Learning Context-Free Grammars from Positive Data and Membership Queries

(based on joint work with Ryo Yoshinaka)

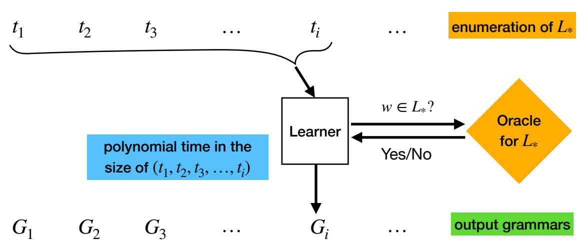
Makoto Kanazawa, Hosei University

Background

- Algorithmic Learning Theory
  - ✳ **Grammatical Inference**
    - Finite automata
      - → satisfactory learning algorithms
    - Context-free grammars / pushdown automata
      - → special subclasses
        - deterministic one-counter machines
        - context-deterministic grammars

There is a satisfactory learning algorithm for the regular languages. How much can a similar learning algorithm be made to work for context-free languages?
The goal is not (necessarily) to present an algorithm for the entire class of context-free languages.
(Both positive and negative results are valuable.)

Learning from Positive Data and Membership Queries



- There exists l such that $G_l = G_{l+1} = \cdots$ and $L(G_l) = L_*$.
- No “delaying trick”.

You have to specify the “learning paradigm”.
I write L_* for the target language.
You might wonder why you need positive data when you have access to the membership oracle.
Positive data is the input; queries are part of your work.
The “update time” is supposed to be polynomial in the size of the input positive data.
There is a cheap way of achieving polynomial update time by

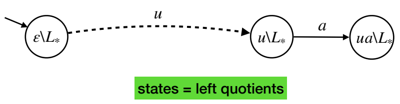
processing only an initial segment of the positive data. This is not allowed. (It's difficult to make this requirement precise, though.)

Regular Languages

- **Left quotient** of a language $L \subseteq \Sigma^*$ by a string $u \in \Sigma^*$:

$$u \backslash L = \{ x \in \Sigma^* \mid ux \in L \}$$

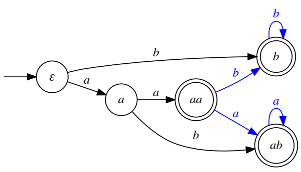
- A language L is regular if and only if $\{ u \backslash L \mid u \in \Sigma^* \}$ is finite.
- Every regular language has a canonical **minimal DFA**.



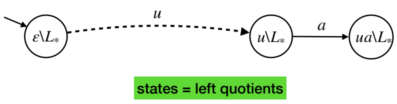
We first look at a learning algorithm for the regular languages under this learning paradigm.
We start with a review of some basic facts about regular languages.
There is a canonical minimal DFA for every regular language.
The states of this DFA correspond to the (nonempty) left quotients of the language.

Example

$$L_* = aba^* \cup bb^* \cup aa(a^* \cup b^*)$$



$$\begin{aligned}
\varepsilon \backslash L_* &= L_* \\
a \backslash L_* &= ba^* \cup a(a^* \cup b^*) \\
b \backslash L_* &= b^* \\
aa \backslash L_* &= a^* \cup b^* \\
ab \backslash L_* &= a^*
\end{aligned}$$



$$\begin{aligned}
bb \backslash L_* &= b^* = b \backslash L_* \\
aaa \backslash L_* &= a^* = ab \backslash L_* \\
aab \backslash L_* &= b^* = b \backslash L_* \\
aba \backslash L_* &= a^* = ab \backslash L_*
\end{aligned}$$

There is one more left quotient, namely the empty set, which would correspond to a dead state. We consider minimal DFAs without dead states.

Right-Linear Grammars

- Right-linear CFG G_* corresponding to a minimal DFA for L_*:

$G_* = (N_*, \Sigma, P_*, S)$

$N_* = \{\, u\backslash L_* \mid u \in \Sigma^* \,\}$

$P_* = \{\, u\backslash L_* \to a\,(ua\backslash L_*) \mid u \in \Sigma^*, a \in \Sigma \,\} \cup \{\, u\backslash L_* \to \varepsilon \mid u \in L_* \,\}$

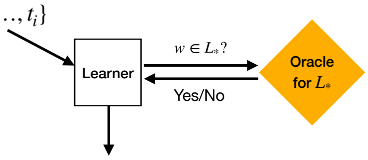
$S = L_*\ (= \varepsilon\backslash L_*)$

states = nonterminals = left quotients

Just an alternative notation.

Inference of Regular Languages

$T = \{t_1, \dots, t_i\}$



$\mathrm{Pref}(T) = \{\, u \mid uv \in T \,\}$
$\mathrm{Suff}(T) = \{\, v \mid uv \in T \,\}$

$G = (N, \Sigma, P, \langle\!\langle \varepsilon \rangle\!\rangle)$

represents $u\backslash L_*$ — length-lexicographic order on Σ^*

$N = \{\, \langle\!\langle u \rangle\!\rangle \mid u \in \mathrm{Pref}(T) \wedge \forall v \in \mathrm{Pref}(T)\,(v \prec u \to (\{\varepsilon\} \cup \Sigma)\,\mathrm{Suff}(T) \cap (v\backslash L_*) \neq (\{\varepsilon\} \cup \Sigma)\,\mathrm{Suff}(T) \cap (u\backslash L_*) \,\}$

approximates $v\backslash L_* \neq u\backslash L_*$

$P = \{\, \langle\!\langle u \rangle\!\rangle \to a\,\langle\!\langle v \rangle\!\rangle \mid \mathrm{Suff}(T) \cap (ua\backslash L_*) = \mathrm{Suff}(T) \cap (v\backslash L_*) \,\} \cup \{\, \langle\!\langle u \rangle\!\rangle \to \varepsilon \mid u \in L_* \,\}$

approximates $ua\backslash L_* = v\backslash L_*$

This is basically Angluin's algorithm. The order of the presentation of the positive data doesn't matter, so I'm treating it as a set T.
The algorithm outputs right-linear grammars.
Use <<u>> as the representation of u \ L_*.
Different strings may correspond to the same left quotient. Try to use the lexicographically least one.

