Learnable representations for natural languages: A Tutorial

Alexander Clark

Department of Computer Science
Royal Holloway, University of London

Acknowledgements

Joint work with Franck Thollard, Rémi Eyraud and Amaury Habrard, and many others.

Summary

We present efficient, correct algorithms for unsupervised learning of representations for natural languages:

- Use new representations that are designed to be learnable
  - Based on distributional analysis from structural linguistics
  - Empiricist – the data determines the representation.
- Richly structured context sensitive languages
  - Class of languages seems a good match to the class of natural languages;
- Based on lattice theory
- Use a symbolic learning paradigm

Table of contents

Outline

Problem

Learning representations for natural language

How can we construct representations for language?
Two motivations

- Scientific goal – understand first language acquisition (FLA)
- Engineering goal – build effective language processing systems

Current approach

Use supervised learning from annotated data.

This is no solution to either problem

How to study learnability?

Normal approach

- Take an existing class of representations (CFGs, TAGs, ACGs, PRCGs . . .)
- Try to design learning algorithms for that class

Two problems

- "At the most fundamental level, it is not clear that there is any meaningful empirical motivation for the representational assumptions of any current formal model of syntax." Blevins, J. (2009)
- Learning standard representations is computationally hard: Kearns and Valiant (1994), Angluin and Kharitonov (1995) . . .

How to study learnability?

Normal approach

- Take an existing class of representations (CFGs, TAGs, ACGs, PRCGs . . .)
- Try to design learning algorithms for that class

Two problems

- "At the most fundamental level, it is not clear that there is any meaningful empirical motivation for the representational assumptions of any current formal model of syntax." Blevins, J. (2009)
- Learning standard representations is computationally hard: Kearns and Valiant (1994), Angluin and Kharitonov (1995) . . .

Reasonable Research Strategy

In FLA we don't know what the representations are but we do know that they are learnable.

Slogan

Put learnability first!
Look for new representations that are intrinsically learnable.

Normal direction

Function from representation to language
Context free grammar $G \rightarrow$ context free language $L(G)$
Non-terminal $\rightarrow$ set of strings derived from non-terminal

Opposite Direction

Function from language to representation
$L \rightarrow R(L)$
From set of strings $\rightarrow$ representational primitive of formalism

Reasonable Research Strategy

In FLA we don't know what the representations are but we do know that they are learnable.

Slogan

Put learnability first!
Look for new representations that are intrinsically learnable.

Normal direction

Function from representation to language
Context free grammar $G \rightarrow$ context free language $L(G)$
Non-terminal $\rightarrow$ set of strings derived from non-terminal

Opposite Direction

Function from language to representation
$L \rightarrow R(L)$
From set of strings $\rightarrow$ representational primitive of formalism

Reasonable Research Strategy

In FLA we don't know what the representations are but we do know that they are learnable.

Slogan

Put learnability first!
Look for new representations that are intrinsically learnable.

Normal direction

Function from representation to language
Context free grammar $G \rightarrow$ context free language $L(G)$
Non-terminal $\rightarrow$ set of strings derived from non-terminal

Opposite Direction

Function from language to representation
$L \rightarrow R(L)$
From set of strings $\rightarrow$ representational primitive of formalism

Empiricist models

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
- Richer structures will give you more powerful representations

Outline

Regular languages

A success story for learning

Regular language L
Defined by a Deterministic Finite Automaton A

- finite set of states Q
- Transition function $q \rightarrow^a q'$ given by $\delta(q, a) = q'$
- final states Q_F
- initial state q_o

Define

- $L(q) = \{w | \delta(q, w) \in Q_F\}$
- $L(A) = L(q_0)$

Myhill-Nerode theorem

Right congruence

$u \equiv_R v$ iff for all suffixes w
$uw \in L$ iff $vw \in L$
i.e. $u^{-1}L = v^{-1}L$; write $[u]_R$ for congruence class

Myhill-Nerode theorem

- Number of states in minimal DFA
- Number of equivalence classes under this relation

We can tell whether $u \equiv_R v$:

- Take a sample:
- Compare suffixes of u with suffixes of v

Example

$L = (ab)^*$

	λ	ab	b	bab
λ	1	1	0	0
ab	1	1	0	0
a	0	0	1	1
aba	0	0	1	1
ababa	0	0	1	1
b	0	0	0	0

- $[\lambda]_R = \{\lambda, ab, abab \dots\}$
- $[a]_R = \{a, aba, abab, \dots\}$
- $[b]_R = \{b, aa, bba, \dots\}$

Transitions

Define states to be congruence classes!

Key point

If $u \equiv_R v$ then $ua \equiv_R va$

- $[u] \rightarrow^a [ua]$

- Take an element of $[a]$, say a.
- Append b we get $[ab] = [\lambda]$
- Transition $[a] \rightarrow^b [\lambda]$

Initial state is $[\lambda]$, final set of states is every state $[u]$ where $u \in L$.

Learning

Correspondence

Between

- Representational primitives — States $L(q)$
- Language properties — Right congruence classes

Canonical DFA

$L \rightarrow A(L)$

- Set of states = set of equivalence classes
- Transition $\delta([u], a) = [ua]$.
- $[u] \in Q_F$ iff $u \in L$
- $q_0 = [\lambda]$.

Learning DFAs and PDFAs

Results

- Angluin (1982)
- Oncina and Garcia (1992)
- Carrasco and Oncina (1999)
- Ron et al (JCSS 1998)
- Clark and Thollard (JMLR 2004)

Limitations

- Relies on frequent suffixes co-occurring with frequent prefixes.
- Won't work on e.g. parity functions with a uniform distribution over n-bit strings.
- Kearns and Valiant (JACM 1994)

Outline

Beyond regular languages

How can we extend these results beyond regular languages?

Context (or *environment*)

A context is just a pair of strings $(l, r) \in \Sigma^* \times \Sigma^*$.
$(l, r) \odot u = lur$ $f = (l, r)$.

Distribution of a string

$C_L(u) = \{(l, r) | lur \in L\} = \{f | f \odot u \in L\}$

Syntactic congruence

$u \equiv v$ iff $C_L(u) = C_L(v)$
Write $[u]$ for class of u

"Distributional Learning" models the distribution of strings.

Observable structure

We can observe whether $u \equiv v$: compare observed contexts of u with those of v.

Key point

$u \equiv u'$ and $v \equiv v'$ then $uv \equiv u'v'$

Canonical rules

$L \to G(L)$

- V is the set of congruence classes of a finite set of strings
- $[u]$ is a start symbol iff $u \in L$
- $[a] \to a$
- $[uv] \to [u][v]$

Example

Target language

Suppose $L = \{a^n b^n | n > 0\}$.

Finite set of congruence classes

- $[a] = \{a\}$, $[b] = \{b\}$
- $[ab] = \{ab, aabb, \dots\}$
- $[abb] = \{abb, aabbb \dots\} \dots$

- $[a] \rightarrow a, [b] \rightarrow b$
- $[ab] \rightarrow [a][b]$ and $[abb] \rightarrow [ab][b]$
- Since $[ab] = [aabb]$, $[ab] \rightarrow [a][abb]$

Example

Target language

Suppose $L = \{a^n b^n | n > 0\}$.

