Exact query learning of regular and context-free grammars.

Alexander Clark

Department of Philosophy
King's College London
alexsclark@gmail.com

Turing Institute, September 2017

Outline

1. Exact query learning
2. Angluin's algorithm for learning DFAs.
   (Actually a much less elegant version)
3. An extension to learning CFGs.

Instance space: $\mathcal{X}$

Infinite and continuous

$\mathbb{R}^n$: Real valued vector spaces: physical quantities

Finite and discrete

$\{0, 1\}^n$ Bit strings

'Discrete Infinity'

Discrete combinatorial objects:

Σ^*: strings, trees, graphs, ...

GRAMMATICAL INFERENCE

Strings of what?

- words
- characters or phonemes
- user interface actions
- robot actions
- states of some computational device . . .

Concepts are formal languages: sets of strings

1. *a*, *bcd*, *ef*
2. *ab*, *abab*, *ababab*, ...
3. *xabx*, *xababx*, ..., *yaby*, *yababy*, ...
4. *ab*, *aabb*, *aaabbb*, ...
5. *ab*, *aabb*, *abab*,*aababb*, ...
6. *abcd*, *abbbcddd*, *aabccd*, ...
7. *ab*, *ababb*, *ababbabbb*, ...

Concepts are formal languages: sets of strings

1. *a*, *bcd*, *ef* **Finite list**
2. *ab*, *abab*, *ababab*, . . . **Markov model/bigram**
3. *xabx*, *xababx*, . . . , *yaby*, *yababy*, . . . **Finite automaton**
4. *ab*, *aabb*, *aaabbb*, . . . **Linear CFG**
5. *ab*, *aabb*, *abab*,*aababb*, . . . **CFG**
6. *abcd*, *abbbcddd*, *aabccd*, . . . **Multiple CFG**
7. *ab*, *ababb*, *ababbabbb*, . . . **PMCFG**

Exact learning

Exact learning

Because we have a *set* of *discrete* objects it's not unreasonable to require exact learning.

Theoretical Guarantees

Moreover, we may *need* algorithms with some theoretical guarantees: proofs of their correctness.

Exact learning

Exact learning

Because we have a *set* of *discrete* objects it's not unreasonable to require exact learning.

Theoretical Guarantees

Moreover, we may *need* algorithms with some theoretical guarantees: proofs of their correctness.

Application domains:

- Software verification
- Models of language acquisition
- NLP (?)

Learning models

- Distribution free PAC model – too hard and not relevant
- Distribution learning PAC models.
- Identification in the limit from positive examples.
- Identification in the limit from positive and negative examples.

Minimally Adequate Teacher model

Information sources

Target T, Hypothesis H

- Membership Queries: take an arbitrary $w \in \mathcal{X}$:
  Is $w \in L(T)$?
- Equivalence queries:
  Is $L(H) = L(T)$?
  Answer: either yes or a counterexample in
  $L(H) \setminus L(T) \cup L(T) \setminus L(H)$

We require the algorithm to run in polynomial time: in size of target and size of longest counterexample.

Minimally Adequate Teacher model

Information sources

Target T, Hypothesis H

- Membership Queries: take an arbitrary $w \in \mathcal{X}$:
  Is $w \in L(T)$?
- Equivalence queries:
  Is $L(H) = L(T)$?
  Answer: either yes or a counterexample in
  $L(H) \setminus L(T) \cup L(T) \setminus L(H)$

We require the algorithm to run in polynomial time: in size of target and size of longest counterexample.
There is a loophole with this definition.

Equivalence queries?

- Not available in general
- Not computable in general (e.g. with CFGs); or computationally expensive.

But we can simulate it easily enough, if we can sample from the target and hypothesis.

Equivalence queries?

- Not available in general
- Not computable in general (e.g. with CFGs); or computationally expensive.

But we can simulate it easily enough, if we can sample from the target and hypothesis.

Extended EQs

Standardly we assume that the hypothesis must be in the class of representations that is learned. This is a problem later on, so we will allow *extended* EQs.

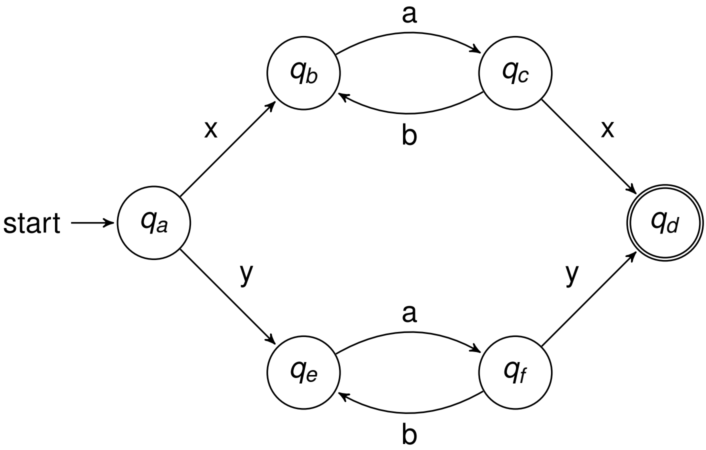
Example : Learning DFAs, but we allow EQs with NFAs.

