Grammatical inference and subregular phonology

Adam Jardine, Rutgers University

Tel Aviv Winter School
December 9-12, 2019

1 Overview

• The interplay between linguistic universals and acquisition is at the heart of explanation in generative linguistics:

> "[V]arious formal and substantive universals are intrinsic properties of the language-acquisition system, these providing a schema that is applied to data and that determines...the grammar that may emerge upon presentation of appropriate data."

Chomsky, *Aspects*, p. 53

• Results in computational learning theory agrees with this approach:

> "[I]f an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems."

Wolpert and Macready (1997), p. 69, in one of the "No Free Lunch" papers

• Course description:

> "The **subregular hypothesis** identifies computational universals of phonological patterns. **Grammatical inference** ... allows us to develop learning procedures that take advantage of these universals. ...

subregular hypothesis

grammatical inference

• Rough overview:

Day 1: Learning, languages, and grammars

Day 2: Learning strictly local grammars

Day 3: Automata and input strictly local functions

Day 4: Learning functions and stochastic patterns, other open questions

• By the end of this course, you should be able to engage with the literature, and start your own research project!

2 Defining learning

2.1 What is learning?

• What do we mean when we say a child/animal/machine has 'learned' something?

• What do we mean when we say a child learned their language?

• What is the nature of the **sample**?

sample

• When is learning successful?

2.2 Grammatical inference

- Grammatical inference is a subfield of computer science that aims to formalize these questions
- A **grammar** is a finite representation of a (potentially infinite) language
- Grammatical inference **studies algorithms that solve the problem of inducing a grammar from a finite sample of data**
- Two prongs of attack:
  - **formal** grammatical inference
  - **empirical** grammatical inference

Canonical text: de la Higuera (2010), *Grammatical Inference*

grammar

formal/empirical grammatical inference

- We are going to focus on theoretical grammatical inference:
  - What is the learning problem?
  - What is an algorithm that *solves* the problem?

> Learning is going to be measured as the convergence toward a stable and good solution. In an ideal world, one would hope to have this convergence depend on a magic number: as soon as a given quantity of information or data is available, the intended grammar would be learned. ... But many things can go wrong: the data may not be representative or, even when it is, we may be facing some intractable problem. It is therefore necessary to impose some conditions on the data in order to secure a learning result, which will therefore always be read as: provided the data available has a minimal quality (with respect to a target and the criterion we impose), we can ensure that the solution is good. (Heinz et al., 2016, p.24)

3 Languages and Grammars

- What is a pattern?
- In phonology, there are essentially two kinds of patterns:
  Well-formedness (phonotactics) *NC̥
  Transformations (processes) /NC̥/ → [NC̬]
- Phonotactics described **sets** of words:

  The set of *well-formed* words according to *NC̥ is

  {an, anda, amba, lalalalanda, blık, ffffffff, ...}

  and the set of *ill-formed* words is

  {anta, ampa, lalalalaŋka, ...}.

- Transformations are **relations** pairing underlying forms with surface forms:
  /NC̥/ → [NC̬]

  {(an, an), (anda, anda), (anta, anda), (lalalalampa, lalalalamba),...}

3.1 Formal languages

- Some definitions:

  Alphabet (Σ) — **alphabet,** Σ

  String w — **string**

  The **empty string** λ is — **empty string,** λ

  Σ^* is — Σ^*

  A **formal language** (or often just **language**) L is — **(formal) language** L

  Alternatively, we can think of L as a function mapping each string in Σ^* to either $\top$ (true) or $\bot$ (false). — $\top, \bot$

- Some other notation:
  - $(w)^n$ is n repetitions of w (e.g. $b^4 = bbbb$, $(ab)^3 = ababab$, etc.) — $(w)^n$
  - $|w|$ is the length of w (e.g. $|\lambda| = 0$, $|ababab| = 6$, etc.) — $|w|$
  - $w{\cdot}v$ or just wv is the **concatenation** of w and v (e.g. $ab{\cdot}ba = abba$). — **concatenation** $w \cdot v, wv$

3.2 Language classes

- There are a lot of formal languages. In fact, many languages are not even **computable**—that is, there is no finite procedure that can determine all and only the strings in the language. — **computable**
- We will restrict ourselves to languages that are computable. The set of all computable languages is a **class**. A class, usually denoted $\mathcal{C}$, is a set of languages. — **class** $\mathcal{C}$
- Interesting classes are those characterized by some abstract property (such as computability). Some examples are given below and in Figure 1. This will make more sense when we see some examples.
- Importantly, the property that defines a class can[1] lead to a learning procedure for the class. We'll show this with a very specific linguistic example.

[1] Not always!

finite (Fin)
strictly local (SL)
regular (Reg)
context free (CF)

Example 1 *Let $\Sigma = \{a, b\}$. Example languages are the set of strings...*

- $L_1 = \{a, bb, aaba\}$ ***(finite)***
- *of the form* $(ab)^n$, $L_2 = \{\lambda, ab, abab, ababab, ...\}$ ***(strictly local)***
- *of the form* $(aa)^n$, $L_3 = \{\lambda, aa, aaaa, aaaaaa, ...\}$ ***(regular)***
- *of the form* a^nb^n, $L_4 = \{\lambda, ab, aabb, aaabbb, ...\}$ ***(context free)***