Finite set of congruence classes

- $[a] = \{a\}$, $[b] = \{b\}$
- $[ab] = \{ab, aabb, \dots\}$
- $[abb] = \{abb, aabbb \dots\} \dots$

- $[a] \rightarrow a, [b] \rightarrow b$
- $[ab] \rightarrow [a][b]$ and $[abb] \rightarrow [ab][b]$
- Since $[ab] = [aabb]$, $[ab] \rightarrow [a][abb]$

Elementary results

Substitutable languages

assume that if $C_L(u) \cap C_L(v)$ is non empty then $C_L(u) = C_L(v)$.

Results

- Clark and Eyraud (JMLR 2008)
- PAC learning NTS-languages: Clark (2007)
- *k*-*l* substitutable languages, Yoshinaka (2008)

Limitations

One symbol per congruence class won't work for natural languages:

- Congruence classes are very many and very close together
- Learning model assumes that either they are identical or they are completely unrelated.
- Need to have a more powerful representation that represents the structure of the congruence classes
- Languages aren't context free

$NP \rightarrow DT\ N$

	company	companies	associate	furniture	sheep	oil
the	1	1	1	1	1	1
a	1				1	
an			1			1
this	1		1	1	1	1
those		1			1	
some		1		1	1	1

- Every determiner and noun is in a different congruence class
- Different rule for each combination

$NP \rightarrow DT\ N$

	company	companies	associate	furniture	sheep	oil
the	1	1	1	1	1	1
a	1				1	
an			1			1
this	1		1	1	1	1
those		1			1	
some		1		1	1	1

- Every determiner and noun is in a different congruence class
- Different rule for each combination

Outline

Lattice

Lattice

$C_L(u)$ is a set of contexts, so we should represent it as a lattice. (Sestier, 1960)

Example

$N \Rightarrow^* n$, $N \Rightarrow^* v$ and $M \Rightarrow^* m$, $M \Rightarrow^* v$ then

- $C(v) = C(n) \cup C(m)$

Linguistically: the distribution of “her” is going to be a union of the distribution of “his” and “him”.

Contextual Features

- Pick a finite set of k contexts F
- represent $C_L(u)$ by $C_L(u) \cap F$
- This gives us 2^k possible congruence classes

Representation

We define a formalism that directly uses this lattice structure:

Lemma

For any language L and for any strings u, u', v, v' if $C(u) \supseteq C(u')$ and $C(v) \supseteq C(v')$, then $C(uv) \supseteq C(u'v')$.

$$C(w) \supseteq \bigcup_{\substack{u,v: \\ uv=w}} \bigcup_{\substack{u' \in K: \\ C(u') \subseteq C(u)}} \bigcup_{\substack{v' \in K: \\ C(v') \subseteq C(v)}} C(u'v') \qquad (1)$$

Representation

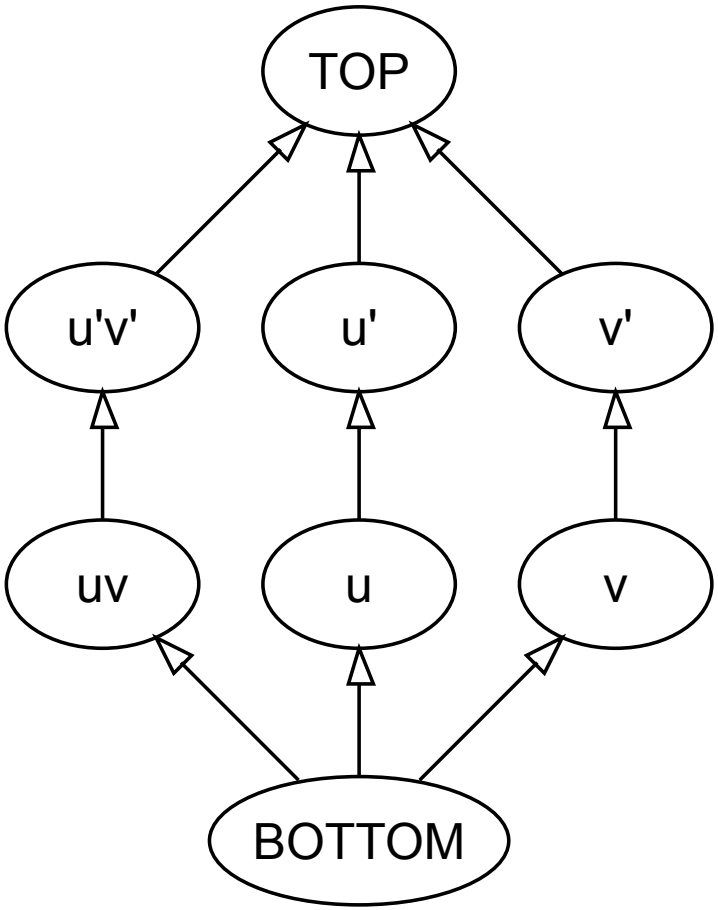
We define a formalism that directly uses this lattice structure:

Lemma

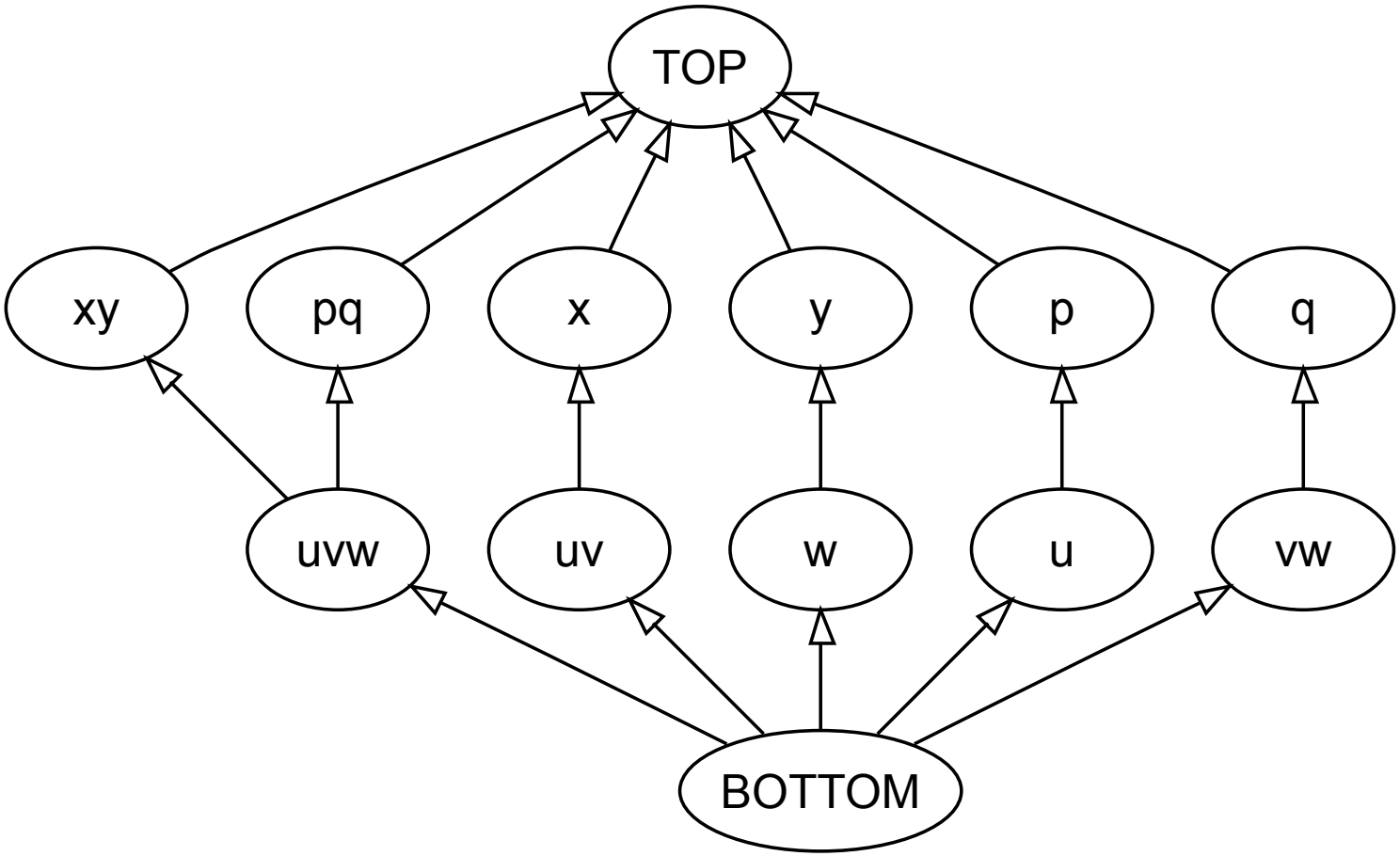
For any language L and for any strings u, u', v, v' if $C(u) \supseteq C(u')$ and $C(v) \supseteq C(v')$, then $C(uv) \supseteq C(u'v')$.