$N_* = \{\, u\backslash L_* \mid u \in \Sigma^* \,\}$

represents $u\backslash L_*$ — lexicographic order on Σ^*

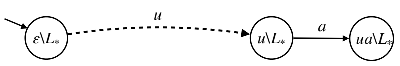
$N = \{\, \langle\!\langle u \rangle\!\rangle \mid u \in \mathrm{Pref}(T) \wedge \forall v \in \mathrm{Pref}(T)\,(v \prec u \to (\{\varepsilon\} \cup \Sigma)\,\mathrm{Suff}(T) \cap (v\backslash L_*) \neq (\{\varepsilon\} \cup \Sigma)\,\mathrm{Suff}(T) \cap (u\backslash L_*) \,\}$

approximates $v\backslash L_* \neq u\backslash L_*$

$P_* = \{\, u\backslash L_* \to a\,(ua\backslash L_*) \mid u \in \Sigma^*, a \in \Sigma \,\} \cup \{\, u\backslash L_* \to \varepsilon \mid u \in L_* \,\}$
$\quad = \{\, u\backslash L_* \to a\,(v\backslash L_*) \mid ua\backslash L_* = v\backslash L_* \,\} \cup \{\, u\backslash L_* \to \varepsilon \mid u \in L_* \,\}$

$P = \{\, \langle\!\langle u \rangle\!\rangle \to a\,\langle\!\langle v \rangle\!\rangle \mid \mathrm{Suff}(T) \cap (ua\backslash L_*) = \mathrm{Suff}(T) \cap (v\backslash L_*) \,\} \cup \{\, \langle\!\langle u \rangle\!\rangle \to \varepsilon \mid u \in L_* \,\}$

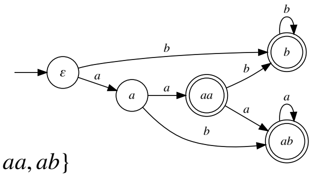
approximates $ua\backslash L_* = v\backslash L_*$



Compare the learner's output with the canonical right-linear CFG (minimal DFA) of the target regular language.

Example

$L_* = aba^* \cup bb^* \cup aa(a^* \cup b^*)$



$$T = \{b, aa, ab\}$$
$$\mathrm{Pref}(T) = \{\varepsilon, a, b, aa, ab\}$$
$$\mathrm{Suff}(T) = \{\varepsilon, a, b, aa, ab\}$$
$$\{\varepsilon, a, b\}\,\mathrm{Suff}(T) \cap (\varepsilon\backslash L_*) = \{b, aa, ab, bb, aaa, aab\}$$
$$\{\varepsilon, a, b\}\,\mathrm{Suff}(T) \cap (a\backslash L_*) = \{a, b, aa, ab, ba, aaa, baa\}$$
$$\{\varepsilon, a, b\}\,\mathrm{Suff}(T) \cap (b\backslash L_*) = \{\varepsilon, b, bb\}$$
$$\{\varepsilon, a, b\}\,\mathrm{Suff}(T) \cap (aa\backslash L_*) = \{\varepsilon, a, b, aa, aaa, bb\}$$
$$\{\varepsilon, a, b\}\,\mathrm{Suff}(T) \cap (ab\backslash L_*) = \{\varepsilon, a, aa, aaa\}$$

The prefixes of the strings in T are distinguishable from each other. This positive data is enough to make the learner arrive at the correct hypothesis.

---

represents $u\backslash L_*$ lexicographic order on Σ^*

$$N = \{ \langle\!\langle u \rangle\!\rangle \mid u \in \mathrm{Pref}(T) \wedge \forall v \in \mathrm{Pref}(T)\,(v \prec u \rightarrow (\{\varepsilon\} \cup \Sigma)\,\mathrm{Suff}(T) \cap (v\backslash L_*) \neq (\{\varepsilon\} \cup \Sigma)\,\mathrm{Suff}(T) \cap (u\backslash L_*) \}$$

approximates $v\backslash L_* \neq u\backslash L_*$

$$P = \{ \langle\!\langle u \rangle\!\rangle \rightarrow a\,\langle\!\langle v \rangle\!\rangle \mid \mathrm{Suff}(T) \cap (ua\backslash L_*) = \mathrm{Suff}(T) \cap (v\backslash L_*) \} \cup \{ \langle\!\langle u \rangle\!\rangle \rightarrow \varepsilon \mid u \in L_* \}$$

approximates $ua\backslash L_* = v\backslash L_*$

$$x \in (\{\varepsilon\} \cup \Sigma)\,\mathrm{Suff}(T) \cap (u\backslash L_*) \iff x \in (\{\varepsilon\} \cup \Sigma)\,\mathrm{Suff}(T) \wedge ux \in L_*$$

- N and P are determined by $\mathrm{Pref}(T)\,(\{\varepsilon\} \cup \Sigma)\,\mathrm{Suff}(T) \cap L_*$.
- $O(n^2)$ queries to the oracle for L_* suffice.