Discussion

- An abstraction from the statistical problems of learning, that allow you to focus on the computational issues.
- Completely symmetrical between the language and its complement.

Deterministic Finite State Automaton

$xa(ba)^*x \cup ya(ba)^*y$



Myhill-Nerode theorem (1958)

Definition

Two strings u, v are right-congruent ($u \equiv_R v$) in a language L if for all strings w
$uw \in L$ iff $vw \in L$

Equivalently: define $u^{-1}L = \{w \mid uw \in L\}$.

$$u^{-1}L = v^{-1}L$$

- Clearly an equivalence relation.
- And a congruence in that if $u \equiv_R v$ then $ua \equiv_R va$

Canonical DFA

States correspond to equivalence classes!

String u

Equivalence class $[u] = \{v \mid u^{-1}L = v^{-1}L\}$

State should generate all strings in $u^{-1}L$

Two elements of the algorithm

1. Determine whether two prefixes are congruent.
2. Construct an automaton from the congruence classes we have so far identified.

Automaton construction

Data $xax, yay, xabax, yabay \in L_*$

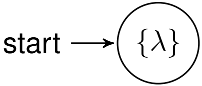
Automaton construction

Data $xax, yay, xabax, yabay \in L_*$
Some prefixes:
$\lambda, x, xa, xax, xab, xaba, xabax, y, ya, yay, yab, yaba, yabay$

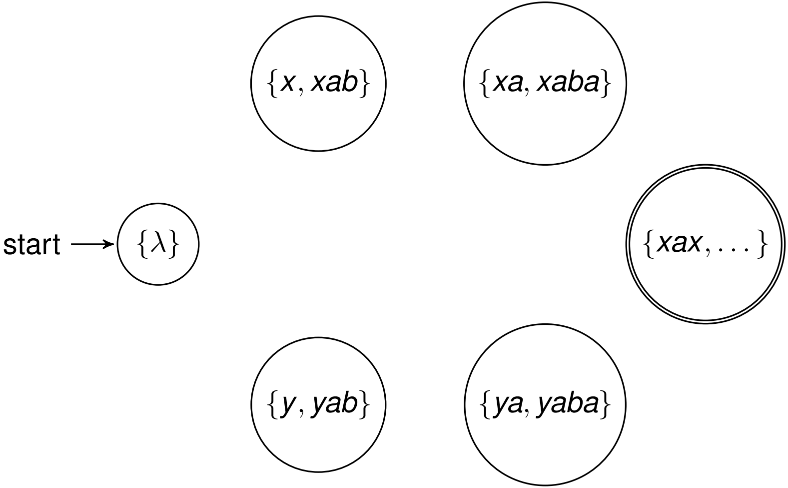
Automaton construction

Data $xax, yay, xabax, yabay \in L_*$
Some prefixes:
$\lambda, x, xa, xax, xab, xaba, xabax, y, ya, yay, yab, yaba, yabay$
Congruence classes: $\{\lambda\}$, $\{x, xab\}$, $\{xa, xaba\}$,
$\{xax, xabax, yay, yabay\}$, $\{y, yab\}$, $\{ya, yaba\}$