$$C(w) \supseteq \bigcup_{\substack{u,v: \\ uv=w}} \bigcup_{\substack{u' \in K: \\ C(u') \subseteq C(u)}} \bigcup_{\substack{v' \in K: \\ C(v') \subseteq C(v)}} C(u'v') \tag{1}$$

Lattice rules



Lattice rules II



Contextual Binary Feature Grammars

Formalism

A CBFG G is a tuple $\langle F, f_s, P, P_L, \Sigma \rangle$.

- $f_s \in F$ is the sentence feature (λ, λ)
- Productions
  - P_L has $x \to a$ where $x \subseteq F$ and $a \in \Sigma$.
  - P has $x \to y, z$ where $x, y, z \subseteq F$.
- Informally: if u has features y, and v has features z, then uv will have the features in x.
- f_G is a recursive map from $\Sigma^* \to 2^F$
- We want $f_G(u)$ to approximate $C(u) \cap F$.

Recursive computation

$$f_G(\lambda) = \emptyset \tag{2}$$

$$f_G(w) = \bigcup_{(c \to w) \in P_L} c \qquad \text{iff } |w| = 1 \tag{3}$$

$$f_G(w) = \bigcup_{u,v:uv=w} \; \bigcup_{\substack{x \to yz \in P: \\ y \subseteq f_G(u) \wedge \\ z \subseteq f_G(v)}} x \qquad \text{iff } |w| > 1. \tag{4}$$

This is similar to a CKY parsing algorithm: $\mathcal{O}(|w|^3|P|)$

Recursive computation

$$f_G(\lambda) = \emptyset \tag{2}$$

$$f_G(w) = \bigcup_{(c \to w) \in P_L} c \qquad \text{iff } |w| = 1 \tag{3}$$

$$f_G(w) = \bigcup_{u,v:uv=w} \bigcup_{\substack{x \to yz \in P: \\ y \subseteq f_G(u) \wedge \\ z \subseteq f_G(v)}} x \qquad \text{iff } |w| > 1. \tag{4}$$

This is similar to a CKY parsing algorithm: $\mathcal{O}(|w|^3|P|)$

Power of BFGs

1. Include all CFGs
2. Can represent non context free languages; (almost) closed under intersection
3. Can compactly represent languages that require exponentially large context free grammars: the finite language consisting of all permutations of a finite alphabet.
4. Equivalent to subclass of Conjunctive Grammars (Okhotin, 2001)

Writing down a grammar

Assume we have a finite set of strings K, and a finite set of features F, and a membership oracle:

Productions P

- If u, v and uv are in K
- $C(u)$ and $C(v)$ combine to form $C(uv)$
- Add production $C(uv) \cap F \rightarrow C(u) \cap F, C(v) \cap F$

Productions P_L

- For a letter $a \in \Sigma$, we add $C(a) \cap F \rightarrow a$.

Search

- Given an oracle for the language L, and a choice for K and F, we can *write down* a grammar $G_0(K, L, F)$.
- G, the hypothesis, is a function of K and F.
- Is it easy to find the right K and F?

Monotonicity of K

Obvious

As K increases the language increases

If $K \subseteq K^+$, then $L(G_0(K, L, F)) \subseteq L(G_0(K^+, L, F))$

Proof: the set of productions increases as we increase the number of examples.

