A polynomial algorithm for the inference of context free languages

Alexander Clark [1], Rémi Eyraud [2], Amaury Habrard [2]

[1]Department of Computer Science
Royal Holloway, University of London

[2]Laboratoire d'Informatique Fondamentale
Université de Provence, Marseille

Summary

A distributional learning algorithm

- Positive unstructured data and membership queries
- Polynomial update time
- Correct for a large class of CFLs that includes all regular languages
- Use a context sensitive formalism
- Essentially trivial algorithm
- . . . and it works in practice.

Why?

research goal

Find a class of languages that is efficiently learnable and includes the natural languages

- First language acquisition (FLA)
- Positive data and MQ are a placeholder for a more realistic probabilistic learning model
- Highly expressive models
- Mildly context sensitive

How

Normal GI

- Given a class of representations $\mathcal{G}$
- Study its learnability under various paradigms

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

research strategy

Look for representations that are intrinsically learnable:

- real problems are computational not information theoretic
- use representations without hidden structure
- primitives must be observable
  - definable in purely language theoretic terms

How

Normal GI

- Given a class of representations $\mathcal{G}$
- Study its learnability under various paradigms

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

research strategy

Look for representations that are intrinsically learnable:

- real problems are computational not information theoretic
- use representations without hidden structure
- primitives must be observable
  - definable in purely language theoretic terms

Substring relations

Regular learning

Prefix suffix relation

$u \sim_L v$ iff $uv \in L$

$u^{-1}L = \{w | uw \in L\}$

Contexts

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

$(l, r) \odot u = lur$

Context-substring relation

$(l, r) \sim_L u$ iff $lur \in L$

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

Substring relations

Regular learning

Prefix suffix relation

$u \sim_L v$ iff $uv \in L$

$u^{-1}L = \{w | uw \in L\}$

Contexts

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

$(l, r) \odot u = lur$

Context-substring relation

$(l, r) \sim_L u$ iff $lur \in L$

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

Distributional learning

Normal GI

For a string u, we want to predict if $u \in L$

model function $u \rightarrow \{0, 1\}$

Distributional learning

model function $u \rightarrow C(u)$

more general

- as $(\lambda, \lambda) \odot u = u$
- so $(\lambda, \lambda) \in C(u)$ iff $u \in L$

Distributional learning

Normal GI

For a string u, we want to predict if $u \in L$
model function $u \rightarrow \{0, 1\}$

Distributional learning

model function $u \rightarrow C(u)$
more general

- as $(\lambda, \lambda) \odot u = u$
- so $(\lambda, \lambda) \in C(u)$ iff $u \in L$

Two problems with modelling $C(u)$

1. Number of strings u is infinite:
   - Take some finite set of primitive elements (strings) K if $u \in K$ we know about $C(u)$
   - Plus some way of computing $C(uv)$ from its parts $C(u), C(v)$
2. $C(u)$ may be infinite if L is infinite:
   - We need some finite representation of C

Two problems with modelling $C(u)$

1. Number of strings u is infinite:
   - Take some finite set of primitive elements (strings) K if $u \in K$ we know about $C(u)$
   - Plus some way of computing $C(uv)$ from its parts $C(u), C(v)$
2. $C(u)$ may be infinite if L is infinite:
   - We need some finite representation of C

Solution 1

Clark & Eyraud (2005), Clark (2006), Yoshinaka (2008)

If $C(u) = C(u')$ and $C(v) = C(v')$ then $C(uv) = C(u'v')$

Syntactic congruence

$u \equiv_L v$ iff $C(u) = C(v)$

Write $[u]$ for equivalence class of u.

Finite representation/primitive elements

Congruence classes of observed substrings $[u]$

Basic rules give a CFG in CNF

$[uv] \to [u][v]$ and $[a] \to a$

If you can tell whether $C(u) = C(v)$ then learning is trivial!

