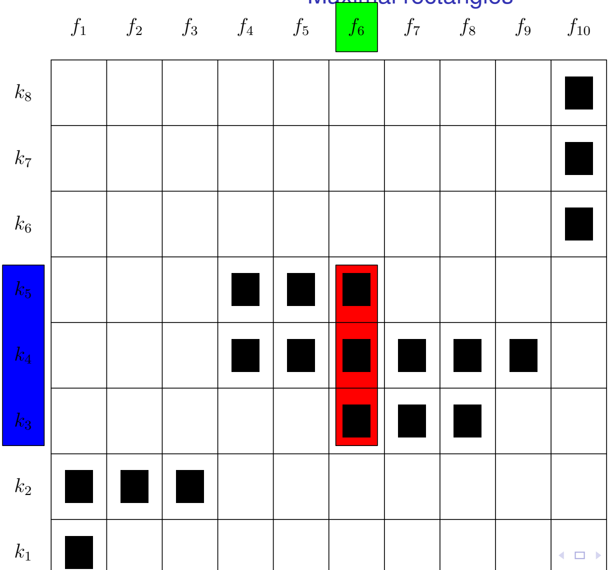
Concepts

Maximal rectangles

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
k_8										■
k_7										■
k_6										■
k_5				■	■	■				
k_4				■	■	■	■	■	■	
k_3						■	■	■		
k_2	■	■	■							
k_1	■									

Concepts

Maximal rectangles



Concepts

Maximal rectangles

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
k_8										■
k_7										■
k_6										■
k_5				■	■	■				
k_4				■	■	■	■	■	■	
k_3						■	■	■		
k_2	■	■	■							
k_1	■									

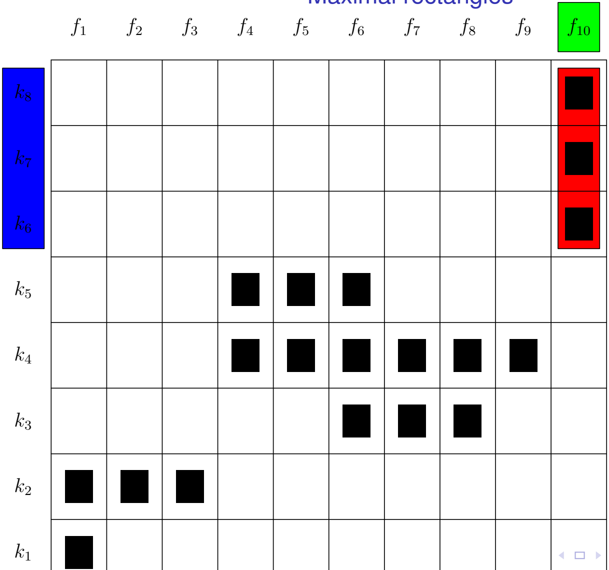
Concepts

Maximal rectangles

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
k_8										■
k_7										■
k_6										■
k_5				■	■	■				
k_4				■	■	■	■	■	■	
k_3						■	■	■		
k_2	■	■	■							
k_1	■									

Concepts

Maximal rectangles



Concepts

Maximal rectangles

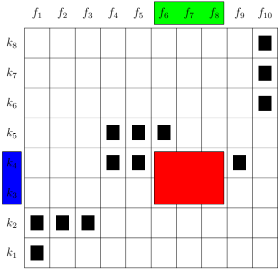
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
k_8										■
k_7										■
k_6										■
k_5				■	■	■				
k_4				■	■	■	■	■	■	
k_3						■	■	■		
k_2	■	■	■							
k_1	■									

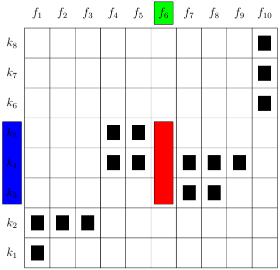
Concepts

Maximal rectangles

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
k_8										■
k_7										■
k_6										■
k_5				■	■	■				
k_4				■	■	■	■	■	■	
k_3						■	■	■		
k_2	■	■	■							
k_1	■									