Monotonicity of F

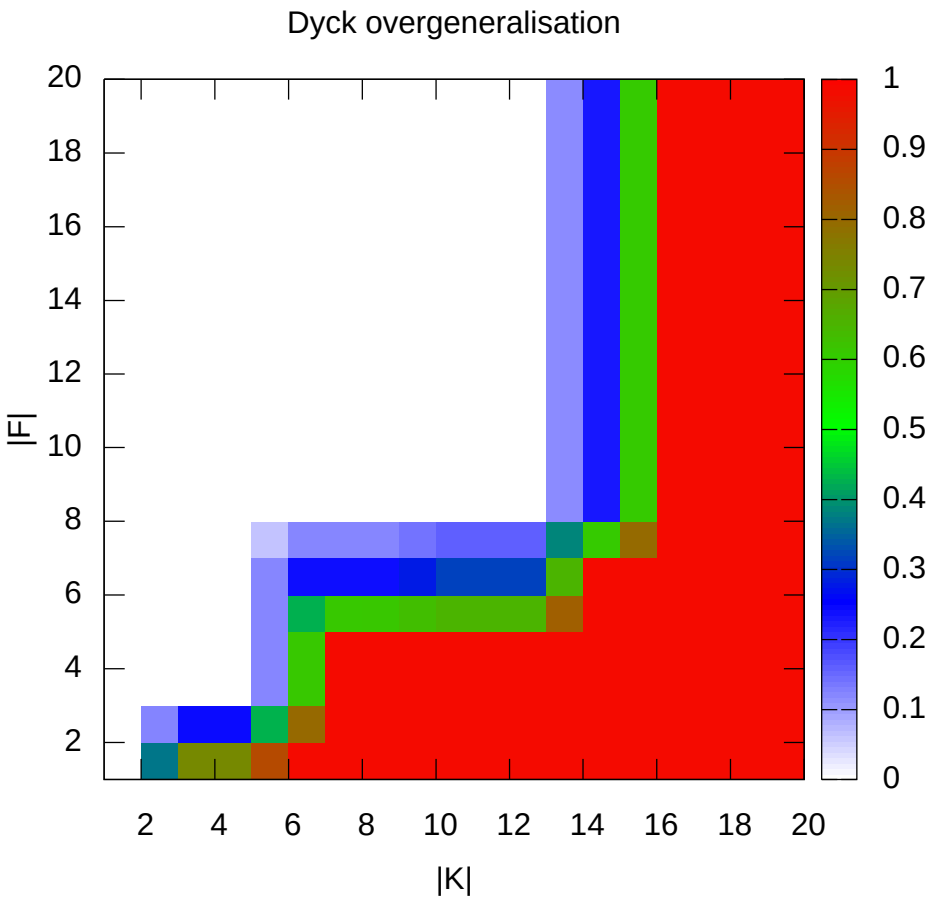
Not so obvious

As F increases the language decreases

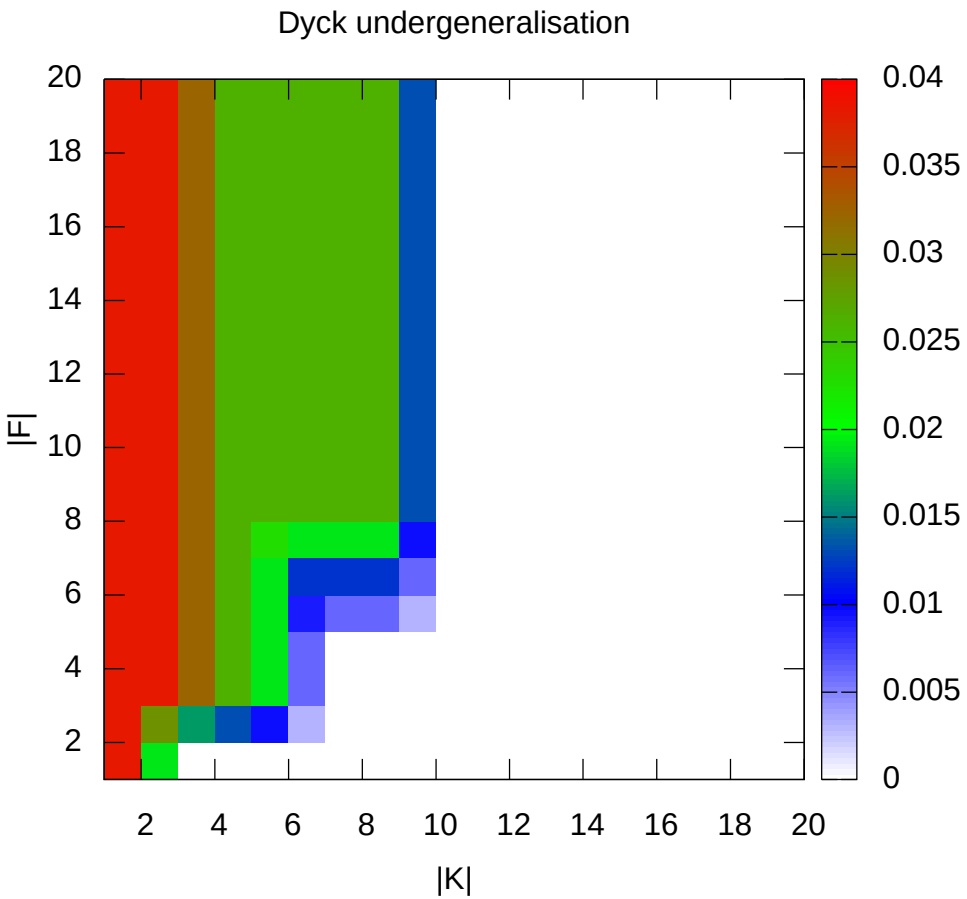
If $F \subseteq F^+$, then $L(G(K, L, F)) \supseteq L(G(K, L, F^+))$

- The features define the conditions under which we predict a feature on the head.
- if u has y and v has features z, then we predict that uv has some features x.

Dyck language
example



Dyck language
example



Algorithm

Goal:

- simple algorithm to prove correctness and polynomial efficiency
- no attempt at scalability

Basic idea

- If the language undergenerates, add some strings to the kernel.
  Go right.
- If the language overgenerates, add some contexts.
  Go up.

Algorithm

Input a sequence of strings $w_1, w_2 \ldots$

1. $D = \{w_1, \ldots, w_n\}$
2. Test set is $T = Con(D) \odot Sub(D)$
3. For every $w \in T$
   1. If $w \in L$ but not in current hypothesis; add strings to K, and add contexts to F
   2. If $w \notin L$ but is in current hypothesis: add contexts to F

This means we are getting closer, but will it converge?

- If it has the Finite Context Property, then we can get zero overgeneralisation
- If it has the Finite Kernel property, then we can get zero undergeneralisation

Algorithm

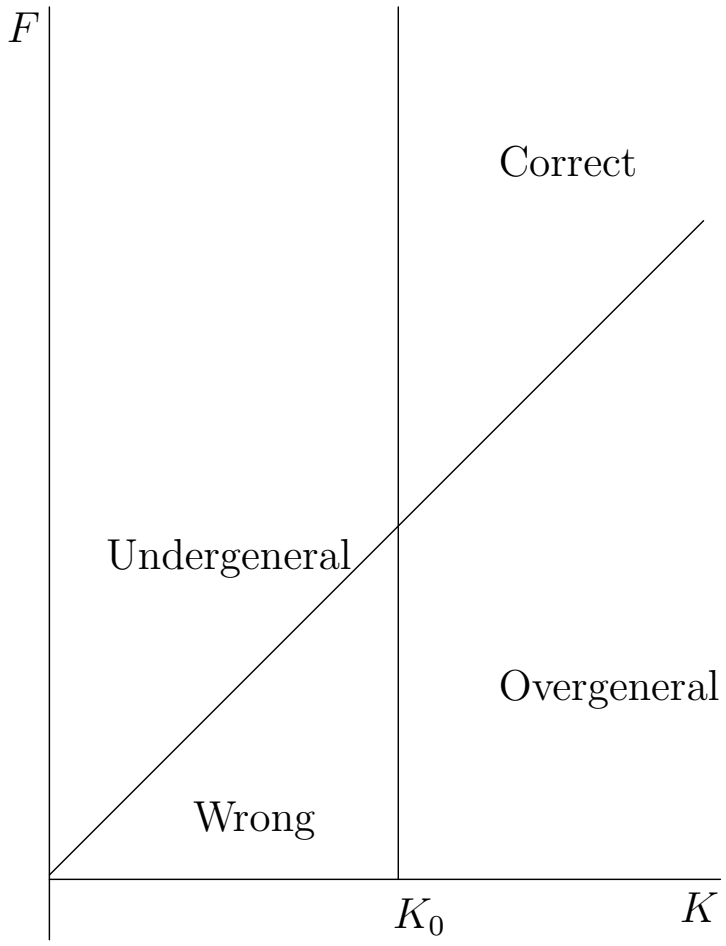
Input a sequence of strings $w_1, w_2 \ldots$

1. $D = \{w_1, \ldots, w_n\}$
2. Test set is $T = Con(D) \odot Sub(D)$
3. For every $w \in T$
   1. If $w \in L$ but not in current hypothesis; add strings to K, and add contexts to F
   2. If $w \notin L$ but is in current hypothesis: add contexts to F

This means we are getting closer, but will it converge?

- If it has the Finite Context Property, then we can get zero overgeneralisation
- If it has the Finite Kernel property, then we can get zero undergeneralisation

Diagram



Finite Context Property

Definition

A string u in a language L has the finite context property (FCP) if there is a finite set of contexts $F_u \subseteq C(u)$ such that

- For any v
- if $F_u \subseteq C(v)$ then $C(u) \subseteq C(v)$.

This is the inductive leap – from a finite set of evidence to an infinite set.

- Compare substitutable languages – any string in $C(u)$ is enough
- Compare Adriaans context-separability – $|F| = 1$

Finite Context Property

Definition

A string u in a language L has the finite context property (FCP) if there is a finite set of contexts $F_u \subseteq C(u)$ such that

- For any v
- if $F_u \subseteq C(v)$ then $C(u) \subseteq C(v)$.