Problems

- We model $C(u)$ as a finite unstructured set of congruence classes.
  - In real languages, no two words are exactly alike;
  - There are a lot of congruence classes;
  - Learning is very slow, and it is hard to get it right.
- The congruence classes are *sets of contexts* so the right structure is a lattice. (Sestier, 1960)

Example

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

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

Problems

- We model $C(u)$ as a finite unstructured set of congruence classes.
  - In real languages, no two words are exactly alike;
  - There are a lot of congruence classes;
  - Learning is very slow, and it is hard to get it right.
- The congruence classes are *sets of contexts* so the right structure is a lattice. (Sestier, 1960)

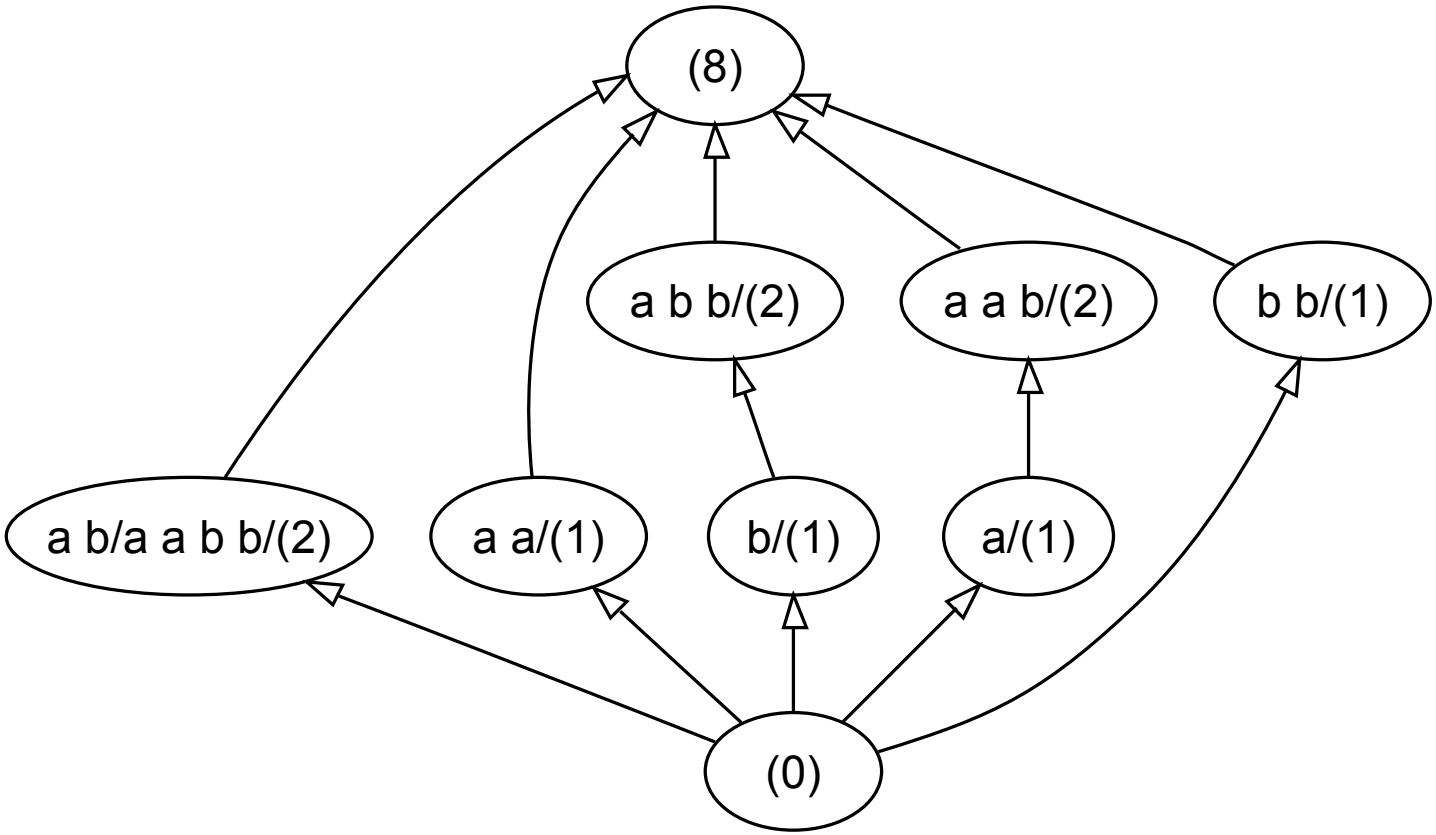
Example

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