To compute N and P, you have to decide equality between various finite sets.
This is done by queries to the membership oracle.

Context-Free Grammars

$$G = (N, \Sigma, P, S)$$

- What do nonterminals correspond to?

$$\frac{\text{left quotients}}{\text{regular languages}} = \frac{??}{\text{context-free languages}}$$

- The set of terminal strings derived from a nonterminal is included in some **quotient** of the language of the grammar:

$$S \Rightarrow_G^* uAv \quad \text{implies} \quad L_G(A) \subseteq u\backslash L(G)/v = \{ x \in \Sigma^* \mid uxv \in L(G) \}$$

$L_G(A) = \{ x \in \Sigma^* \mid A \Rightarrow_G^* x \}$
$L(G) = L_G(S)$

- It seems there's nothing further that can be said in general.

How can a similar approach work for context-free languages?
Left quotients played an important role in the case of regular languages.
A left quotient corresponds to a state of the minimal DFA, and membership in it can be determined by a membership query.

Simple Case: Grammars with Just One Nonterminal

Dyck language

$$\boxed{\begin{array}{l} S \to \varepsilon \\ S \to aSbS \end{array}} \qquad D_1 = \{\, x \in \{a,b\}^* \mid |x|_a = |x|_b \wedge \forall uv(uv = x \to |u|_a \geq |u|_b)\,\}$$

$$\pi\colon \quad S \to w_0\, S\, w_1\, \dots\, S\, w_k \qquad (w_i \in \Sigma^*)$$

When should π be in the hypothesized grammar?

$$\pi \text{ is } \textbf{valid} \overset{\text{def}}{\iff} L_* \supseteq w_0\, L_*\, w_1\, \dots\, L_*\, w_k$$

Why is this reasonable?

Let's bypass the problem of how to deal with nonterminals and consider the special class of CFGs whose start symbol is the only nonterminal. An example of such a CFG is a grammar for the Dyck language. ε denotes the empty string.

$$\pi\colon \quad S \to w_0\, S\, w_1\, \dots\, S\, w_k \qquad (w_i \in \Sigma^*)$$

$$\pi \text{ is } \textbf{valid} \overset{\text{def}}{\iff} L_* \supseteq w_0\, L_*\, w_1\, \dots\, L_*\, w_k$$

- If π is not valid, π can't be in a correct grammar for L_*.

  If $x_1, \dots, x_k \in L_*$, $w_0 x_1 w_1 \dots x_k w_k \notin L_*$, and $L_* \subseteq L(G)$, then

  $$\begin{aligned} S &\Rightarrow w_0\, S\, w_1\, \dots\, S\, w_k \\ &\Rightarrow^* w_0\, x_1\, w_1\, \dots\, x_k\, w_k \end{aligned}$$
- If all productions in G are valid, then $L(G) \subseteq L_*$.
- All productions in G_* are valid.

The first bullet point is easy to see. If all strings in L_* are in L_G(S), the presence of an invalid production implies that L_G(S) - L_* ≠ ∅. (This need not be so if G has more than one nonterminal, though.)
In order to understand the second and third bullet points, it is useful to understand some basic facts about context-free grammars in general.

Context-Free Grammars

abbreviates three productions: $S \to aD_1bS,\ S \to aA,\ S \to bU$

$$\boxed{\begin{array}{l} S \to aD_1bS \mid aA \mid bU \\ D_1 \to \varepsilon \mid aD_1bD_1 \\ A \to \varepsilon \mid aD_1bA \mid aA \\ U \to \varepsilon \mid Ua \mid Ub \end{array}}$$

$$\overline{D_1} = \{\, x \in \{a,b\}^* \mid |x|_a \neq |x|_b \vee \exists uv(uv = x \wedge |u|_a < |u|_b)\,\}$$

$$L_G(A) = \{\, x \in \Sigma^* \mid A \Rightarrow_G^* x \,\}$$

$(L_G(S), L_G(D_1), L_G(A), L_G(U))$ is the **least fixed point** of the operator $\Phi_G\colon (\mathscr{P}(\{a,b\}^*))^4 \to (\mathscr{P}(\{a,b\}^*))^4$:

$$\Phi_G \begin{bmatrix} X_S \\ X_{D_1} \\ X_A \\ X_U \end{bmatrix} = \begin{bmatrix} aX_{D_1}bX_S \cup aX_A \cup bX_U \\ \varepsilon \cup aX_{D_1}bX_{D_1} \\ \varepsilon \cup aX_{D_1}bX_A \cup aX_A \\ \varepsilon \cup X_Ua \cup X_Ub \end{bmatrix}$$

Let's look at context-free grammars in general.
Productions with the same left-hand side nonterminal are often collected together. Nonterminals are interpreted as sets, and the vertical bar is interpreted as union.
We'll look at this grammar in more detail later.

Pre-fixed Points of Context-Free Grammars

$$G = (N, \Sigma, P, S)$$
Φ_G: associated operator

- $(X_B)_{B \in N}$ is a **pre-fixed point** of $\Phi_G \overset{\text{def}}{\iff}$ $\Phi_G((X_B)_{B \in N}) \subseteq (X_B)_{B \in N}$

componentwise inclusion

- $(L_G(B))_{B \in N}$ is the least pre-fixed point of Φ_G.
- $(X_B)_{B \in N}$ is a pre-fixed point of Φ_G if and only if for every production $A \to w_0 B_1 w_1 \dots B_k w_k$ in P,

$$X_A \supseteq w_0 X_{B_1} w_1 \dots X_{B_k} w_k.$$

Least fixed points coincide with least pre-fixed points.
The advantage of pre-fixed points is that you can look at individual productions in isolation.