This is the inductive leap – from a finite set of evidence to an infinite set.

- Compare substitutable languages – any string in $C(u)$ is enough
- Compare Adriaans context-separability – $|F| = 1$

Fiduciality

- More generally for a set of strings K, we say F is *fiducial* for K if for any u in K and for any v if $C(v) \supset C(u) \cap F$, then $C(v) \supset C(u)$.
- F needs to be a function of K; as K increases we need more features F.

Key lemma

If F is fiducial then the BFG predicts only correct features.
$f_G(w) \subset C(w) \cap F$
Proof:

- Definition of G_0 and $C(uv) \cap F \rightarrow C(u) \cap F, C(v) \cap F$
- Recursive definition of f_G
- Fiduciality

Scope of the FCP

- All regular languages have the FCP
- All substitutable languages have the FCP (of size 1)
- Therefore some non context free languages have the FCP.
  $\{w \in \{a, b, c\}^* || w|_a = |w|_b = |w|_c\}$
- There are CFLs that do not have the FCP.
  e.g. $L = \{a^n b | n > 0\} \cup \{a^n c^m | n > m > 0\}$

Natural languages have the FCP?

Finite kernel property

A finite set $K \subseteq \Sigma^*$ is a kernel for a language L, if for any set of features F, $L(G_0(K, F, L)) \supseteq L$.

- If we have a finite kernel then eventually our hypothesis will be big enough
- All regular languages have finite kernels
- There are CFLs that do not have a finite kernel

We expect to be able to weaken this requirement to include all CFGs with FCP.

Result

Theorem

The algorithm identifies in the limit the class of context free languages with FCP and FKP

- polynomial update time
- uses positive data plus polynomial calls to membership oracle

Includes all regular languages, disjoint palindrome languages, Dyck languages etc.

Outline

Concept lattices

Given a relation between two sets, or a bipartite graph

- Take the maximal bipartite cliques
- Maximal complete rectangles
- These form a complete lattice

Extensively studied in Formal Concept Analysis, Data-mining

Syntactic Concept Lattice

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 $S' = C$.

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

Example

$L = (ab)^* = \{\lambda, ab, abab, \dots\}$

- $S = \{a\}$
- $S' = \{(\lambda, b), (\lambda, bab), (ab, b) \dots\} = [\lambda, b]$
- $S'' = \{a, aba, ababa \dots\} = [a]$
- $S''' = S'$
- The concept is $\langle [a], [\lambda, b] \rangle$
- We can always find a concept $\langle S'', S' \rangle$ or $\langle C', C'' \rangle$

Basic properties

Partial order

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

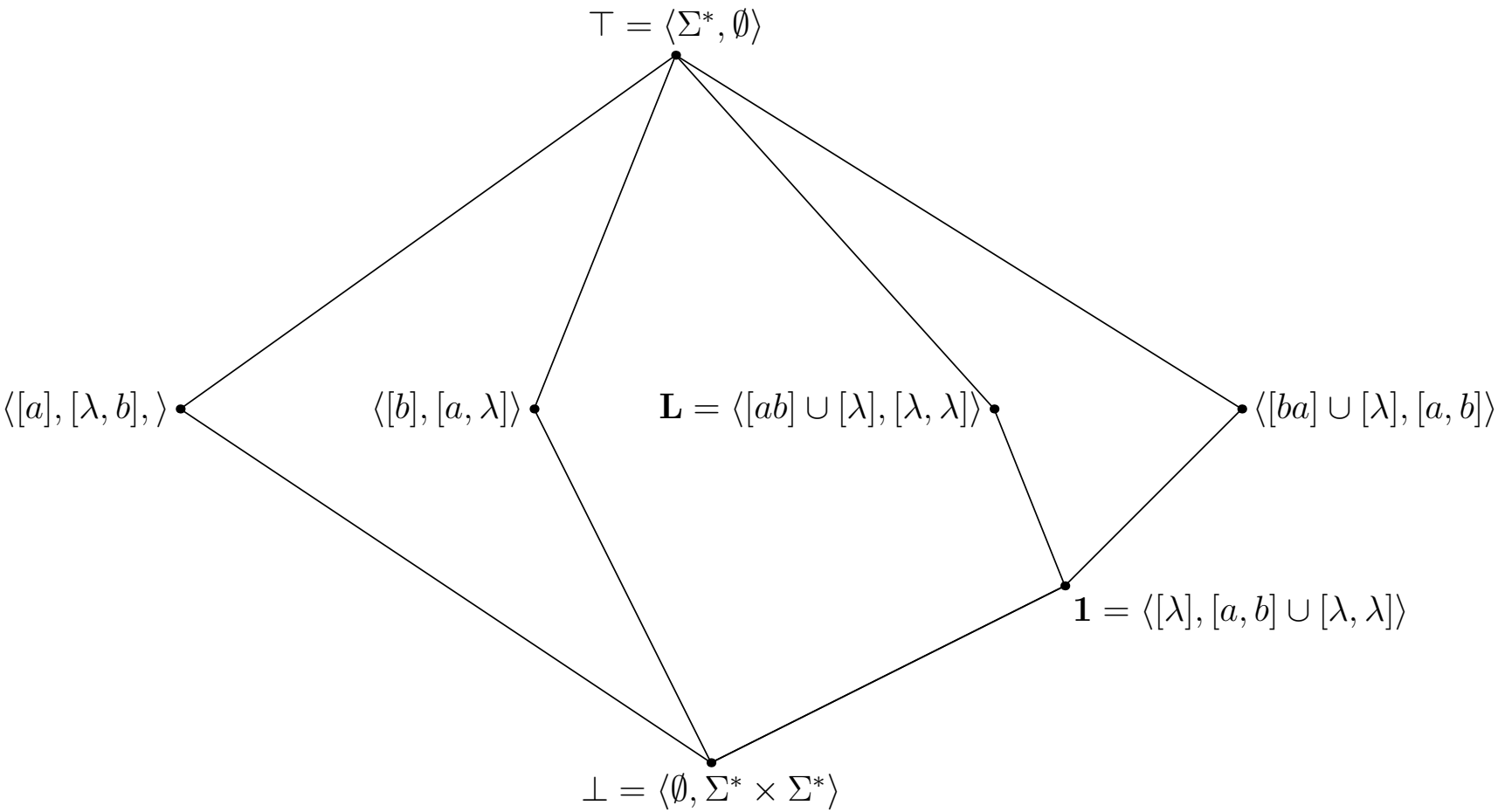
Lattice

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 $\bot = \langle \emptyset, \Sigma^* \times \Sigma^* \rangle$
- Unit $\mathbf{1} = \mathcal{C}(\lambda)$

$L = (ab)^*$



Concatenation

Definition

$$\langle S_x, C_x \rangle \circ \langle S_y, C_y \rangle = \langle (S_x S_y)'', (S_x S_y)' \rangle = \mathcal{C}(S_x S_y)$$

Example

$$\mathcal{C}(a) \circ \mathcal{C}(b) = \mathcal{C}(L)$$
$$\mathcal{C}(a) \circ \mathcal{C}(a) = \mathcal{C}(\{aa, abaa, \dots\}) = \top$$
$$\bot \circ X = \bot$$
$$\top \circ X = \top$$

Residuated lattice

This is a complete residuated lattice; written $\mathfrak{B}(L)$.