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

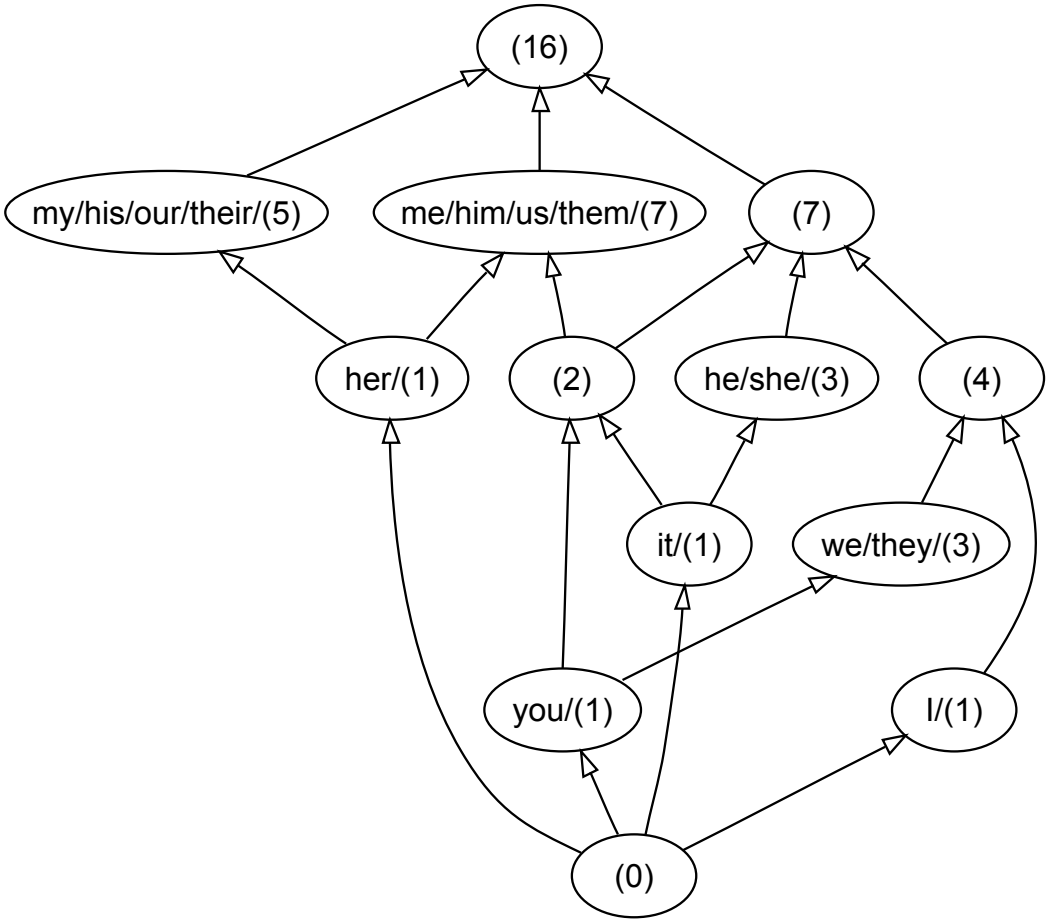
Lattice structure

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



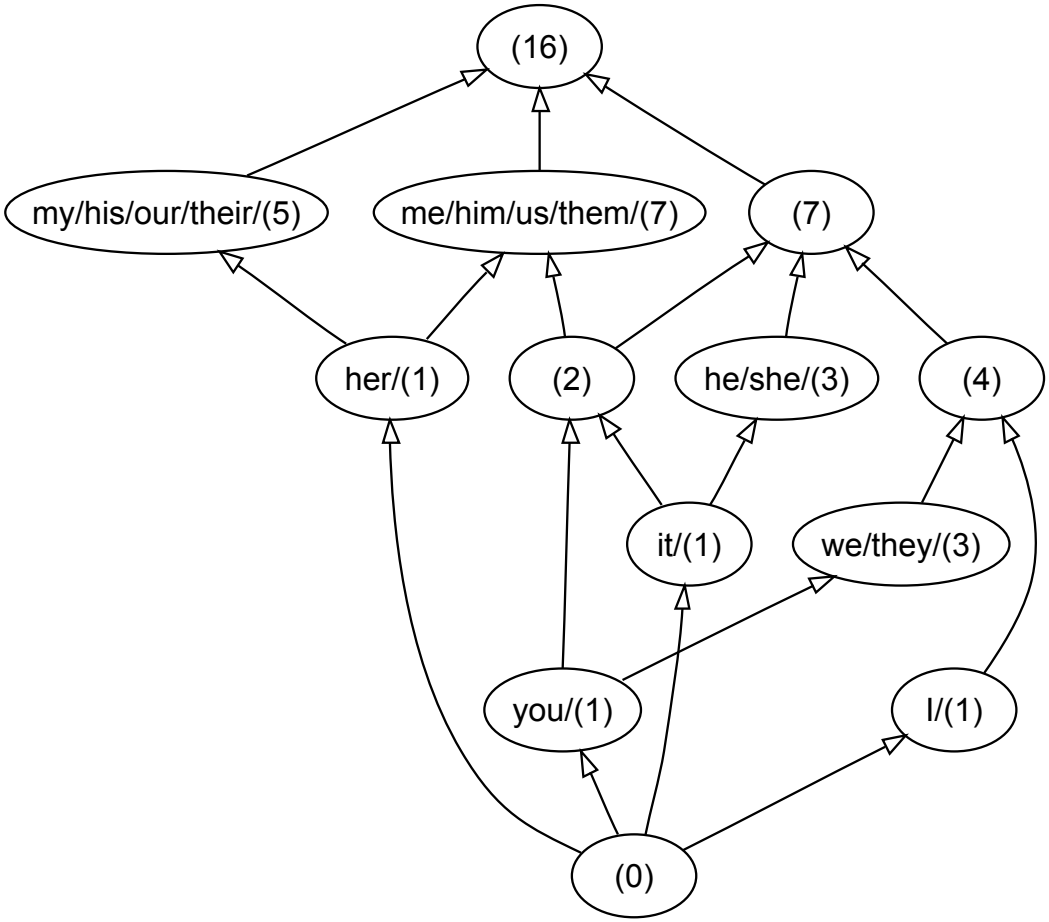
$C(a) \supset C(aab)$ but $(\lambda, abb) \in C(a) \setminus C(aab)$.

Lexical hierarchies



Exponentially many congruence classes!

Lexical hierarchies



Exponentially many congruence classes!

Solution 2

How to represent the lattice of distributions?