Partial order





Top and bottom

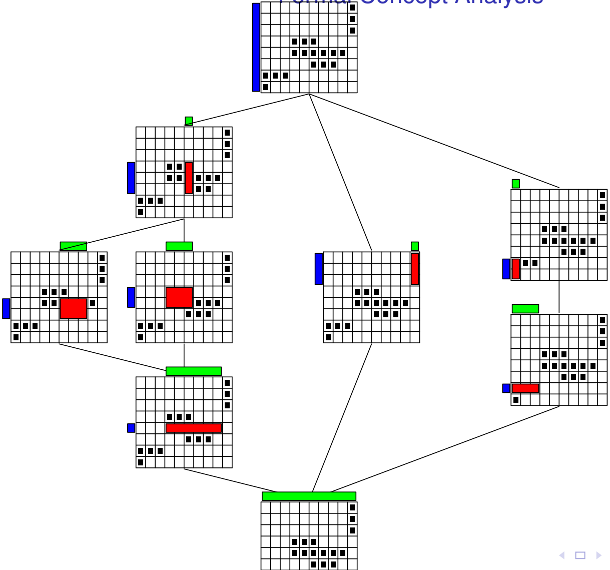
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
k_8										■
k_7										■
k_6										■
k_5				■	■	■				
k_4				■	■	■	■	■	■	
k_3						■	■	■		
k_2	■	■	■							
k_1	■									

Top and bottom

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
k_8										■
k_7										■
k_6										■
k_5				■	■	■				
k_4				■	■	■	■	■	■	
k_3						■	■	■		
k_2	■	■	■							
k_1	■									

Complete Lattice

Formal Concept Analysis



Polar maps

S is a set of strings, and C is a set of contexts.

Polar maps

$S' = \{(l, r) : \forall w \in S \;\; lwr \in L\}$
$C' = \{w : \forall (l, r) \in C \;\; lwr \in L\}$

$L = \{(\lambda, \lambda)\}'$

Concept

A syntactic concept is an ordered pair $\langle S, C \rangle$.
where $C' = S$ and $S' = C$.
Equivalently a maximal pair such that $C \odot S \subseteq L$.

Basic operations

Basic

If $S \subseteq T$, then $S' \supseteq T'$

$S \subseteq S''$

Lemma

$S''' = S', C''' = C'$

Proof: $S \subseteq S''$ so $S' \supseteq S'''$

$(S') \subseteq (S')''$

Closure operator

If $S = S''$ then S is closed.
If $\langle S, C \rangle$ is a concept, then S and C are both closed.
We can consider it as the lattice of closed sets of strings, or the lattice of closed sets of contexts, or both together.

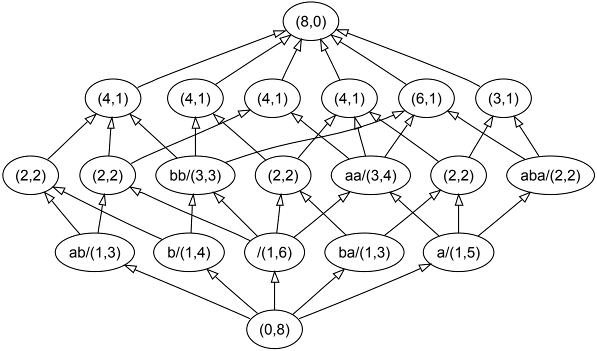
Lattice

Palindrome language over *a*, *b*

	(λ, λ)	(a, λ)	(ab, λ)	(aa, λ)	(b, λ)	(ba, λ)	(λ, b)	(λ, a)
aba	■		■					
bb	■				■		■	
ba		■	■				■	
ab					■	■		■
aa	■	■		■				■
b	■				■	■	■	
a	■	■	■	■				■