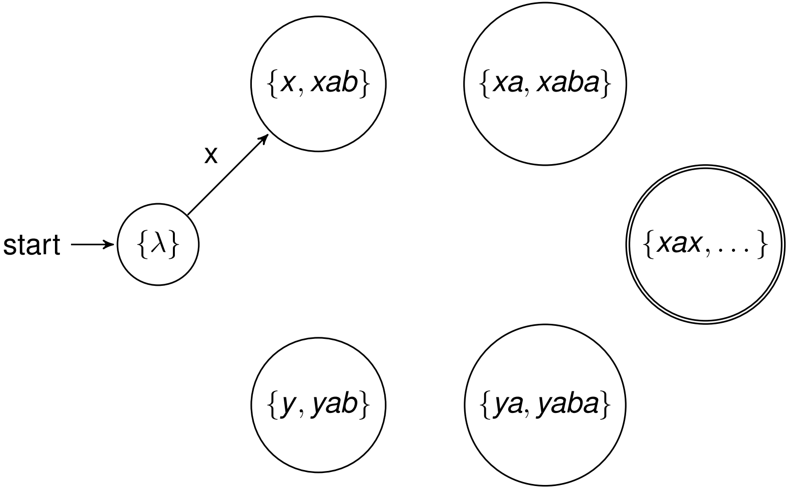
Initial state is the one containing λ



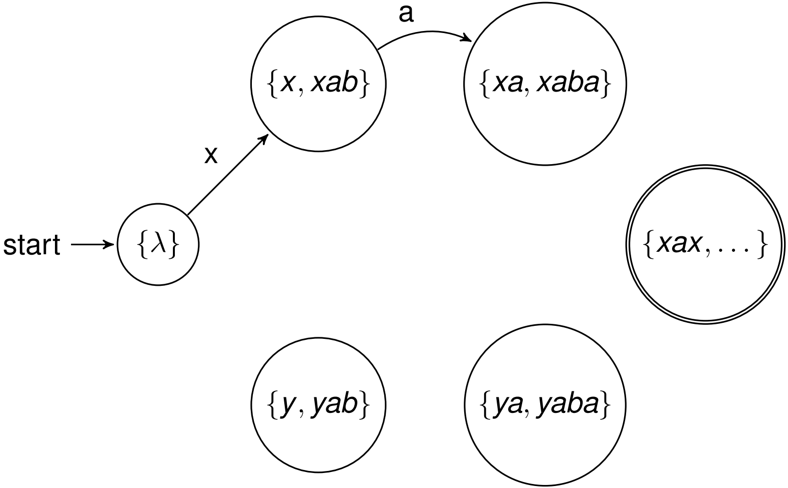
Final states are those containing strings in the language



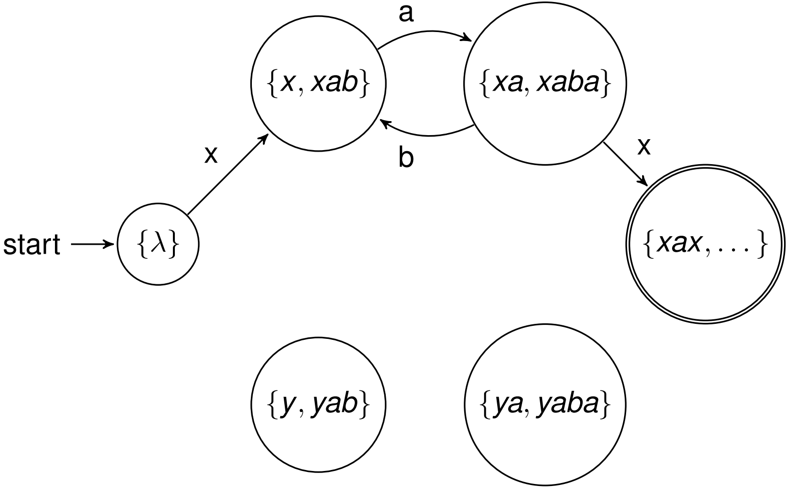
$\lambda \cdot x = x$ so add transition $\lambda \to x$ labeled with x

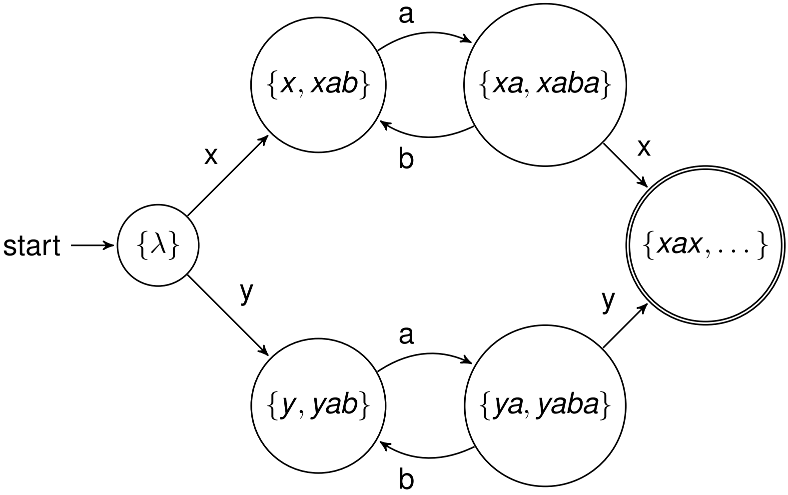


$x \cdot a = xa$ so add transition $x \to xa$ labeled with a



If $u \in q$ and $ua \in q'$ then add transition from $q \to q'$ labeled with a




start
{λ}
x
y
{x, xab}
{y, yab}
a
b
{xa, xaba}
{ya, yaba}
x
y
{xax, ...}

Method number 1

How to test $u^{-1}L = v^{-1}L$

- Assume that if $u^{-1}L \cap v^{-1}L \neq \emptyset$ then they are equal! (only true for "reversible' languages, [Angluin, 1982])
- Then if we observe uw and vw are both in the language, assume $u^{-1}L = v^{-1}L$.