Fixed Points

$$\boxed{\begin{aligned} S &\to aD_1bS \mid aA \mid bU \\ D_1 &\to \varepsilon \mid aD_1bD_1 \\ A &\to \varepsilon \mid aD_1bA \mid aA \\ U &\to \varepsilon \mid Ua \mid Ub \end{aligned}}$$

$$\begin{aligned} X_S &= aX_{D_1}bX_S \cup aX_A \cup bX_U \\ X_{D_1} &= \varepsilon \cup aX_{D_1}bX_{D_1} \\ X_A &= \varepsilon \cup aX_{D_1}bX_A \cup aX_A \\ X_U &= \varepsilon \cup X_Ua \cup X_Ub \end{aligned}$$

Pre-fixed Points

$$\boxed{\begin{aligned} S &\to aD_1bS \mid aA \mid bU \\ D_1 &\to \varepsilon \mid aD_1bD_1 \\ A &\to \varepsilon \mid aD_1bA \mid aA \\ U &\to \varepsilon \mid Ua \mid Ub \end{aligned}}$$

$$\begin{aligned} X_S &\supseteq aX_{D_1}bX_S \cup aX_A \cup bX_U \\ X_{D_1} &\supseteq \varepsilon \cup aX_{D_1}bX_{D_1} \\ X_A &\supseteq \varepsilon \cup aX_{D_1}bX_A \cup aX_A \\ X_U &\supseteq \varepsilon \cup X_Ua \cup X_Ub \end{aligned}$$

Pre-fixed Points

$$\begin{array}{|l|}\hline S \to aD_1bS \mid aA \mid bU \\ D_1 \to \varepsilon \mid aD_1bD_1 \\ A \to \varepsilon \mid aD_1bA \mid aA \\ U \to \varepsilon \mid Ua \mid Ub \\ \hline \end{array}$$

$S \to aD_1bS$	$X_S \supseteq aX_{D_1}bX_S$
$S \to aA$	$X_S \supseteq aX_A$
$S \to bU$	$X_S \supseteq bX_U$
$D_1 \to \varepsilon$	$X_{D_1} \supseteq \varepsilon$
$D_1 \to aD_1bD_1$	$X_{D_1} \supseteq aX_{D_1}bX_{D_1}$
$A \to \varepsilon$	$X_A \supseteq \varepsilon$
$A \to aD_1bA$	$X_A \supseteq aX_{D_1}bX_A$
$A \to aA$	$X_A \supseteq aX_A$
$U \to \varepsilon$	$X_U \supseteq \varepsilon$
$U \to Ua$	$X_U \supseteq X_Ua$
$U \to Ub$	$X_U \supseteq X_Ub$

Simple Case: Grammars with Just One Nonterminal

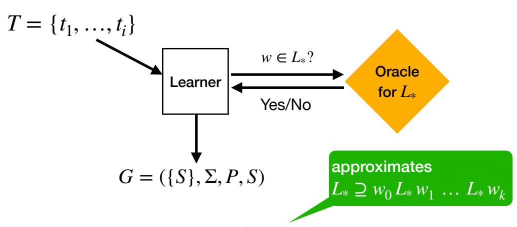
$$\pi: \quad S \to w_0 S w_1 \dots S w_k \qquad (w_i \in \Sigma^*)$$

$$\pi \text{ is } \textbf{valid} \overset{\text{def}}{\iff} L_* \supseteq w_0 L_* w_1 \dots L_* w_k$$

- If π is not valid, π can't be in a correct grammar for L_*.
- If all productions in G are valid, then $L(G) \subseteq L_*$.
  $\because L_*$ is a pre-fixed point of G.
- All productions in G_* are valid.
  $\because L_* = L(G_*)$ is the least pre-fixed point of G_*.

L(G) is the S-component of the least pre-fixed point of G.
L_* is the S-component of the least pre-fixed point of G_*.
If G has enough many productions that all productions in G_* are in G, then L_* ⊆ L(G).

Inference of Context-Free Grammars with Just One Nonterminal



$$P = \{\, S \to w_0 S w_1 \dots S w_k \mid L_* \supseteq w_0 T w_1 \dots T w_k,\ k \le r, |w_i| \le s\ (0 \le i \le k) \,\}$$

- Need a constant bound r on k, since deciding $L_* \supseteq w_0 T w_1 \dots T w_k$ requires $|T|^k$ queries to the oracle for L_*.
- Placing a constant bound s on $|w_i|$ makes the set of possible productions finite.

Need to place some bound on |w_i| to make P finite. Using a constant bound is the easiest, but then the class of grammars becomes finite. It is possible to use a non-constant bound, but you can't generate all regular languages with just one nonterminal, so this class is too restrictive to be interesting anyway.

Learning Context-Free Grammars (with More Than One Nonterminal)

$$\pi: \quad A \to w_0\, B_1\, w_1\, \dots\, B_k\, w_k$$

$$\pi \text{ is } \textbf{valid} \overset{\text{def}}{\iff} [\![A]\!]^{L_*} \supseteq w_0\, [\![B_1]\!]^{L_*}\, w_1\, \dots\, [\![B_k]\!]^{L_*}\, w_k$$

- A hypothesized nonterminal B "denotes" a set $[\![B]\!]^{L_*}$ relative to the target language L_*, independently of the rest of the hypothesized grammar.
- In particular, it is not necessarily the case that $L_G(B) = [\![B]\!]^{L_*}$ (even in the limit).
- Membership in $[\![B]\!]^{L_*}$ **reduces in polynomial time** to membership in L_*.
- This reduction is uniform across different target languages.