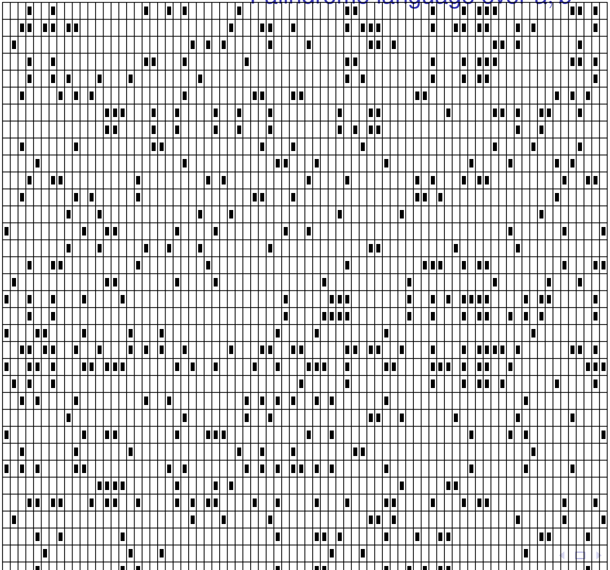
Lattice

Many rectangles

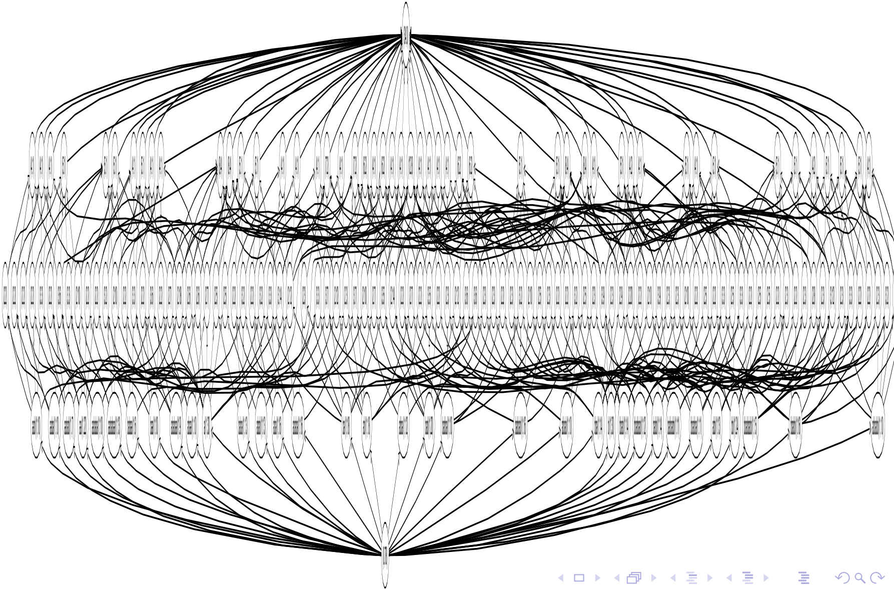


Lattice

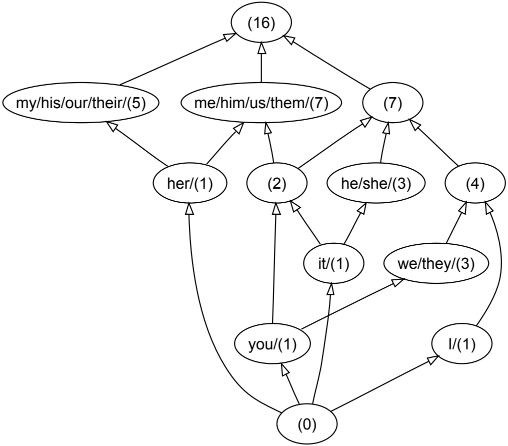
Palindrome language over *a*, *b*



Lattice



Linguistic concepts



Formally

Polar maps

$S' = \{(l, r) \in F : \forall w \in S \;\; lwr \in L\}$
$C' = \{w \in K : \forall (l, r) \in C \;\; lwr \in L\}$

Concept

Ordered pair $\langle S, C \rangle$

- $S \subseteq K$ the set of strings
- $C \subseteq F$ is a set of contexts

$S' = C$ and $C' = S$

$\mathcal{C}(S) = \langle S'', S' \rangle$

Relation to CFGs

Define

Given a CFG G for each non-terminal N

- Yield: $Y(N) = \{w | N \stackrel{*}{\Rightarrow} w\}$
- Contexts: $C(N) = \{(l, r) | S \stackrel{*}{\Rightarrow} lNr\}$.

Clearly $C(N) \odot Y(N) \subseteq L$
Each non-terminal will be a rectangle – but not necessarily maximal.

Technical detail

- These rectangles are "concepts" which form a complete lattice $\mathfrak{B}(K, L, F)$
- We use a concatenation operation $X \circ Y$ and a lower bound $X \wedge Y$.

Concatenation

$\langle S_1, C_1 \rangle \circ \langle S_2, C_2 \rangle = \langle (S_1 S_2)'', (S_1 S_2)''' \rangle$
Given two sets of strings S_x, S_y
Concatenate them $S_x S_y$
Find the shared set of contexts C_{xy}
Result is the concept defined by C'_{xy}

Dyck language

λ, ab, abab, aabb, abaabb ...

	(λ,λ)	(a,λ)	(λ,b)
λ	■		
a			■
b		■	
ab	■		

- $L = \langle\{\lambda, ab\}, (\lambda,\lambda)\rangle$
- $A = \langle\{a\}, (\lambda, b)\rangle$
- $B = \langle\{b\}, (a, \lambda)\rangle$
- $\top$
- $\bot$

Dyck language

- $L = \langle \{\lambda, ab\}, (\lambda, \lambda) \rangle$
- $A = \langle \{a\}, (\lambda, b) \rangle$
- $B = \langle \{b\}, (a, \lambda) \rangle$
- $\top$
- $\bot$

	$\top$	L	A	B	$\bot$
$\top$	$\top$	$\top$	$\top$	$\top$	$\bot$
L	$\top$	L	A	B	$\bot$
A	$\top$	A	$\top$	L	$\bot$
B	$\top$	B	$\top$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

Goal

Predict which concept a string is in:

- Define function $\phi : \Sigma^* \rightarrow \mathfrak{B}(K, L, F)$
- A string w is in the language if $\phi(w)$ has the context (λ, λ).
- We want $\phi(w) = \langle S, C \rangle$ to mean that $C_L(w) \cap F = C$.