$xax, xabax$ are both in the language so $x \equiv xab$ and $xa \equiv xaba$ and $xax \equiv xabax$ ...

Method number 2

How to test

$u^{-1}L = v^{-1}L$

Method number 2

- Assume data is generated by some probabilistic automaton.
- Use a statistical measure of distance between $P(uw|u)$ and $P(vw|v)$ (e.g L_∞ norm)
- PAC learning PDFA [Ron et al., 1998], [Clark and Thollard, 2004]

Method number 3: Angluin style algorithm

How to test

$u^{-1}L = v^{-1}L$

- If we have MQs we can take a finite set of suffixes J and test whether
  $u^{-1}L \cap J = v^{-1}L \cap J$
- If there are a finite number of classes, then there is a finite set which will give correct answers.

Data structure

Maintain an observation table:

- **Rows** : K is a set of prefixes
- **Columns** J is a set of suffixes that we use to test equivalence of residuals of rows.
- **Entries** 0 or 1 depending on whether the concatenation is in or not.

Data structure

Maintain an observation table:

- Rows : K is a set of prefixes
- Columns J is a set of suffixes that we use to test equivalence of residuals of rows.
- Entries 0 or 1 depending on whether the concatenation is in or not.

Hankel matrix in spectral approaches

$$H = \mathbb{R}^{\Sigma^* \times \Sigma^*}$$

where $H[u, v] = 1$ if $uv \in L_*$ and 0 otherwise

Observation table example

	λ	x	ax	xax
λ	0	0	0	1
x	0	0	1	0
xa	0	1	0	0
xax	1	0	0	0
xab	0	0	1	0
xaba	0	1	0	0
xabax	1	0	0	0

Observation table example

	λ	x	ax	xax
λ	0	0	0	1
x	0	0	1	0
xab	0	0	1	0
xa	0	1	0	0
xaba	0	1	0	0
xax	1	0	0	0
xabax	1	0	0	0

Observation table example

	λ	x	ax	xax
λ	0	0	0	1
x	0	0	1	0
xab	0	0	1	0
xa	0	1	0	0
xaba	0	1	0	0
xax	1	0	0	0
xabax	1	0	0	0

Monotonicity properties

- Increasing rows increases the language hypothesized.
- Increasing columns decreases the language hypothesized.

Algorithm I

1. Start with $K = J = \{\lambda\}$.
2. Fill in OT with MQs
3. Construct automaton.
4. Ask an EQ.
5. If it is correct, terminate
6. Otherwise process the counterexample and goto 2.

Algorithm II

If we have a positive counterexample w

Add every prefix of w to the set of prefixes K.

If we have a negative counterexample w

Naive: Add all suffixes of w to J.

Smart: Walk through the derivation of w and find a single suffix using MQs.

Proof

- If we add rows and keep the columns the same, then we will increase the states and transitions will monotonically increase.
- If we add columns and keep the rows the same, the language defined will monotonically decrease.

Angluin's actual algorithm

Two parts of the table:

- K
- $K \cdot \Sigma$

Ensure that the table is

Closed
: every row in $K \cdot \Sigma$ is equivalent to a row in K

Consistent
: the resulting automaton is deterministic.

Minimize the number of EQs which are in practice more expensive than MQs.

Later developments

- Algorithmic improvements by [Kearns and Vazirani, 1994], [Balcázar et al., 1997]
- Extension to regular tree languages [Drewes and Högberg, 2003]
- Extension to a slightly nondeterministic automata [Bollig et al., 2009]

Context free grammars

variant of Chomsky normal form

- A set of nonterminals V
- A set of start symbols I
  (normally we just have one start symbol S)
- Productions:

Binary	$A \to BC$
Lexical	$A \to a$
	(also $A \to \lambda$ sometimes)

We will write $A \stackrel{*}{\Rightarrow} w$ if we can derive w from A.

Contexts and substrings

Context (or *environment*)

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

Given a language $L \subseteq \Sigma^*$.

Distribution of a string

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

Analogous to $u^{-1}L$

Important difference with regular languages

Regular languages

- Prefixes and suffixes are both strings.
- Swapping them is boring: we just get an automaton which processes from right to left.

Context-free grammars

- Substrings and contexts are of different types.
- Swapping them gives two qualitatively different algorithms:

Primal	[Clark, 2010]
Dual	[Shirakawa and Yokomori, 1993]

Syntactic congruence

Replace the right congruence with the two-sided congruence.