The general case of context-free grammars with more than one nonterminal.
In the case of the regular languages, we used nonterminals that denote left quotients of the target language.
In the case of CFGs with just one nonterminal, we used a nonterminal that denotes the target language.
We use certain representations as nonterminals that denote sets relative to L_*.
It is often the case that L_G(B) = [[B]]^{L_*} when the output grammar has converged to a correct grammar for L_*, but even then it does not always hold.
You're supposed to be able to tell whether a particular string belongs to the denotation of a nonterminal by making queries to the oracle for L_*.
Since you don't know the identity of L_*, this reduction must be independent of L_*.

Simplest Class: Nonterminals Denote Quotients

- The learner uses pairs of strings as nonterminals.

$$\begin{aligned} [\![\langle\!\langle u, v\rangle\!\rangle]\!]^{L_*} &= u \backslash L_* / v \\ &= \{\, x \in \Sigma^* \mid uxv \in L_* \,\} \end{aligned}$$

$$P = \{\, A \to w_0\, B_1\, w_1\, \dots\, B_k\, w_k \mid [\![A]\!]^{L_*} \supseteq w_0\, (\mathrm{Sub}(T) \cap [\![B_1]\!]^{L_*})\, w_1\, \dots\, (\mathrm{Sub}(T) \cap [\![B_k]\!]^{L_*})\, w_k,\ k \le r, |w_i| \le s\ (0 \le i \le k) \,\}$$

approximates $[\![A]\!]^{L_*} \supseteq w_0\, [\![B_1]\!]^{L_*}\, w_1\, \dots\, [\![B_k]\!]^{L_*}\, w_k$

$\mathrm{Sub}(T) = \{\, x \mid uxv \in T \,\}$

$$S = \langle\!\langle \varepsilon, \varepsilon \rangle\!\rangle$$

- L_* has infinitely many quotients unless it is regular, so the learner must stop creating new nonterminals.

We start with a very simple instantiation of this idea, where nonterminals denote quotients of L_*.
The strings u, v used to represent nonterminals are drawn from positive data, similarly to the case of the regular languages.

Algorithm 1

$T_0 := \varnothing; E_0 := \varnothing; J_0 := \varnothing; G_0 := (\{\langle\langle \varepsilon, \varepsilon\rangle\rangle\}, \Sigma, \varnothing, \langle\langle \varepsilon, \varepsilon\rangle\rangle);$
for $i = 1, 2, 3, \ldots$ **do**
  $T_i := T_{i-1} \cup \{t_i\};$
  if $T_i \subseteq L(G_{i-1})$ **then**
    $J_i := J_{i-1};$
  else
    $J_i := \mathrm{Con}(T_i);$
  end
  $N_i := \{ \langle\langle u, v\rangle\rangle \mid (u, v) \in J_i \wedge$
    $\forall (u', v') \in J_i ((u', v') \prec_2 (u, v) \to E \cap (u' \backslash L_* / v') \neq E \cap (u \backslash L_* / v) \};$
  $P_i := \{ A \to w_0 B_1 w_1 \ldots B_k w_k \mid A, B_1, \ldots, B_k \in N_i,$
    $[\![A]\!]^{L_*} \supseteq w_0 (\mathrm{Sub}(T_i) \cap [\![B_1]\!]^{L_*}) w_1 \ldots (\mathrm{Sub}(T_i) \cap [\![B_k]\!]^{L_*}) w_k,$
    $k \leq r, |w_j| \leq s \ (1 \leq j \leq k) \}$
  output $G_i := (N_i, \Sigma, P_i, \langle\langle \varepsilon, \varepsilon\rangle\rangle);$
end

expand J_i only when $T_i \nsubseteq L(G_{i-1})$

$\mathrm{Con}(T) = \{ (u, v) \mid uwv \in T \}$

lexicographic product of $\prec$ with itself

$E = \Sigma^{\leq s} (\mathrm{Sub}(T_i) \Sigma^{\leq s})^{\leq r}$

Do not pay too much attention to the details of the algorithm.
The important points are:
- nonterminals are pairs of strings and denote quotients of the target language
- these pairs of strings are drawn from positive data
- the learner creates new nonterminals only when the positive data is inconsistent with the previous hypothesis
- the learner tries to include only

CFGs with the Quotient Property

- $G = (N, \Sigma, P, S)$ has the **quotient property** $\overset{\text{def}}{\iff}$ G has a pre-fixed point $(X_B)_{B \in N}$ with $X_S = L(G)$ such that $X_B \in \mathcal{Q}(L(G))$ for all $B \in N$.

  $\mathcal{Q}(L) = \{ u \backslash L / v \mid u, v \in \Sigma^* \}$

Theorem.
- Algorithm 1 successfully learns L_* if and only if L_* has a grammar with the quotient property.
- If Algorithm 1 converges to G, then G has the quotient property.