Recursive definition

- $\phi(a) = \mathcal{C}(a)$ (look it up)
- $\phi(ab) = \phi(a) \circ \phi(b)$
- $\phi(abc) = \phi(ab) \circ \phi(c)$, OR $\phi(a) \circ \phi(bc)$

Goal

Predict which concept a string is in:

- Define function $\phi : \Sigma^* \to \mathfrak{B}(K, L, F)$
- A string w is in the language if $\phi(w)$ has the context (λ, λ).
- We want $\phi(w) = \langle S, C \rangle$ to mean that $C_L(w) \cap F = C$.

Recursive definition

- $\phi(a) = \mathcal{C}(a)$ (look it up)
- $\phi(ab) = \phi(a) \circ \phi(b)$
- $\phi(abc) = \phi(ab) \circ \phi(c)$, OR $\phi(a) \circ \phi(bc)$
- $\phi(abc) = \phi(ab) \circ \phi(c) \wedge \phi(a) \circ \phi(bc)$

Distributional lattice grammars

Derivation: efficient $\mathcal{O}(|w|^3)$ algorithm

Definition

A distributional lattice grammar (DLG) is a tuple $\langle K, D, F \rangle$

- K is a finite subset of strings that includes Σ and λ
- F is a finite set of contexts that includes (λ, λ)
- D is a finite subset of $F \odot KK$

Definition

$\phi : \Sigma^* \to \mathfrak{B}(K, D, F)$.

- for all $a \in \Sigma$, $\phi(a) = \mathcal{C}(a)$
- for all w with $|w| > 1$,

$$\phi(w) = \bigwedge_{u,v \in \Sigma^+ : uv = w} \phi(u) \circ \phi(v)$$

Derivation example

Dyck language

$\{\lambda, ab, aabb, abab, aaababbb \ldots\}$
$F = \{(\lambda, \lambda), (\lambda, b), (a, \lambda)\}$
$K = \{\lambda, a, b, ab\}$

- $\top = \langle K, \emptyset \rangle$
- $\bot = \langle \emptyset, F \rangle$
- $\mathbf{L} = \langle \{\lambda, ab\}, \{(\lambda, \lambda)\} \rangle$
- $A = \langle \{a\}, \{(\lambda, b)\} \rangle$
- $B = \langle \{b\}, \{(a, \lambda)\} \rangle$

Derivation example

Dyck language

$\{\lambda, ab, aabb, abab, aaababbb \dots\}$
$F = \{(\lambda, \lambda), (\lambda, b), (a, \lambda)\}$
$K = \{\lambda, a, b, ab\}$

1. $\phi(a) = A$
2. $\phi(ab) = \phi(a) \circ \phi(b) = \mathbf{L}$, $\phi(aa) = \top$ **...**
3. $\phi(aab) = (\phi(a) \circ \phi(ab)) \wedge (\phi(aa) \circ \phi(b)) = A \wedge \top = A$
4. **...**
5. $\phi(aaababbb) = \phi(a) \circ \phi(aababbb) \wedge \cdots = \mathbf{L}$

Example

$L = \{a^n b^n c^m | n, m \geq 0\} \cup \{a^m b^n c^n | n, m \geq 0\}$

	(λ, λ)	$(aaabb, bccc)$	$(aaabbc, \lambda)$	$(\lambda, abbccc)$	$(aaab, bccc)$	(aa, bbc)	$(aa, bbbc)$	(abb, cc)	$(abbb, cc)$
bcc									■
aab							■		
$bbcc$	■							■	
bc	■							■	
abc	■								
$aabb$	■					■			
ab	■					■			
c	■		■						■
b					■				
a	■			■			■		

Example

$L = \{a^n b^n c^m | n, m \geq 0\} \cup \{a^m b^n c^n | n, m \geq 0\}$

	(aa, bbc)	(λ, λ)	(abb, cc)	$(aaabb, bccc)$	$(aaabbc, \lambda)$	$(\lambda, abbccc)$	$(aaab, bccc)$	$(aa, bbbc)$	$(abbb, cc)$
bcc									■
aab								■	
abc		■							
$aabb$	■	■							
ab	■	■							
λ	■	■	■	■					
$bbcc$		■	■						
bc		■	■						
c		■			■				■
b							■		

Example

$L = \{a^n b^n c^m | n, m \geq 0\} \cup \{a^m b^n c^n | n, m \geq 0\}$

	(aa, bbc)	(λ, λ)	(abb, cc)	(aaabb, bccc)	(aaabbc, λ)	(λ, abbccc)	(aaab, bccc)	(aa, bbbc)	(abbb, cc)
bcc									■
aab								■	
abc		■							
aabb	■	■							
ab	■	■							
λ	■	■	■	■					
bbcc		■	■						
bc		■	■						
c		■			■				■
b							■		