- Concatenation is a monoid: associative and with unit $\mathcal{C}(\{\lambda\})$.
- Lattice : $X \wedge Y$ is a greatest lower bound, $X \vee Y$ is a least upper bound; $X \leq Y$ is a partial order.
- The two operations interact properly, and we have binary operations $/$, and $\backslash$ such that
  $X \circ Y \leq Z$ iff $X \leq Z/Y$ iff $Y \leq X\backslash Z$

Concatenation

Definition

$$\langle S_x, C_x \rangle \circ \langle S_y, C_y \rangle = \langle (S_x S_y)'', (S_x S_y)' \rangle = \mathcal{C}(S_x S_y)$$

Example

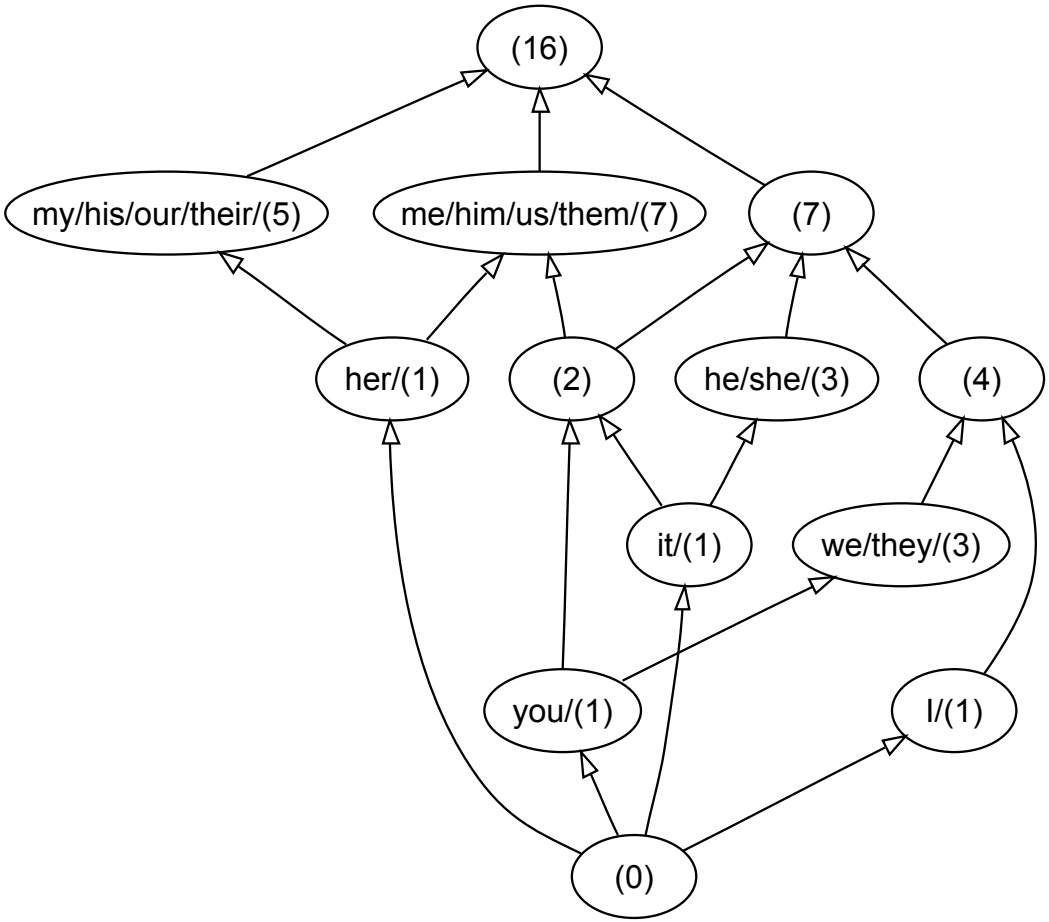
$$\mathcal{C}(a) \circ \mathcal{C}(b) = \mathcal{C}(L)$$
$$\mathcal{C}(a) \circ \mathcal{C}(a) = \mathcal{C}(\{aa, abaa, \dots\}) = \top$$
$$\bot \circ X = \bot$$
$$\top \circ X = \top$$

Learnable

Easy to compute from raw data

$\odot$	(λ,λ)	(λ,b)	(a,λ)	(a,b)	(b,λ)
λ	✓			✓	
a		✓			
b			✓		
aa					
ab	✓				
ba				✓	
aba		✓			
aab					
$abab$	✓				

Lexical hierarchies



Constructing a representation

Natural question

How do we use this to define a representation?

This lattice is a representation already!

Recursive definition

- If we know $\mathcal{C}(a)$ for all $a \in \Sigma$
- Concatenation defines the concept of a string uv in terms of the concepts of the strings u, v.
  $\mathcal{C}(uv) = \mathcal{C}(u) \circ \mathcal{C}(v)$.
- Then we can compute $\mathcal{C}(u)$ from its letters.
- If $\mathcal{C}(u)$ contains (λ, λ) then it is in the language.

Constructing a representation

Natural question

How do we use this to define a representation?

This lattice is a representation already!

Recursive definition

- If we know $\mathcal{C}(a)$ for all $a \in \Sigma$
- Concatenation defines the concept of a string uv in terms of the concepts of the strings u, v.
  $\mathcal{C}(uv) = \mathcal{C}(u) \circ \mathcal{C}(v)$.
- Then we can compute $\mathcal{C}(u)$ from its letters.
- If $\mathcal{C}(u)$ contains (λ, λ) then it is in the language.

Regular languages
Lattice is finite

Define a recursive function $\phi : \Sigma^* \to \mathfrak{B}(L)$

Definition

- $\phi(\lambda) = \mathcal{C}(\lambda)$
- $\phi(a) = \mathcal{C}(a)$ where $a \in \Sigma$
- $\phi(uv) = \phi(u) \circ \phi(v)$
- If $\phi(w) \leq \mathcal{C}(L)$ then $w \in L$.

As it is associative, any split u, v gives the same answer.

Correct

$\phi(w) = \mathcal{C}(w)$ and so $L(\mathfrak{B}(L)) = L$

Example

Negative

$w = abb$;
$\phi(w) = \phi(a) \circ \phi(b) \circ \phi(b) = \top$.
This is not in the language

Positive

$w = abab$;
$\phi(w) = \phi(a) \circ \phi(b) \circ \phi(a) \circ \phi(b) = \mathcal{C}(L)$
This is in the language.