Examples of CFGs with the Quotient Property

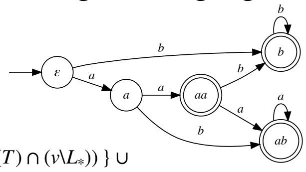
- CFGs with just one nonterminal.
- Right-linear grammars corresponding to minimal DFAs of regular languages.
- $L = \{ a^n b^n \mid n \geq 0 \} \{ a^n b^n \mid n \geq 0 \}$

$$S \to AA$$
$$A \to \varepsilon \mid aAb$$
$$S = \varepsilon \backslash L / \varepsilon \ (= L),$$
$$A = a \backslash L / bab$$

L_G(S) = ε \ L(G) / ε.
Left quotients are quotients.
You need at least two nonterminals for the third example.

An Analogous Algorithm for Regular Languages

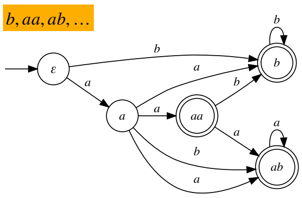
$$L_* = aba^* \cup bb^* \cup aa(a^* \cup b^*)$$



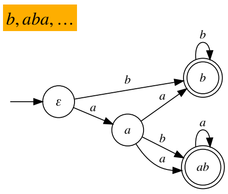
$$P = \{ \langle\!\langle u \rangle\!\rangle \to a \langle\!\langle v \rangle\!\rangle \mid u\backslash L_* \supseteq a\,(\mathrm{Suff}(T) \cap (v\backslash L_*)) \} \cup$$
$$\{ \langle\!\langle u \rangle\!\rangle \to \varepsilon \mid u \in L_* \}$$

approximates $u\backslash L_* \supseteq a\,(v\backslash L_*)$ ($\iff ua\backslash L_* \supseteq v\backslash L_*$)

$b, aa, ab, \ldots$



$b, aba, \ldots$



To see how reasonable Algorithm 1 is, let's look at how its analogue for the regular languages behaves. Before, you had a condition that approximates a certain identity. Here, you have a condition that approximates an inclusion.

Γ-closure

- Γ: finite set of operations on $\mathscr{P}(\Sigma^*)$ (of variable arity)
- For $\mathscr{L} \subseteq \mathscr{P}(\Sigma^*)$,

$$\Gamma(\mathscr{L}) = \{ f(L_1, \ldots, L_m) \mid f \colon (\mathscr{P}(\Sigma^*))^m \to \mathscr{P}(\Sigma^*), f \in \Gamma, L_1, \ldots, L_m \in \mathscr{L} \}$$
$$\Gamma^0(\mathscr{L}) = \mathscr{L}$$
$$\Gamma^{n+1}(\mathscr{L}) = \mathscr{L} \cup \Gamma(\Gamma^n(\mathscr{L}))$$

Γ-closure of $\mathscr{Q}(L)$

- Sets in $\bigcup_{t \geq 0} \Gamma^t(\mathscr{Q}(L))$ can be represented by expressions built from $\langle\!\langle u, v \rangle\!\rangle$ and symbols for operations in Γ.

Now let's look at more general classes of representations (used as nonterminals).

Extended Regular Closure

$$\Gamma = \{\cap\,, \overline{\,\cdot\,}, \cup\,\} \cup \{\varnothing, \varepsilon\} \cup \Sigma \cup \{\text{concatenation}, {}^*\,\}$$

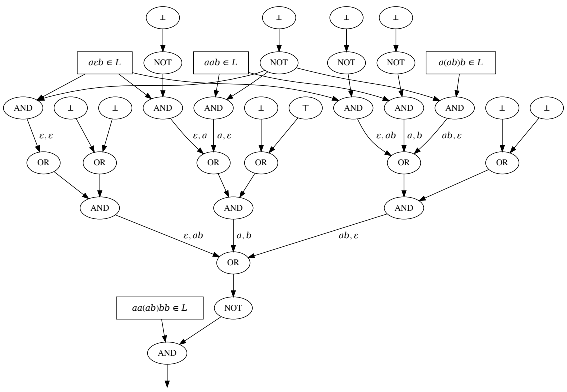
extended regular expression over query atoms

$$\begin{aligned}
&[\![\langle\!\langle aa, bb \rangle\!\rangle \cap \overline{(\langle\!\langle a, b \rangle\!\rangle\, \overline{\varnothing})\,(a \cup b)}]\!]^L \\
&\quad = (aa\backslash L/bb) \cap (\{a,b\}^* - ((a\backslash L/b)(\{a,b\}^* - \varnothing))(\{a\} \cup \{b\})) \\
&\quad = (aa\backslash L/bb) \cap (\{a,b\}^* - (a\backslash L/b)\{a,b\}^*\{a,b\}) \\
&\quad = \{ x \mid x \in aa\backslash L/bb \wedge \text{no proper prefix of } x \text{ is in } a\backslash L/b \}
\end{aligned}$$

- If e is an extended regular expression over query atoms, then $[\![e]\!]^L$ **reduces in polynomial time** to L.

$$[\![e]\!]^L \leq^P_{tt} L$$

The algorithm works when Γ-expressions (expressions that stand for sets belonging to the Γ-closure) translate into polynomial-time reductions.
When Γ consists of the Boolean and regular operations, we get polynomial-time truth-table reduction.



$ab \in [\![\langle\!\langle aa, bb\rangle\!\rangle \cap \overline{(\langle\!\langle a, b\rangle\!\rangle\, \overline{\varnothing})\, (a \cup b)}]\!]^L$

The Boolean circuit for the truth function the reduction uses for this particular input. The circuit for x ∈ [[e]]^L depends on e and x, but not on L.