Example

$L = \{a^n b^n c^m | n, m \geq 0\} \cup \{a^m b^n c^n | n, m \geq 0\}$

	(aa, bbc)	(λ, λ)	(abb, cc)	(aaabb, bccc)	(aaabbc, λ)	(λ, abbccc)	(aaab, bccc)	(aa, bbbc)	(abbb, cc)
bcc									■
aab								■	
abc		■							
aabb	■	■							
ab	■	■							
λ	■	■	■	■					
bbcc		■	■						
bc		■	■						
c		■			■				■
b							■		

Learnability I

Search

Language is defined by choice of K and F
How can we find suitable K and F?

Lemma 1

as we increase K the language defined by $\langle K, L, F\rangle$ decreases monotonically
It will always converge to a subset of L in a finite time

Lemma 2

As we increase the set of contexts F the language monotonically increases.
Any sufficiently large set of contexts will do.

Search problem is trivial

Naive Algorithm

Start with $F = \{(\lambda, \lambda)\}, K = \Sigma \cup \{\lambda\}$

- If we see a string that is not in our hypothesis, the hypothesis is too small, and we add contexts to F
- Add strings to K if it will change the lattice at all.

Clark, (CoNLL, 2010)

DLGs can be learnt from positive data and MQs
Polynomial update time

Power of Representation

Language class

Let $\mathcal{L}$ be the set of all languages L such that there is a *finite* set of contexts F s.t. $L = L(\mathfrak{B}(\Sigma^*, L, F))$

Learnable class includes

1. All regular languages
2. Some but not all CFLs (all the examples so far)
3. Some non context free languages

Not in DLG?

$L = \{a^n b | n > 0\} \cup \{a^n c^m | m > n > 0\}$

	(a, λ)	(aa, λ)	(aaa, λ)	(a^4, λ)	(a^5, λ)	(a^6, λ)	(a^7, λ)
c^9	■	■	■	■	■	■	■
c^8	■	■	■	■	■	■	■
c^7	■	■	■	■	■	■	■
c^6	■	■	■	■	■	■	■
c^5	■	■	■	■	■	■	
c^4	■	■	■	■	■		
ccc	■	■	■	■			
cc	■	■	■				
c	■	■					
b	■	■	■	■	■	■	■

Context sensitive example

MIX language variant

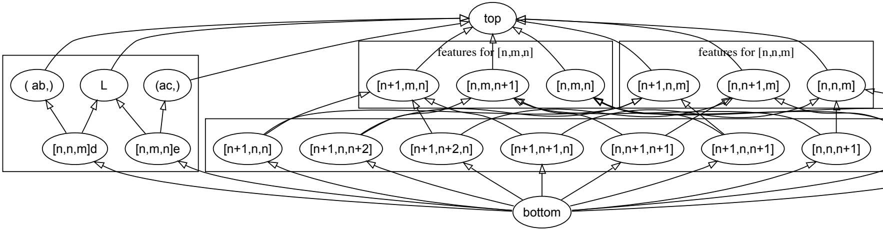
Let $M = \{(a, b, c)^*\}$, we consider the language $L = L_{abc} \cup L_{ab} \cup L_{ac}$ where

- $L_{ab} = \{wd | w \in M, |w|_a = |w|_b\}$,
- $L_{ac} = \{we | w \in M, |w|_a = |w|_c\}$,
- $L_{abc} = \{wf | w \in M, |w|_a = |w|_b = |w|_c\}$.

$F = \{(\lambda, \lambda), (\lambda, d), (\lambda, ad), (\lambda, bd), (\lambda, e), (\lambda, ae), (\lambda, ce), (\lambda, f), (ab, \lambda), (ac, \lambda)\}$

This is a non-context-free language in the learnable class.

Lattice


top
features for [n,m,n]
features for [n,n,m]
(ab,)
L
(ac,)
[n+1,m,n]
[n,m,n+1]
[n,m,n]
[n+1,n,m]
[n,n+1,m]
[n,n,m]
[n,n,m]d
[n,m,n]e
[n+1,n,n]
[n+1,n,n+2]
[n+1,n+2,n]
[n+1,n+1,n]
[n,n+1,n+1]
[n+1,n,n+1]
[n,n,n+1]
bottom

Outline

Syntactic concept lattice

Infinite limit

Let $K \to \Sigma^*$ and $F \to \Sigma^* \times \Sigma^*$

$\mathfrak{B}(K, L, F) \to \mathfrak{B}(L)$

- $\mathfrak{B}(L)$ is the *syntactic concept lattice*
- Same construction as the Universal Automaton for regular languages
- $\mathfrak{B}(L)$ is finite iff L is regular