Example

Negative

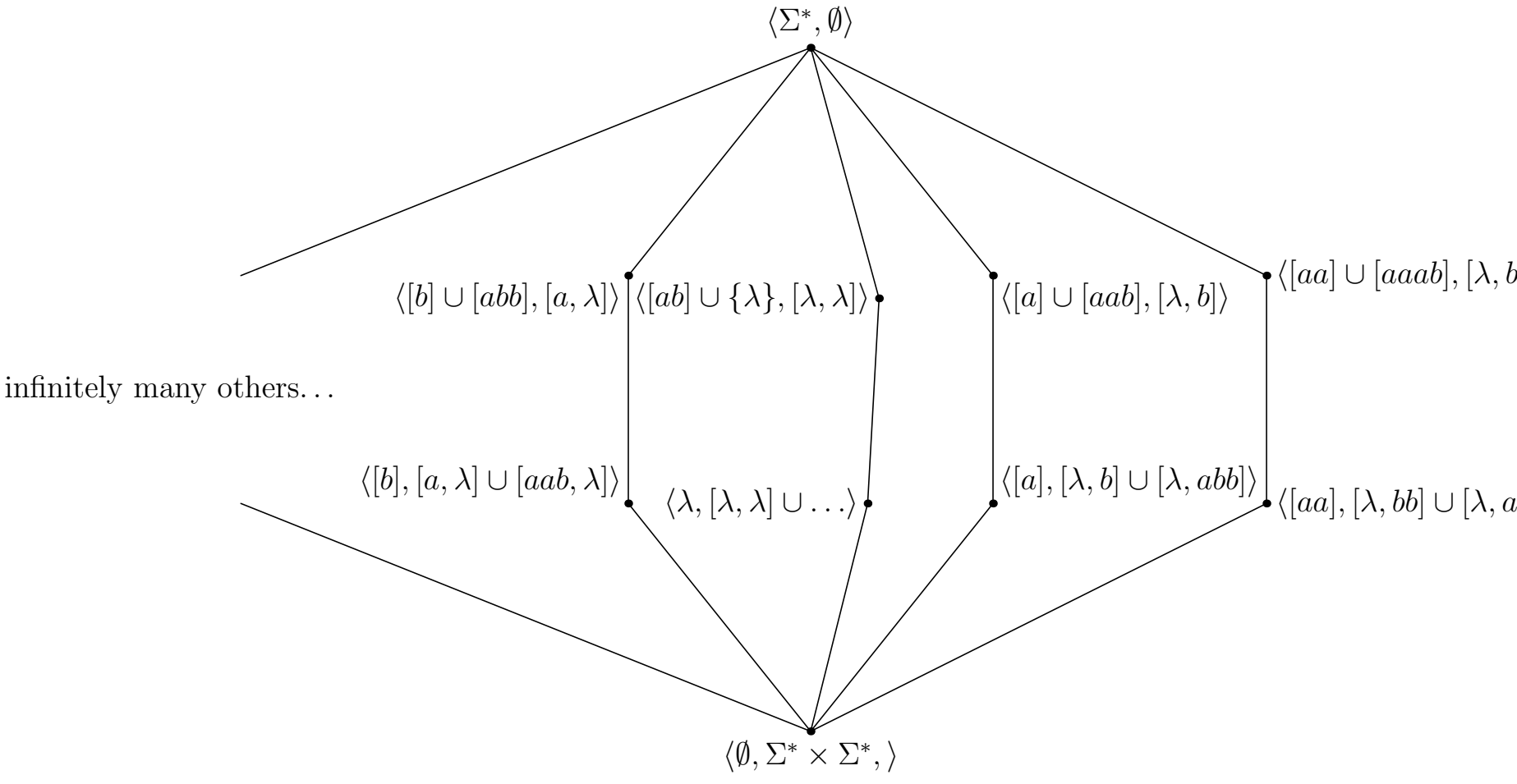
$w = abb$;
$\phi(w) = \phi(a) \circ \phi(b) \circ \phi(b) = \top$.
This is not in the language

Positive

$w = abab$;
$\phi(w) = \phi(a) \circ \phi(b) \circ \phi(a) \circ \phi(b) = \mathcal{C}(L)$
This is in the language.

Non-regular languages

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



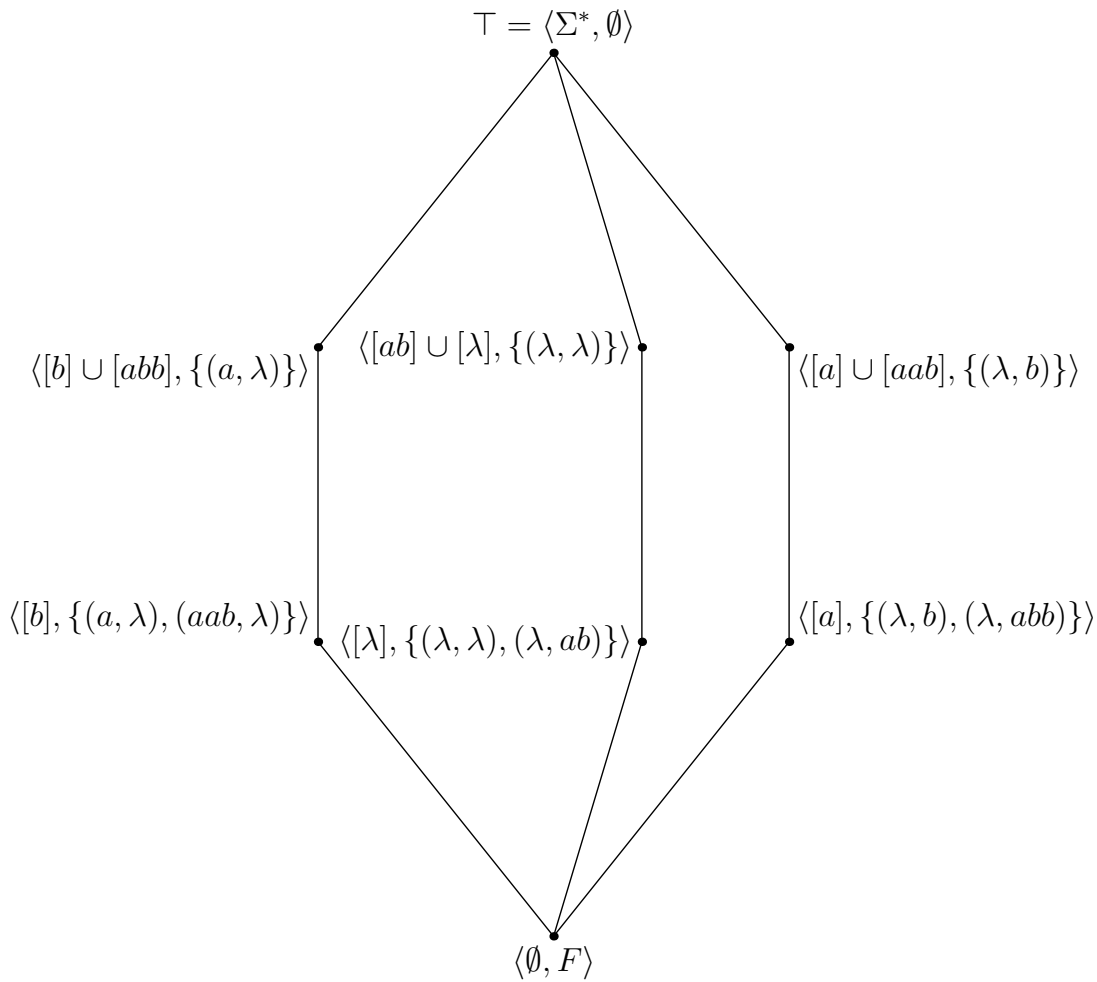
Finite feature set

Fix a finite set of contexts $F \subset \Sigma^* \times \Sigma^*$

- $(\lambda, \lambda) \in F$.
- Define a complete lattice $\mathfrak{B}(L, F)$.
- Concatenation defined as before
- No longer a residuated lattice
- Clearly this can have at most $2^{|F|}$ elements

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

- (λ, λ)
- (λ, b)
- (λ, abb)
- (a, λ)
- (aab, λ)
- (λ, ab)



Two problems

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

Not exact computation

- $\mathcal{C}(u) \circ \mathcal{C}(v) \neq \mathcal{C}(uv)$
  $\mathcal{C}(a^{100}) \circ \mathcal{C}(b^{100}) = \top \circ \top = \top \neq \mathcal{C}(a^{100}b^{100}) = \mathcal{C}(L)$
- But $\mathcal{C}(uv) \leq \mathcal{C}(u) \circ \mathcal{C}(v)$.