- Take a finite set of contexts F, and consider $C(w) \cap F$.
- Now we have $2^{|F|}$ congruence classes!
- Underlying lattice is not distributive but it doesn't matter.
- Underlying lattice is often infinite (if language is not regular) but that doesn't matter either

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

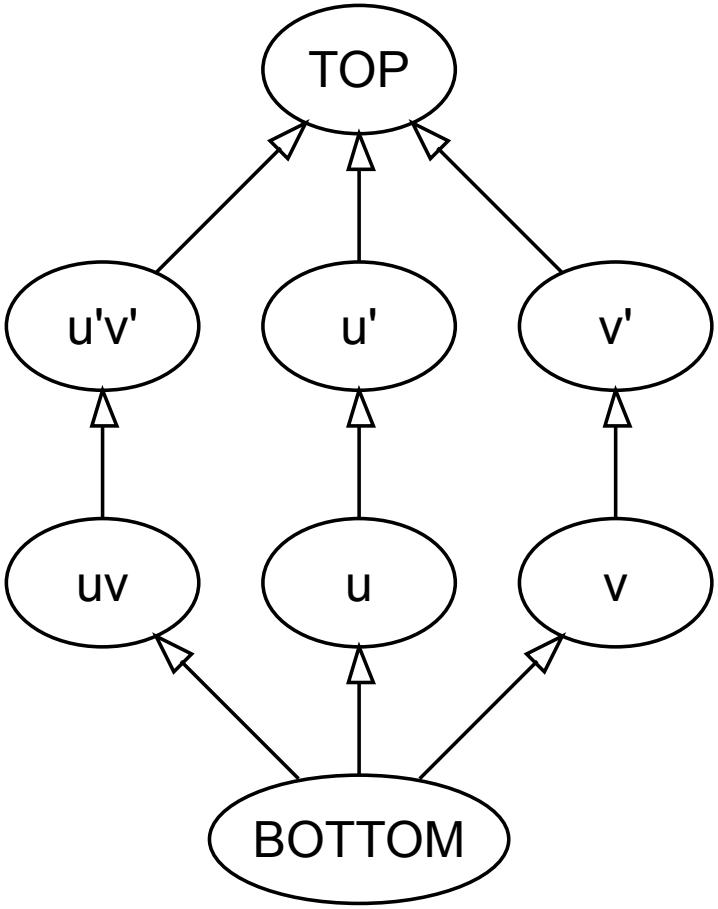
We define a formalism that directly uses this lattice structure:

Lemma

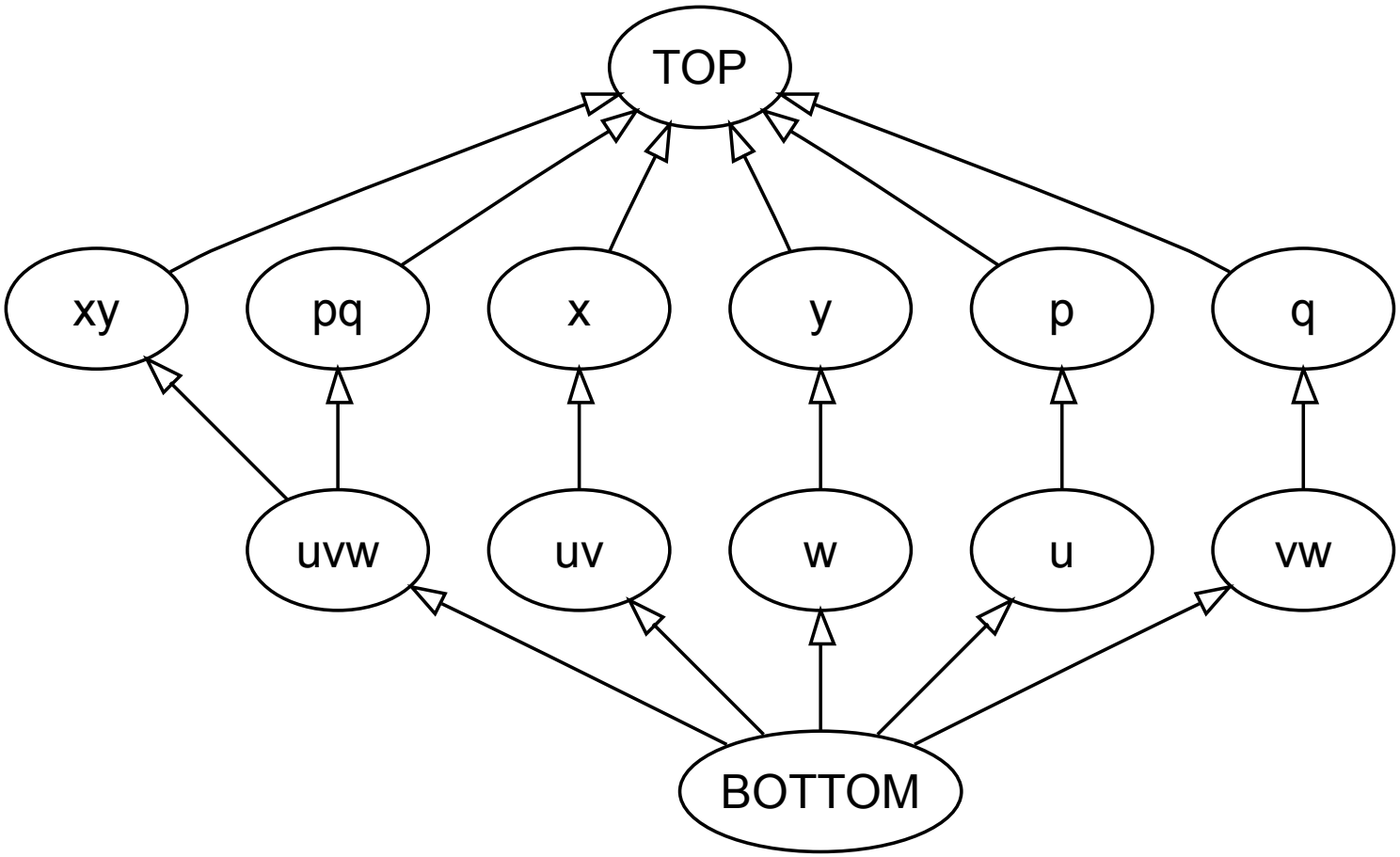
For any language L and for any strings u, u', v, v' if $C(u) \supseteq C(u')$ and $C(v) \supseteq C(v')$, then $C(uv) \supseteq C(u'v')$.

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

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 Range Concatenation Grammars (Boullier, 2000)