Basic properties

Partial order

$\langle S_1, C_1 \rangle \leq \langle S_2, C_2 \rangle$ iff $S_1 \subseteq S_2$ iff $C_1 \supseteq C_2$

Lattice

The set of concepts of a language form a complete lattice

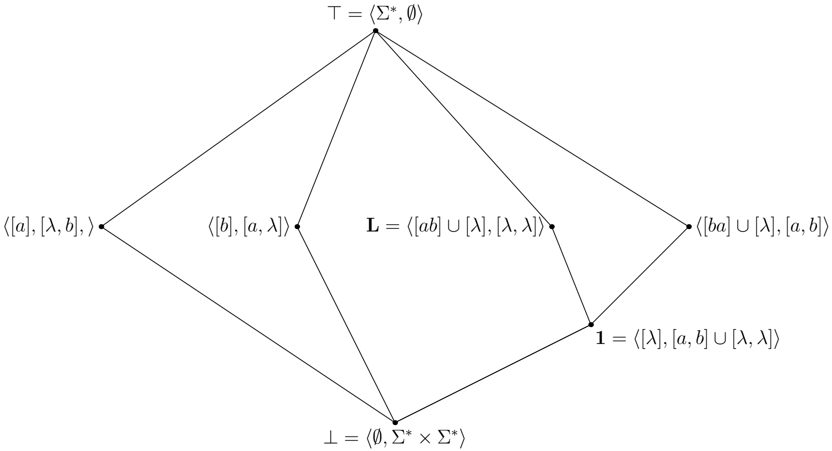
$\langle S_x, C_x \rangle \wedge \langle S_y, C_y \rangle = \langle S_x \cap S_y, (S_x \cap S_y)' \rangle$

Finite iff L is regular

Typical concepts

- Language $\langle L, \{(\lambda, \lambda)\}'' \rangle = \mathcal{C}(L) = \mathcal{C}((\lambda, \lambda))$
- Top $\top = \langle \Sigma^*, \emptyset \rangle$
- Bottom $\perp = \langle \emptyset, \Sigma^* \times \Sigma^* \rangle$
- Unit $\mathbf{1} = \mathcal{C}(\lambda)$

$L = (ab)^*$


$\top = \langle \Sigma^*, \emptyset \rangle$
$\langle [a], [\lambda, b], \rangle$
$\langle [b], [a, \lambda] \rangle$
$\mathbf{L} = \langle [ab] \cup [\lambda], [\lambda, \lambda] \rangle$
$\langle [ba] \cup [\lambda], [a, b] \rangle$
$\mathbf{1} = \langle [\lambda], [a, b] \cup [\lambda, \lambda] \rangle$
$\perp = \langle \emptyset, \Sigma^* \times \Sigma^* \rangle$

Concatenation is a monoid

Associativity

$(X \circ Y) \circ Z = X \circ (Y \circ Z)$

Lattice ordered monoid: monotonicity of concatenation w.r.t. partial order:

$X \leq Y$ then $X \circ Z \leq Y \circ Z$ etc.

Complete residuated lattice

- $X = \langle S_x, C_x \rangle$ and $Y = \langle S_y, C_y \rangle$ are concepts.
- Then define the residual $X/Y = \mathcal{C}(C_x \odot (\lambda, S_y))$
- $Y \setminus X = \mathcal{C}(C_x \odot (S_y, \lambda))$

These are unique, and satisfy the following conditions:

Lemma

$Y \leq X \setminus Z$ iff $X \circ Y \leq Z$ iff $X \leq Z/Y$.

Categorial grammar

Lambek calculus and CG

The Lambek calculus is based entirely on the theory of residuation.

Maximisation

- $Y \circ Z \leq X$

Categorial grammar

Lambek calculus and CG

The Lambek calculus is based entirely on the theory of residuation.

Maximisation

- $Y \circ Z \leq X$
- $(X/Z) \circ Z \leq X$

Categorial grammar

Lambek calculus and CG

The Lambek calculus is based entirely on the theory of residuation.

Maximisation

- $Y \circ Z \leq X$
- $(X/Z) \circ Z \leq X$
- $(X/Z) \circ (X/Z) \backslash X \leq X$
- Also have the opposite direction $X/(Z \backslash X) \circ (Z \backslash X) \leq X$

Structural descriptions from the lattice

Congruence based approaches

Get parse trees, but they are not useful.