Definition

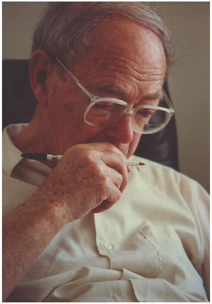
$u \equiv_L v$ iff $C_L(u) = C_L(v)$
This is a congruence:
$u \equiv_L v$ implies $uw \equiv_L vw$ and $wu \equiv_l wv$

There are an infinite number of classes if L is not regular!

Distributional Learning

Zellig Harris (1949, 1951)

Here as throughout these procedures X and Y are substitutable if for every utterance which includes X we can find (or gain native acceptance for) an utterance which is identical except for having Y in the place of X



Learnable class

Language class

Class of all CFGs where the non-terminals generate strings that are congruent

- If $A \stackrel{*}{\Rightarrow} u$ and $A \stackrel{*}{\Rightarrow} v$ then $u \equiv_L v$

Learnable class

Language class

Class of all CFGs where the non-terminals generate strings that are congruent

- If $A \stackrel{*}{\Rightarrow} u$ and $A \stackrel{*}{\Rightarrow} v$ then $u \equiv_L v$

- Includes all regular languages
- Some non-regular languages (Dyck language)
- Not all context-free languages (palindrome language)
- (Roughly) NTS languages [Boasson and Sénizergues, 1985]

Basic representational idea

Representation

Nonterminals correspond to congruence classes

Basic representational idea

Representation

Nonterminals correspond to congruence classes

String u

Equivalence class $[u] = \{v \mid C_L(u) = C_L(v)\}$

Nonterminal should generate all strings in $[u]$

1. Test whether $u \equiv v$
2. Build a grammar from the congruence classes.

Build grammar

X, Y, Z are sets of substrings.

Branching rules

If $u \in Y$, $v \in Z$ and $uv \in X$
Add production $X \rightarrow YZ$

Lexical rules

If $a \in X$
Add production $X \rightarrow a$

Initial symbols

If X has context (λ, λ)
Add X to set of initial symbols
(Equivalently $S \rightarrow X$)

Three ways of testing

1. Assume that if $lur, lvr \in L$ then $u \equiv v$ (Substitutable languages [Clark and Eyraud, 2007])
2. Assume data generated by a PCFG [Clark, 2006], [Shibata and Yoshinaka, 2013]
3. Angluin style approach [Clark, 2010]

Test

How to test if $C_L(u) = C_L(v)$?
Pick a finite set of contexts J
Test $C_L(u) \cap J = C_L(v) \cap J$ using MQs

Observation table

We fill in the OT with MQs as normal.

Rows A set of substrings K – which includes Σ and λ

Columns A set of contexts J which includes (λ, λ)

Equivalence

$u \sim_J v$ iff $C_L(u) \cap J = C_L(v) \cap J$
Equal rows

Example

Dyck language

Language of well-matched brackets

$$\lambda, ab, abab, aabb, abaabb, \ldots$$

Example

Dyck language

		J		
		(λ, λ)	(a, λ)	(λ, b)
K	λ	1	0	0
	a	0	0	1
	b	0	1	0
	ab	1	0	0
	aab	0	0	1
	abb	0	1	0
	aa	0	0	0
	ba	0	0	0
	bb	0	0	0
	bab	0	1	0
	aba	0	0	1
	$abab$	1	0	0

Example

Dyck language

	(λ, λ)	(a, λ)	(λ, b)
λ	1	0	0
ab	1	0	0
abab	1	0	0
a	0	0	1
aab	0	0	1
aba	0	0	1
b	0	1	0
abb	0	1	0
bab	0	1	0
aa	0	0	0
ba	0	0	0
bb	0	0	0

Example

Dyck language

	(λ, λ)	(a, λ)	(λ, b)	Non-terminals
λ	1	0	0	
ab	1	0	0	$\to S \in I$
$abab$	1	0	0	
a	0	0	1	
aab	0	0	1	$\to A$
aba	0	0	1	
b	0	1	0	
abb	0	1	0	$\to B$
bab	0	1	0	
aa	0	0	0	
ba	0	0	0	Discard
bb	0	0	0	

Example

Three non-terminals

- $S = \{\lambda, ab, abab\}$
- $A = \{a, aab, aba\}$
- $B = \{b, abb, bab\}$

- $A \to a, B \to b, S \to \lambda$

Example

Three non-terminals

- $S = \{\lambda, ab, abab\}$
- $A = \{a, aab, aba\}$
- $B = \{b, abb, bab\}$

- $A \to a, B \to b, S \to \lambda$
- $a \in A, b \in B, ab \in S$ so $S \to AB$

Example

Three non-terminals

- $S = \{\lambda, ab, abab\}$
- $A = \{a, aab, aba\}$
- $B = \{b, abb, bab\}$

- $A \to a, B \to b, S \to \lambda$
- $a \in A, b \in B, ab \in S$ so $S \to AB$
- $a \in A, ab \in S, aab \in A$ so $A \to AS$

Example

Three non-terminals

- $S = \{\lambda, ab, abab\}$
- $A = \{a, aab, aba\}$
- $B = \{b, abb, bab\}$

- $A \to a, B \to b, S \to \lambda$
- $a \in A, b \in B, ab \in S$ so $S \to AB$
- $a \in A, ab \in S, aab \in A$ so $A \to AS$
- $A \to SA, B \to SB, B \to BS, S \to SS$

Note that this grammar defines the Dyck language.