Not associative

- Estimate aab
- $\mathcal{C}(a) \circ (\mathcal{C}(a) \circ \mathcal{C}(b)) = \mathcal{C}(aab)$
- $(\mathcal{C}(a) \circ \mathcal{C}(a)) \circ \mathcal{C}(b) = \top \circ \mathcal{C}(b) = \top$

Compute an upper bound ϕ

Using the meet/greatest lower bound

But we know that

- $\mathcal{C}(aab) \leq \mathcal{C}(a) \circ \mathcal{C}(ab)$
- $\mathcal{C}(aab) \leq \mathcal{C}(aa) \circ \mathcal{C}(b)$

So

- $\mathcal{C}(aab) \leq (\mathcal{C}(a) \circ \mathcal{C}(ab)) \wedge (\mathcal{C}(aa) \circ \mathcal{C}(b))$

Generally

- $\mathcal{C}(w) \leq \bigwedge_{u,v} \mathcal{C}(u) \circ \mathcal{C}(v)$ for all $u, v \in \Sigma^+$ such that $w = uv$
- Define $\phi(w) = \bigwedge_{u,v:uv=w} \mathcal{C}(u) \circ \mathcal{C}(v)$

Representation

Data

- Lattice $\mathfrak{B}(L, F)$; operations $\circ, \wedge$
- Atomic elements $\mathcal{C}(a), \mathcal{C}(b), \ldots \mathcal{C}(\lambda), \mathcal{C}(\{(\lambda, \lambda)\})$

Definition

$\phi : \Sigma^* \rightarrow \mathfrak{B}(L, F)$.

- $\phi(\lambda) = \mathcal{C}(\lambda)$
- for all $a \in \Sigma$, (i.e. for all $w, |w| = 1$)
  $\phi(a) = \mathcal{C}(a)$
- for all w with $|w| > 1$,
  $\phi(w) = \bigwedge_{u,v \in \Sigma^+ : uv = w} \phi(u) \circ \phi(v)$
- Define $L(\mathfrak{B}(L, F)) = \{w : \phi(w) \leq \mathcal{C}(\{(\lambda, \lambda)\})\}$

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. $L = L(\mathfrak{B}(L, F))$

Includes

1. all regular languages
2. all CFLs with the Finite Context Property
3. Some non context free languages – including something close to the MIX language.
4. Not all CFLs

Monotonicity lemma (I)

Subset

For all w we have $\phi(w) \geq \mathcal{C}(w)$
Therefore $L(\mathfrak{B}(L, F)) \subseteq L$

Increasing the feature set

Suppose $F \subseteq G$
Define natural map $f : \mathfrak{B}(L, G) \to \mathfrak{B}(L, F)$

Lemma

$f(\phi_G(w)) \leq \phi_F(w)$
Corollary: $L(\mathfrak{B}(L, F)) \subseteq L(\mathfrak{B}(L, G))$

As F increases the language increases . . .
. . . so any large enough set of features is ok.

Writing down a grammar

Assume a finite set of features F, and a membership oracle:

Lattice

We have a finite set of strings K
Define a lattice $\mathfrak{B}(K, L, F)$ as before
Forms a complete lattice.

Concatenation

$\langle S_1, C_1 \rangle \circ \langle S_2, C_2 \rangle$
has features $(S_1 S_2)' \cap F$ which may not be a concept.

Closed under concatenation

$\mathfrak{B}(K, L, F)$ is closed under concatenation, if concatenation is always defined.
We can always increase K until it is closed.

Writing down a grammar

Assume a finite set of features F, and a membership oracle:

Lattice

We have a finite set of strings K
Define a lattice $\mathfrak{B}(K, L, F)$ as before
Forms a complete lattice.

Concatenation

$\langle S_1, C_1 \rangle \circ \langle S_2, C_2 \rangle$
has features $(S_1 S_2)' \cap F$ which may not be a concept.

Closed under concatenation

$\mathfrak{B}(K, L, F)$ is closed under concatenation, if concatenation is always defined.
We can always increase K until it is closed.

Monotonicity lemma (II)

Increase K

$J \subset K$ we define the obvious map
$g : \mathfrak{B}(J, L, F) \rightarrow \mathfrak{B}(K, L, F)$

Lemma

$g(\phi_J(w)) \leq \phi_K(w)$
As K increases $L(\mathfrak{B}(K, L, F))$ decreases.
For sufficiently large K, $\mathfrak{B}(K, L, F)$ is isomorphic to $\mathfrak{B}(L, F)$.

Example

$L = \{a^n b^n | n \geq 0\} \setminus \{a^{100} b^{100}\}$.
If K is small: $L(\mathfrak{B}(K, L, F))$ will be $a^n b^n$
When K includes a^{100}, b^{100} the language will contract

Monotonicity lemma (II)

Increase K

$J \subset K$ we define the obvious map
$g : \mathfrak{B}(J, L, F) \to \mathfrak{B}(K, L, F)$

Lemma

$g(\phi_J(w)) \leq \phi_K(w)$
As K increases $L(\mathfrak{B}(K, L, F))$ decreases.
For sufficiently large K, $\mathfrak{B}(K, L, F)$ is isomorphic to $\mathfrak{B}(L, F)$.

Example

$L = \{a^n b^n | n \geq 0\} \setminus \{a^{100} b^{100}\}$.
If K is small: $L(\mathfrak{B}(K, L, F))$ will be $a^n b^n$
When K includes a^{100}, b^{100} the language will contract

Search problem is trivial

Search problem

We want $\mathfrak{B}(K, L, F)$ to define L.
How to find K and F?

Summary

If F is big enough then $\mathfrak{B}(L, F)$ defines the right language.
Then if K is big enough then $\mathfrak{B}(K, L, F)$ is correct.

Naive Algorithm

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

- If hypothesis is too big, then add strings to K
- If hypothesis is too small, add contexts to F

Outline

Chomsky Hierarchy

We move away from the Chomsky hierarchy.

Chomsky (1968/2006)

“The concept of "phrase structure grammar" was explicitly designed to express the richest system that could reasonable be expected to result from the application of Harris-type procedures to a corpus.”

- If distributional learning can’t work on CFGs, then this is a problem with CFGs.
- They were intended to be learnable – if they are not, then this is a problem with the representation.
- We still have structure in the data – the dependencies will in general be a DAG not a tree.

Statistical Modelling

Symbolic learning

We have efficient algorithms for constructing representations for non-regular languages

- Ignore problems of sparsity, noise etc
- Assume we have a membership oracle: Clark and Lappin (2009)

Probabilistic model

Input is a large noisy corpus

- Replace whole contexts by partial contexts– adjacent words or word classes.
- PLSA/LDA (or non-parametric variant) to identify clusters in the data.
- Model parameters can represent very specific or more abstract combinatorial properties

Conclusion

Jackendoff (2008)

1. Descriptive constraint: the class of languages must be sufficiently rich to represent natural languages
2. Learnability constraint: there must be a way for the child to learn these representations from the data available
3. Evolutionary constraint: it must not posit a rich, evolutionarily implausible language faculty

- An approach that potentially satisfies all three criteria.