Admissible structures for a string w

Each span has a concept $\psi[i, j]$

- $\psi[i, j] \geq \mathcal{C}(w[i : j])$
- $\psi[i, j] \geq \bigwedge_k \psi[i, k] \circ \psi[k, j]$
- $\psi[0, l] \leq \mathcal{C}(L)$

Maximal structures

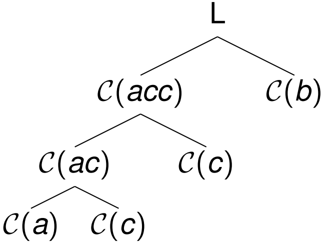
The set of maximal structures under the natural partial order can be viewed as the set of structural descriptions.
Discard $\top$ symbols and construct a graph or DAG.

Simple ambiguous language

Example

Dyck language with ambiguous symbol:
$\{ab, aabb, abab, aaababbb \ldots\}$
Add a symbol c which can be an a or a b
$\{ab, ac, cb, aabb, cccc, abab, aaacabbb \ldots\}$

Consider $accb$ – this could be $(ab)(ab)$ or $(a(ab)b)$.

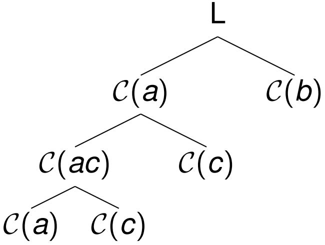


Simple ambiguous language

Example

Dyck language with ambiguous symbol:
$\{ab, aabb, abab, aaababbb \ldots\}$
Add a symbol c which can be an a or a b
$\{ab, ac, cb, aabb, cccc, abab, aaacabbb \ldots\}$

Consider $accb$ – this could be $(ab)(ab)$ or $(a(ab)b)$.

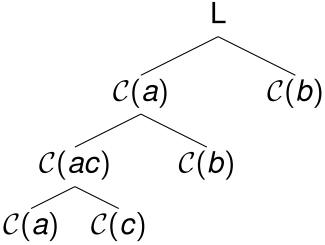


Simple ambiguous language

Example

Dyck language with ambiguous symbol:
$\{ab, aabb, abab, aaababbb \ldots\}$
Add a symbol c which can be an a or a b
$\{ab, ac, cb, aabb, cccc, abab, aaacabbb \ldots\}$

Consider $accb$ – this could be $(ab)(ab)$ or $(a(ab)b)$.

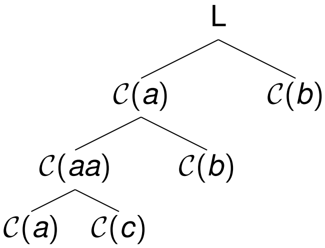


Simple ambiguous language

Example

Dyck language with ambiguous symbol:
$\{ab, aabb, abab, aaababbb \ldots\}$
Add a symbol c which can be an a or a b
$\{ab, ac, cb, aabb, cccc, abab, aaacabbb \ldots\}$

Consider $accb$ – this could be $(ab)(ab)$ or $(a(ab)b)$.

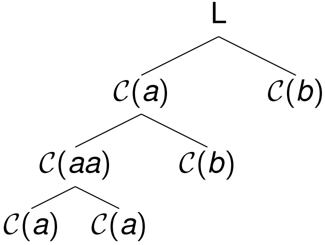


Simple ambiguous language

Example

Dyck language with ambiguous symbol:
$\{ab, aabb, abab, aaababbb \ldots\}$
Add a symbol *c* which can be an *a* or a *b*
$\{ab, ac, cb, aabb, cccc, abab, aaacabbb \ldots\}$

Consider *accb* – this could be $(ab)(ab)$ or $(a(ab)b)$.

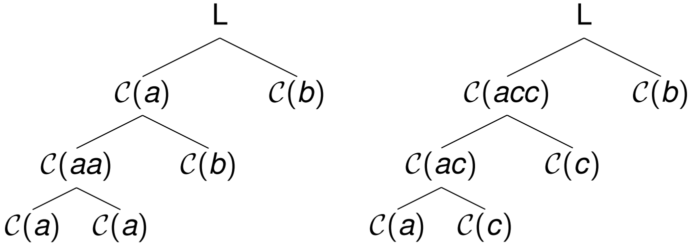


Simple ambiguous language

Example

Dyck language with ambiguous symbol:
$\{ab, aabb, abab, aaababbb \ldots\}$
Add a symbol c which can be an a or a b
$\{ab, ac, cb, aabb, cccc, abab, aaacabbb \ldots\}$

Consider *accb* – this could be $(ab)(ab)$ or $(a(ab)b)$.