Closure and Consistency

Two differences from LSTAR

Closure

For non-regular languages, the number of congruence classes will be infinite.
So there will be classes in KK that are not in K

Consistency

If $u \sim_J u'$ and $v \sim_J v'$ implies
$uv \sim_J u'v'$ then it is *consistent*

Closure and Consistency

Two differences from LSTAR

Closure

For non-regular languages, the number of congruence classes will be infinite.
So there will be classes in KK that are not in K

Consistency

If $u \sim_J u'$ and $v \sim_J v'$ implies
$uv \sim_J u'v'$ then it is *consistent*
If it is not consistent then we need more contexts
(Optional: possibly exponential)

Exponential *thickness*

The shortest string in the language may be exponentially large.

Undergeneralisation

Easy

Positive counterexample from EQ

Suppose we receive a string w such that $w \in L(T) \setminus L(H)$

Add rows

$K \leftarrow K \cup Sub(w)$

Add every substring of w to K.

Observation

Informally

If we have enough contexts for K, then the hypothesis will not overgenerate.

Formally

If for all $u, v \in K$, $u \sim_J v$ implies $u \equiv_L v$, then $L(H) \subseteq L$.

Overgeneralisation

Problem

We generate a string $S \stackrel{*}{\Rightarrow} w$ but $w \notin L$
Note that $|w| \geq 2$

Cause

There must be two strings in K, u, v such that $u \sim_J v$ but not $u \equiv_L v$

Solution

Find these two strings, and return a context in the difference of $C_L(u)$ and $C_L(v)$

Starting point

Problem

$S \stackrel{*}{\Rightarrow} w$ and $S \in I$
But $w \notin L$ the target language

Triple

- A context $(l, r) = (\lambda, \lambda)$
- A non-terminal $X = S$
- A string w

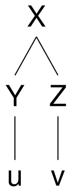
All strings generated by X should have the context (l, r)
$X \stackrel{*}{\Rightarrow} w$ but $(l, r) \notin C_L(w)$

Finding a context

Production

$X \rightarrow YZ$

pick $u'v' \rightarrow u', v'$ in K



Test

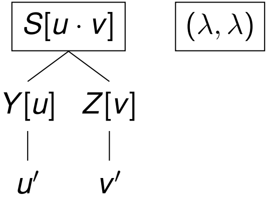
Test if all of the elements of X in K occur in the context (l, r)

If not, then return (l, r)

else recurse

Negative Counter example

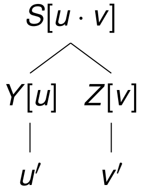
$w = u' \cdot v'$



- $u \cdot v$ should be congruent to $u' \cdot v'$
- But they aren't; as witnessed by context (λ, λ)
- So either $u \not\equiv u'$ or $v \not\equiv v'$
- MQs on $u'v$ and uv'.

Negative Counter example

$w = u' \cdot v'$



$\boxed{(u', \lambda)}$

- $u \cdot v$ should be congruent to $u' \cdot v'$
- But they aren't; as witnessed by context (λ, λ)
- So either $u \not\equiv u'$ or $v \not\equiv v'$
- MQs on $u'v$ and uv'.

Termination

Leaf

X
|
a

Must terminate since

- One element of X must have (l, r)
- But $a \in K$ and a does not have (l, r)
- So (l, r) splits X

Algorithm

Result: A CFG G

$K \leftarrow \{\lambda\}$;

$J \leftarrow \{(\lambda, \lambda)\}$;

$D = L \cap \{\lambda\}$;

$G = \langle K, D, J \rangle$;

while *true* **do**

    if `Equiv(`G`)` *returns correct* **then**

        return G ;

    $w \leftarrow$ `Equiv(`G`)` ;

    if *w is not in $L(G)$* **then**

        $K \leftarrow K \cup Sub(w)$;

    else

        $J \leftarrow J \cup$ `AddContexts(`G,w`)`;

    $G \leftarrow$ `MakeGrammar(`K, D, F`)` ;

Analysis

Assumptions

Target has n non-terminals and is a congruential CFG.
Counter-examples have maximum length l

Number of EQs is bounded.