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

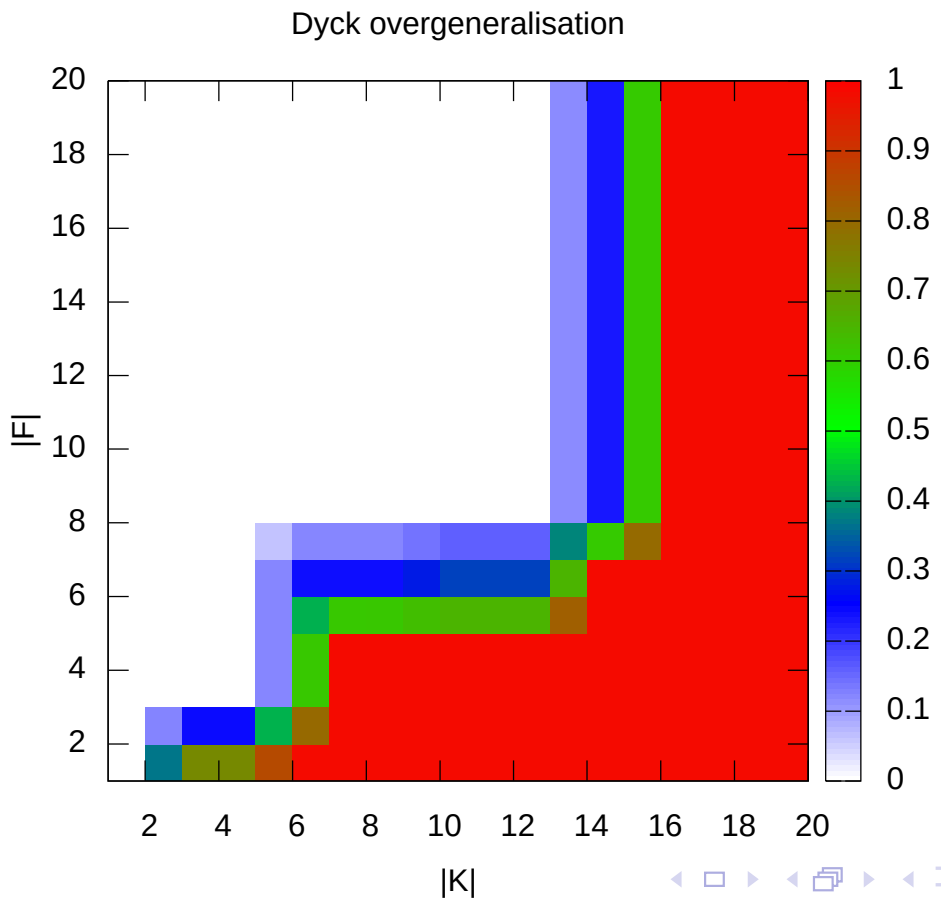
As F increases the language decreases

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.

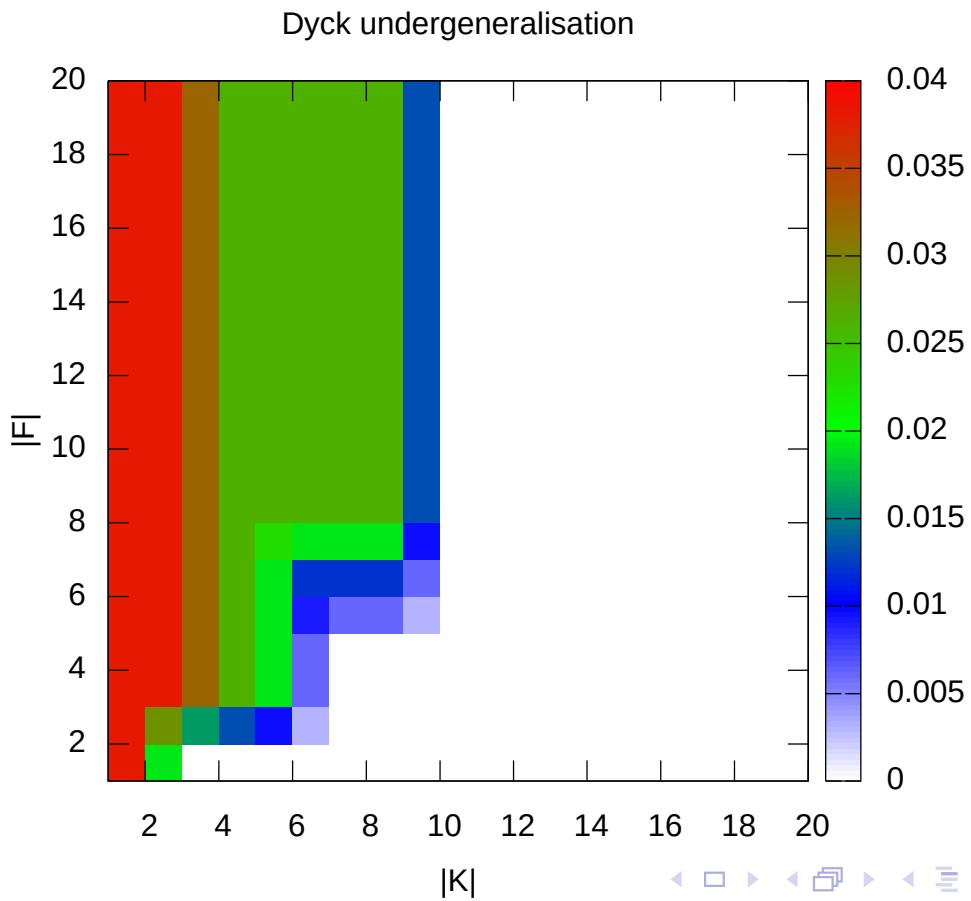
Dyck language

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 $w \in T$
   - 3.1 If $w \in L$ but not in current hypothesis; add strings to K, and add contexts to F
   - 3.2 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