Γ-closure Property

- A CFG $G = (N, \Sigma, P, S)$ has the **Γ^t-property** $\overset{\text{def}}{\iff}$ G has a pre-fixed point $(X_B)_{B \in N}$ with $X_S = L(G)$ such that $X_B \in \Gamma^t(\mathcal{Q}(L(G)))$ for all $B \in N$.
- G has the **Γ-closure property** $\overset{\text{def}}{\iff}$ G has the Γ^t-property for some t.

Learning CFGs with the Extended Regular Closure Property

- The learner uses extended regular expressions over query atoms as nonterminals.

approximates $[\![A]\!]^{L_*} \supseteq w_0\, [\![B_1]\!]^{L_*}\, w_1\, \dots\, [\![B_k]\!]^{L_*}\, w_k$

$$P = \{\, A \to w_0\, B_1\, w_1\, \dots\, B_k\, w_k \mid [\![A]\!]^{L_*} \supseteq w_0\, (\mathrm{Sub}(T) \cap [\![B_1]\!]^{L_*})\, w_1\, \dots\, (\mathrm{Sub}(T) \cap [\![B_k]\!]^{L_*})\, w_k,\ k \le r, |w_i| \le s\ (0 \le i \le k) \,\}$$

$$S = \langle\!\langle \varepsilon, \varepsilon \rangle\!\rangle$$

Algorithm 2

$T_0 := \varnothing; E_0 := \varnothing; J_0 := \varnothing; G_0 := (\{\langle\!\langle \varepsilon, \varepsilon \rangle\!\rangle\}, \Sigma, \varnothing, \langle\!\langle \varepsilon, \varepsilon \rangle\!\rangle);$

for $i = 1, 2, 3, \dots$ **do**

  $T_i := T_{i-1} \cup \{t_i\};$

  if $T_i \subseteq L(G_{i-1})$ **then**

    $J_i := J_{i-1};$

  else

    $J_i := \mathrm{Con}(T_i);$

  end

  $Q_i := \{ \langle\!\langle u, v \rangle\!\rangle \mid (u, v) \in J_i \wedge \forall (u', v') \in J_i ((u', v') \prec_2 (u, v) \to E \cap (u' \backslash L_* / v') \neq E \cap (u \backslash L_* / v) \};$

  $N_i :=$ the set of Γ^t-expressions over Q_i;

  $P_i := \{ A \to w_0 B_1 w_1 \dots B_k w_k \mid A, B_1, \dots, B_k \in N_i, [\![A]\!]^{L_*} \supseteq w_0 (\mathrm{Sub}(T_i) \cap [\![B_1]\!]^{L_*}) w_1 \dots (\mathrm{Sub}(T_i) \cap [\![B_k]\!]^{L_*}) w_k, k \le r, |w_j| \le s \ (1 \le j \le k) \}$

  output $G_i := (N_i, \Sigma, P_i, \langle\!\langle \varepsilon, \varepsilon \rangle\!\rangle);$

end

expand J_i only when $T_i \not\subseteq L(G_{i-1})$

$\mathrm{Con}(T) = \{ (u, v) \mid uwv \in T \}$

lexicographic product of $\prec$ with itself

$E = \Sigma^{\le s} (\mathrm{Sub}(T_i)\, \Sigma^{\le s})^{\le r}$

Algorithm 1 used Q_i as N_i. That's the only difference from Algorithm 1.

$$\{ \cap \} \subseteq \Gamma \subseteq \{ \cap, \overline{\ \cdot\ }, \cup \} \cup \{\varnothing, \varepsilon\} \cup \Sigma \cup \{\text{concatenation}, * \}$$

Theorem.

- Algorithm 2 successfully learns L_* if and only if L_* has a grammar with the Γ^t-property.
- If Algorithm 2 converges to G, then G has the Γ^t-property.

Extended Regular Closure Property

$$\boxed{\begin{array}{l} S \to aD_1bS \mid aA \mid bU \\ D_1 \to \varepsilon \mid aD_1bD_1 \\ A \to \varepsilon \mid aD_1bA \mid aA \\ U \to \varepsilon \mid Ua \mid Ub \end{array}}$$

$$\begin{aligned} \overline{D_1} &= \{ x \in \{a, b\}^* \mid |x|_a \neq |x|_b \vee \exists uv(uv = x \wedge |u|_a < |u|_b) \} \\ &= \{ x \in \{a, b\}^* \mid |x|_a > |x|_b \vee \exists uv(uv = x \wedge |u|_a < |u|_b) \} \end{aligned}$$

$$\begin{aligned} A &= \{ x \in \{a, b\}^* \mid \forall uv(x = uv \to |u|_a \geq |u|_b) \} \\ &= \{ x \in \{a, b\}^* \mid \text{no prefix of } x \text{ is in } D_1 b \} \\ &= \overline{D_1 b \{a, b\}^*} \\ &= [\![\overline{\langle\!\langle \varepsilon, \varepsilon \rangle\!\rangle\, b\, (a \cup b)^*}]\!]^{\overline{D_1}} \end{aligned}$$

Our example grammar has the extended regular closure property.