Each positive EQ answer gives us at least 1 new production
$|K| \leq 1 + n^2 l(l+1)/2$
Each negative EQ gives us a context that increases the number of classes by at least 1.
Number of negative EQs at most $|K|$

Theorem

Algorithm terminates in time polynomial in n and l, and gives the right answer.

Example

$\{a^n b^n \mid n > 0\}$

$ab, aabb, aaabbb, \ldots$

Example

Step 0

	(λ, λ)
λ	0

Grammar

S and no productions

Counter example *ab*

	(λ, λ)
λ	0
a	0
b	0
ab	1

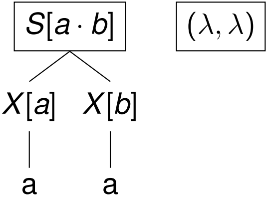
Grammar

S, X

$S \to XX$, $X \to a$, $X \to b$, $X \to \lambda$

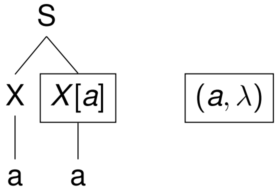
Negative Counter example *aa*

$S = \{ab\}$, $X = \{a, b, \lambda\}$



Negative Counter example *aa*

$S = \{ab\}$, $X = \{a, b, \lambda\}$



Counter example *aa*

	(λ, λ)	(a, λ)
λ	0	0
a	0	0
b	0	1
ab	1	0

Grammar

S, X, B

$S \to XB$, $X \to a$, $B \to b$, $X \to \lambda$

Positive counter example *aabb*

	(λ, λ)	(a, λ)
λ	0	0
a	0	0
b	0	1
ab	1	0
aa	0	0
bb	0	0
aab	0	0
abb	0	1
aabb	1	0

Grammar

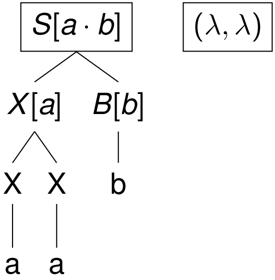
S, X, B

$S \rightarrow XB$, $X \rightarrow a$, $B \rightarrow b$, $X \rightarrow \lambda$,

$X \rightarrow XX$, $X \rightarrow XB$, $X \rightarrow BB \ldots$

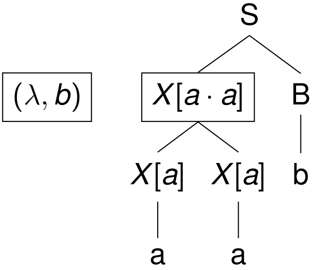
Negative Counter example *aab*

$S = \{ab, aabb\}$, $X = \{a, aa, bb, \lambda\}$, $B = \{b\}$



Negative Counter example *aab*

$S = \{ab, aabb\}$, $X = \{a, aa, bb, \lambda\}$, $B = \{b\}$



counter example *aab*

	(λ, λ)	(a, λ)	(λ, b)
λ	0	0	0
a	0	0	1
b	0	1	0
ab	1	0	0
aa	0	0	0
bb	0	0	0
aab	0	0	1
abb	0	1	0
aabb	1	0	0

Grammar

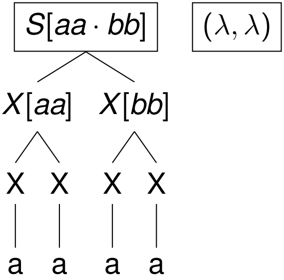
S, A, B, X

$S \to XX$, $X \to AA$, $X \to BB$,

$\dots$

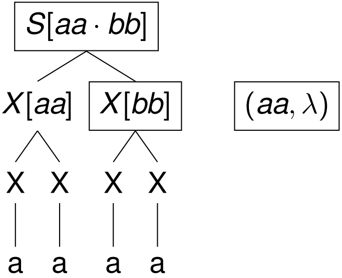
Negative Counter example *aaaa*

$S = \{ab, aabb\}$, $X = \{aa, bb, \lambda\}$, $A = \{a, aab\}$, $B = \{b, abb\}$



Negative Counter example *aaaa*

$S = \{ab, aabb\}$, $X = \{aa, bb, \lambda\}$, $A = \{a, aab\}$, $B = \{b, abb\}$



Some more negative counterexamples

	(λ, λ)	(a, λ)	(λ, b)	(aa, λ)	(λ, bb)
λ	0	0	0	0	0
a	0	0	1	0	0
b	0	1	0	0	0
ab	1	0	0	0	0
aa	0	0	0	0	1
bb	0	0	0	1	0
aab	0	0	1	0	0
abb	0	1	0	0	0
aabb	1	0	0	0	0