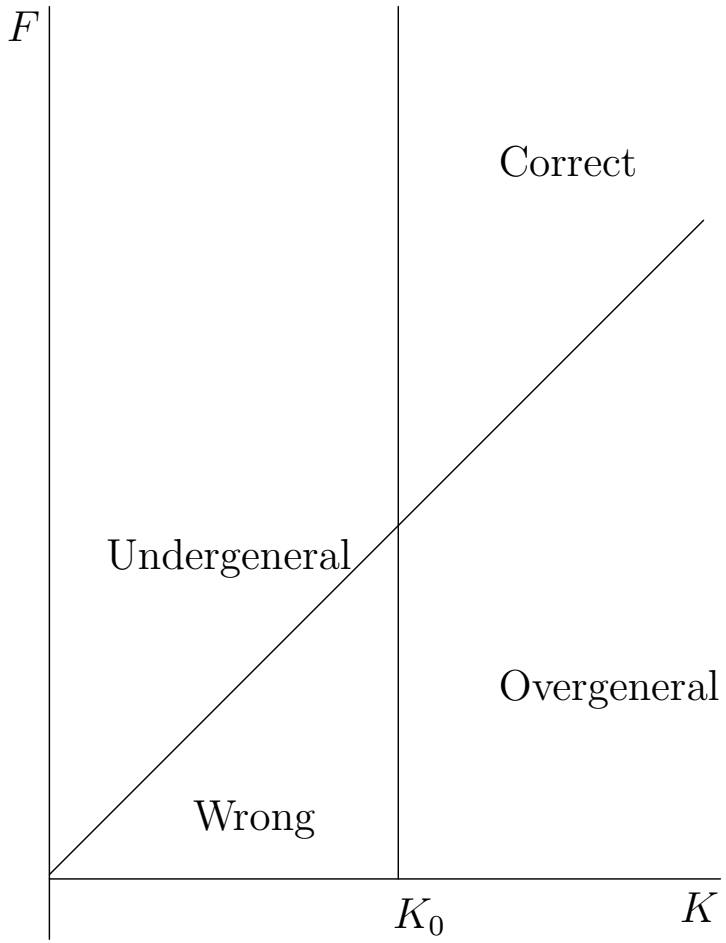
Input a sequence of strings $w_1, w_2 \ldots$

1. $D = \{w_1, \ldots, w_n\}$
2. Test set is $T = Con(D) \odot Sub(D)$
3. For every $w \in T$
   - 3.1 If $w \in L$ but not in current hypothesis; add strings to K, and add contexts to F
   - 3.2 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 (Lemma 8)

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.

Result

Theorem

The algorithm identifies in the limit the class of 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.

Future work

- Rapid generalisation
  - Use more abstract rules: $f \longrightarrow x \cap x', y \cap y'$.
  - Polynomial characteristic set
  - Context sensitive languages
- Probabilistic model for PAC result
- Features that are sets of contexts
- Natural language experiments

Conclusions

1. Distributional learning can be generalised to model the context-substring lattice.
2. This requires a switch to a context sensitive representation directly based on the lattice.
3. This gives rise to efficient algorithms for learning context free and potentially context sensitive languages that have a fairly weak property: the FCP.
4. This is linguistically very interesting, as natural languages seem to have the FCP, and are mildly context sensitive.