Boolean Closure Property

$$\boxed{\begin{array}{l} S \to D_1 b D_1 \mid D_1 a D_1 \mid D_1 b S \mid S a D_1 \\ D_1 \to \varepsilon \mid a D_1 b D_1 \end{array}}$$

$$\overline{D_1} = \{\, x \in \{a,b\}^* \mid \exists mn(m+n>0 \wedge \mathrm{nf}(x) = b^m a^n) \,\}$$

normal form of x under the rewriting $ab \to \varepsilon$

$$\begin{aligned} S &\Rightarrow^* D_1\, b\, \dots\, D_1\, b\, D_1\, a\, D_1\, \dots\, a\, D_1 \\ &\Rightarrow^* x_1\, b\, \dots\, x_m\, b\, y\, a\, z_1\, \dots\, a\, z_n \end{aligned}$$

$$D_1 = [\![\overline{\langle\!\langle \varepsilon, \varepsilon \rangle\!\rangle}]\!]^{\overline{D_1}}$$

In fact, the same language has a grammar with the “Boolean closure property”.

Extended Regular Closure Property

$$\boxed{\begin{array}{l} S \to aA \mid aD_1bS \mid bB \mid bD_1^RaS \\ A \to \varepsilon \mid aA \mid aD_1bA \\ D_1 \to \varepsilon \mid aD_1bD_1 \\ B \to \varepsilon \mid bB \mid bD_1^RaB \\ D_1^R \to \varepsilon \mid bD_1^RaD_1^R \end{array}}$$

$$\overline{O_1} = \{\, x \in \{a,b\}^* \mid |x|_a \neq |x|_b \,\}$$

$$\begin{aligned} A &= \{\, x \in \{a,b\}^* \mid \forall uv(x = uv \to |u|_a \geq |u|_b) \,\} \\ &= \{\, x \in \{a,b\}^* \mid \neg \exists uv(x = uv \wedge |u|_a + 1 = |u|_b) \,\} \\ &= \overline{O_1 b \{a,b\}^*} \\ &= [\![\overline{\overline{\langle\!\langle \varepsilon, \varepsilon \rangle\!\rangle}\, b\, (a \cup b)^*}]\!]^{\overline{O_1}} \\ D_1 &= A \cap O_1 \\ &= [\![\overline{\overline{\langle\!\langle \varepsilon, \varepsilon \rangle\!\rangle}\, b\, (a \cup b)^*} \cap \overline{\langle\!\langle \varepsilon, \varepsilon \rangle\!\rangle}]\!]^{\overline{O_1}} \end{aligned}$$

Theorem. $\overline{O_1}$ does not have a grammar with the Boolean closure property.

The pumping lemma applied to a long string a^p gives

$$\begin{aligned} S &\Rightarrow^* a^{l_1} A a^{r_1} \\ A &\Rightarrow^* a^{l_2} A a^{r_2} \qquad (l_2 + r_2 > 0) \\ A &\Rightarrow^* a^m \end{aligned}$$

If $(X_B)_{B \in N}$ is a pre-fixed point with $X_S = \overline{O}_1$, then

$$\{\, a^{nl_2 + m + nr_2} \mid n \geq 0 \,\} \subseteq X_A \subseteq \bigcap_{n \geq 0} a^{l_1 + nl_2} \backslash \overline{O_1} / a^{nr_2 + r_1}$$

It follows that $\{\, |x|_a - |x|_b \mid x \in X_A \,\}$ is both infinite and co-infinite.

But $\{\, |x|_a - |x|_b \mid x \in u \backslash \overline{O_1} / v \,\} = \mathbb{Z} - \{\, -(|uv|_a - |uv|_b) \}$ is a co-finite set.

CFLs That Have No Grammar with the Extended Regular Closure Property

$$L = \{\, a^l b^m a^n b^q \mid l, m, n, q > 0 \wedge (l = n \vee m > q) \,\}$$

- L is **inherently ambiguous**.
- L does not have a grammar with the extended regular closure property.

Question. Are there any CFLs that are not inherently ambiguous that have no grammar with the extended regular closure property?

Star-Free Closure Property

$$\boxed{\begin{aligned} S &\to aA \mid aD_1bS \mid bB \mid bD_1^RaS \\ A &\to \varepsilon \mid aA \mid aD_1bA \\ D_1 &\to \varepsilon \mid aD_1bD_1 \\ B &\to \varepsilon \mid bB \mid bD_1^RaB \\ D_1^R &\to \varepsilon \mid bD_1^RaD_1^R \end{aligned}}$$

$$\overline{O_1} = \{\, x \in \{a,b\}^* \mid |x|_a \neq |x|_b \,\}$$

$$\begin{aligned} A &= \{\, x \in \{a,b\}^* \mid \forall uv(x = uv \to |u|_a \geq |u|_b) \,\} \\ &= \{\, x \in \{a,b\}^* \mid \neg\exists uv(x = uv \wedge |u|_a + 1 = |u|_b) \,\} \\ &= \overline{O_1 b \{a,b\}^*} \\ &= [\![\overline{\overline{\langle\!\langle \varepsilon, \varepsilon \rangle\!\rangle}\, b\, (a \cup b)^*}]\!]^{\overline{O_1}} \\ &= [\![\overline{\overline{\langle\!\langle \varepsilon, \varepsilon \rangle\!\rangle}\, b\, \overline{\varnothing}}]\!]^{\overline{O_1}} \end{aligned}$$

star-free expression over query atoms

A star-free expression is an extended regular expression that does not contain Kleene star (*).

Extended Regular Closure vs. Star-Free Closure

Question. Are there any CFLs that have a grammar with the extended regular closure property but have no grammar with the star-free closure property?

$$L = \{\, a^n b^m c^l \mid (n \text{ is odd} \wedge n > m) \vee (n \text{ is even} \wedge n > l) \,\}$$

L has a grammar with the extended regular closure property, but we do not know whether it has a grammar with the Boolean closure property. Note that (aa)* is not a star-free regular language (it has no star-free expression).