But $S \rightarrow AB \stackrel{*}{\Rightarrow} AABABB \rightarrow aababb$

Still more negative counterexamples

	(λ,λ)	(a,λ)	(λ,b)	(aa,λ)	(λ,bb)	(λ,abb)	(aab,λ)
λ	0	0	0	0	0	0	0
a	0	0	1	0	0	1	0
b	0	1	0	0	0	0	1
ab	1	0	0	0	0	0	0
aa	0	0	0	0	1	0	0
bb	0	0	0	1	0	0	0
aab	0	0	1	0	0	0	0
abb	0	1	0	0	0	0	0
aabb	1	0	0	0	0	0	0

Final grammar

Nonterminals S, A, B, A_2, B_2, X, Y

- $S \to AB$, $S \to XB$, $S \to AY$, $S \to A_2B_2$
- $A \to a$, $B \to b$, $A_2 \to AA$, $B_2 \to BB$
- $X \to AS$, $X \to A_2B$, $Y \to SB$, $Y \to AB_2$

Final grammar

Nonterminals S, A, B, A_2, B_2, X, Y

- $S \to AB$, $S \to XB$, $S \to AY$, $S \to A_2B_2$
- $A \to a$, $B \to b$, $A_2 \to AA$, $B_2 \to BB$
- $X \to AS$, $X \to A_2B$, $Y \to SB$, $Y \to AB_2$

We end up with a large and redundant grammar; this can be reduced later.

Further extensions

Survey of CFGs and MCFGs

[Clark and Yoshinaka, 2016].

Context-free tree grammars

[Kasprzik and Yoshinaka, 2011]

Recovering a canonical grammar

[Clark, 2013]

Bibliography I

Angluin, D. (1982).
Inference of reversible languages.
Journal of the ACM, 29(3):741–765.

Balcázar, J. L., Díaz, J., Gavaldà, R., and Watanabe, O. (1997).
Algorithms for Learning Finite Automata from Queries: A Unified View, pages 53–72.
Springer US, Boston, MA.

Boasson, L. and Sénizergues, S. (1985).
NTS languages are deterministic and congruential.
J. Comput. Syst. Sci., 31(3):332–342.

Bollig, B., Habermehl, P., Kern, C., and Leucker, M. (2009).
Angluin-style learning of NFA.
In *Proceedings of IJCAI 21*.

Bibliography II

Clark, A. (2006).
PAC-learning unambiguous NTS languages.
In *Proceedings of the 8th International Colloquium on Grammatical Inference (ICGI)*, pages 59–71.

Clark, A. (2010).
Distributional learning of some context-free languages with a minimally adequate teacher.
In Sempere, J. and Garcia, P., editors, *Grammatical Inference: Theoretical Results and Applications. Proceedings of the International Colloquium on Grammatical Inference*, pages 24–37. Springer-Verlag.

Clark, A. (2013).
Learning trees from strings: A strong learning algorithm for some context free grammars.
Journal of Machine Learning Research.

Bibliography III

Clark, A. and Eyraud, R. (2007).
Polynomial identification in the limit of substitutable context-free languages.
Journal of Machine Learning Research, 8:1725–1745.

Clark, A. and Thollard, F. (2004).
PAC-learnability of probabilistic deterministic finite state automata.
Journal of Machine Learning Research, 5:473–497.

Clark, A. and Yoshinaka, R. (2016).
Distributional learning of context-free and multiple context-free grammars.
In Heinz, J. and Sempere, M. J., editors, *Topics in Grammatical Inference*, pages 143–172. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bibliography IV

- Drewes, F. and Högberg, J. (2003).
  Learning a regular tree language from a teacher.
  In Ésik, Z. and Fülöp, Z., editors, *Developments in Language Theory*, pages 279–291. Springer Berlin Heidelberg.

- Kasprzik, A. and Yoshinaka, R. (2011).
  Distributional learning of simple context-free tree grammars.
  In Kivinen, J., SzepesvÃ¡ri, C., Ukkonen, E., and Zeugmann, T., editors, *Algorithmic Learning Theory*, volume 6925 of *Lecture Notes in Computer Science*, pages 398–412. Springer Berlin Heidelberg.

- Kearns, M. J. and Vazirani, U. V. (1994).
  An Introduction to Computational Learning Theory.
  The MIT Press.

Bibliography V

Ron, D., Singer, Y., and Tishby, N. (1998).
On the learnability and usage of acyclic probabilistic finite automata.
J. Comput. Syst. Sci., 56(2):133–152.

Shibata, C. and Yoshinaka, R. (2013).
PAC learning of some subclasses of context-free grammars with basic distributional properties.
In *Proceedings of Algorithmic Learning Theory Conference*, Berlin. Springer.
to appear.

Shirakawa, H. and Yokomori, T. (1993).
Polynomial-time MAT Learning of C-Deterministic Context-free Grammars.
Transactions of the information processing society of Japan, 34:380–390.