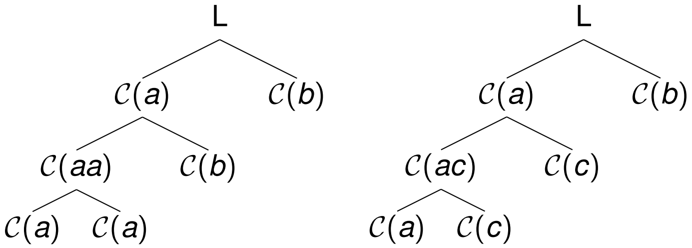


Simple ambiguous language

Example

Dyck language with ambiguous symbol:
$\{ab, aabb, abab, aaababbb \ldots\}$
Add a symbol c which can be an a or a b
$\{ab, ac, cb, aabb, cccc, abab, aaacabbb \ldots\}$

Consider $accb$ – this could be $(ab)(ab)$ or $(a(ab)b)$.

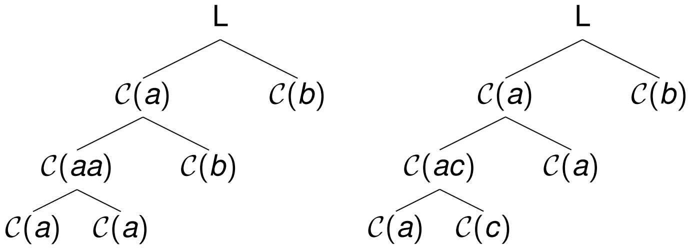


Simple ambiguous language

Example

Dyck language with ambiguous symbol:
$\{ab, aabb, abab, aaababbb \ldots\}$
Add a symbol c which can be an a or a b
$\{ab, ac, cb, aabb, cccc, abab, aaacabbb \ldots\}$

Consider $accb$ – this could be $(ab)(ab)$ or $(a(ab)b)$.

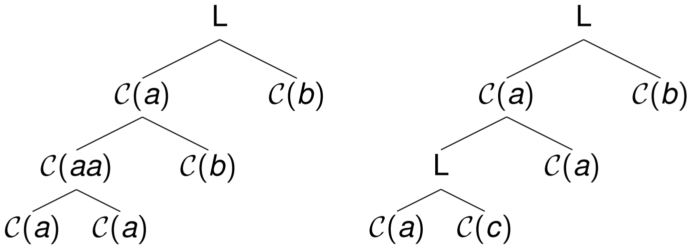


Simple ambiguous language

Example

Dyck language with ambiguous symbol:
$\{ab, aabb, abab, aaababbb \ldots\}$
Add a symbol c which can be an a or a b
$\{ab, ac, cb, aabb, cccc, abab, aaacabbb \ldots\}$

Consider $accb$ – this could be $(ab)(ab)$ or $(a(ab)b)$.

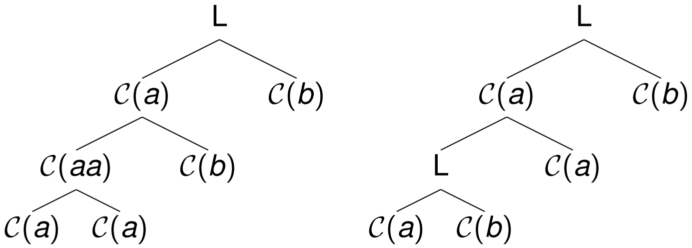


Simple ambiguous language

Example

Dyck language with ambiguous symbol:
$\{ab, aabb, abab, aaababbb \ldots\}$
Add a symbol c which can be an a or a b
$\{ab, ac, cb, aabb, cccc, abab, aaacabbb \ldots\}$

Consider $accb$ – this could be $(ab)(ab)$ or $(a(ab)b)$.



Tension between two notions of language

1950s

Abstraction - Chomsky

- Rich abstract structure that you need to model ambiguity etc.

Requires representations like CFGs, TAGs etc.

Learnability - Shannon

- Observable properties that mean you can learn

n-gram models

Tension between two notions of language

1950s

Abstraction - Chomsky

- Rich abstract structure that you need to model ambiguity etc.

Requires representations like CFGs, TAGs etc.

Learnability - Shannon

- Observable properties that mean you can learn

n-gram models

Not incompatible

There is a very rich abstract structure which is also observable and thus learnable.

CFG from the lattice

Alternatively we can stick with a CFG

- Just consider a polynomial number of elements of $\mathfrak{B}(K, L, F)$
- Fix f, and consider only concepts formed from $F^{\leq f}$
- Rules
  - $X \rightarrow YZ$ if $X \geq Y \circ Z$.
  - $X \rightarrow Y$ if $X \geq Y$
  - $S = \mathcal{C}((\lambda, \lambda))$
  - $X \rightarrow a$ if $\mathcal{C}(a) \leq X$

Clark, ICGI, 2010

Polynomial learnability from positive data and MQs, for fixed f

Conclusion

Richly learnable class:

- The class of languages is not obviously wrong.
- The learnability model is still too weak, but we would expect probabilistic results to be obtainable.
- Switch to a more efficient context-sensitive representation in order obtain efficient results.

However, the context-sensitivity may not be strong enough.