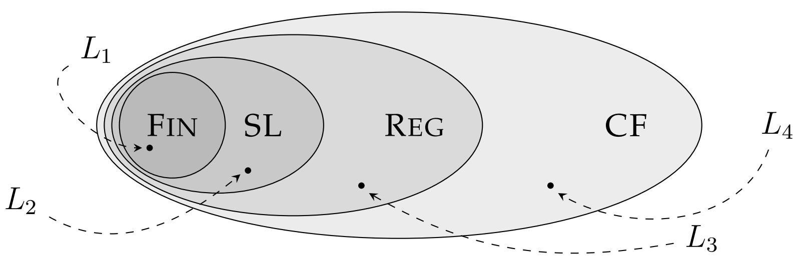


Figure 1: Some linguistically important classes of stringsets

3.3 Strictly local languages and grammars

- We're going to add special boundary symbols $\rtimes, \ltimes \notin \Sigma$ — $\rtimes, \ltimes$
- Let $\rtimes\Sigma^*\ltimes$ refer to the set of strings $\rtimes w \ltimes$ for $w \in \Sigma^*$ — $\rtimes\Sigma^*\ltimes$

Definition 1 (substring) *A string u is a* substring *of another string w iff $w = v_1uv_2$ for some other strings v_1 and v_2.* — **substring**

Definition 2 (k-factor) *A string u is a k-*factor *of another string w iff either:* — **k-factor**

- *$|u| = k$, $|\rtimes w \ltimes| \geq k$, and u is a substring of $\rtimes w \ltimes$; or*
- *$|\rtimes w \ltimes| < k$ and $u = \rtimes w \ltimes$*

- We write $\texttt{fac}_k(w)$ for the set of k-factors of w. — $\texttt{fac}_k(w)$

$$\texttt{fac}_2(abbab) = \{\rtimes a, ab, bb, ba, ab, b\ltimes\}$$

- Other examples:
  - $\texttt{fac}_2(abab) =$
  - $\texttt{fac}_3(aaba) =$
  - $\texttt{fac}_6(aaba) =$
- For a set L of strings, the k-factors of L are $\texttt{fac}_k(L) = \bigcup_{w \in L} \texttt{fac}_k(w)$ — $\texttt{fac}_k(L)$

Definition 3 (SL$_k$ grammar) *A* ***SL$_k$ grammar*** *is a set G of k-factors of Σ^*.* — **SL$_k$ grammar**

A string $w \in \Sigma^*$ ***satisfies*** *G, $w \models G$, if none of the k-factors of w are in the set G; i.e. $\texttt{fac}_k(w) \cap G = \emptyset$.* — **satisfaction ($\models$)**

The set $L(G)$ is the set of strings that satisfy G, i.e.

$$L(G) = \{w \in \Sigma^* \mid w \models G\}$$

- We often call G for L the set of **forbidden k-factors** of L — **forbidden k-factors**
- A language L is **SL** iff it is SL$_k$ for some k. — **SL**
- Let's do some examples.

a. What is a SL$_2$ grammar for the set $(ab)^n$?

b. Let $\Sigma = \{C, V\}$. What is a SL$_3$ grammar for the set of strings over Σ that satisfy the generalization "C does not occur three times in a row"?

c. Consider $\Sigma = \{\sigma, \acute{\sigma}\}$ and the stress pattern of Pintupi:[2]

$\acute{\sigma}\sigma$
$\acute{\sigma}\sigma\sigma$
$\acute{\sigma}\sigma\acute{\sigma}\sigma$
$\acute{\sigma}\sigma\acute{\sigma}\sigma\sigma$
$\acute{\sigma}\sigma\acute{\sigma}\sigma\acute{\sigma}\sigma$
$\acute{\sigma}\sigma\acute{\sigma}\sigma\acute{\sigma}\sigma\sigma$
$\acute{\sigma}\sigma\acute{\sigma}\sigma\acute{\sigma}\sigma\acute{\sigma}\sigma$

Is there a SL$_2$ grammar for this pattern? Is there a SL$_3$ grammar?

[2] Hansen and Hansen (1969)

d. Is there a SL_2 grammar for $(aa)^n$? A SL_3 grammar? A SL_k grammar for any k?

e. How about the following pattern from Ineseño Chumash?

(Applegate, 1972; Heinz, 2010)

ʃ-api-tʃ$^{\mathrm{h}}$ol-it	'I have a stroke of good luck'
s-api-ts$^{\mathrm{h}}$ol-us	'he has a stroke of good luck'
ʃ-api-tʃ$^{\mathrm{h}}$ol-uʃ-waʃ	'he had a stroke of good luck'
ha-ʃxintila-waʃ	'his former Indian name'
s-is-tisi-jep-us	'they (two) show him'
k-ʃu-ʃojin	'I darken it'

- Other major classes identified as relevant to phonotactics are given in the chart in Fig 2. References:
  - **Tier-based strictly local (TSL)** languages – Heinz et al. (2011); McMullin (2016)
  - **Strictly piecewise (SP)** languages – Heinz (2010)
  - For a formal review of these and other subregular classes, and how they relate to natural language stress patterns, see Rogers et al. (2013).

tier-based strictly local (TSL)

strictly piecewise (SP)

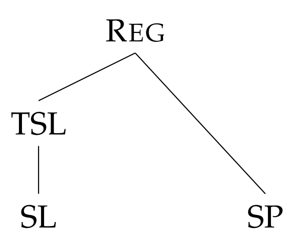


Figure 2: Hierarchy of subregular language classes related to phonotactics.

4 Learning SL languages

4.1 Learning paradigm

- An influential framework is Gold (1967)'s **identification in the limit from positive data (ILPD)**

identification in the limit from positive data (ILPD)

- We assume the learner is learning from a **text**; that is, the learner cannot make requests to the oracle, it only receives examples

text

- A **positive presentation** p of a stringset L is an infinite sequence

positive presentation p

$$p(0), p(1), p(2), ...$$

such that for every $w \in L$, there is some i such that $p(i) = w$.

- Let $p[i]$ denote the finite sequence $p(0), p(1), ..., p(i)$.

$p[i]$

- A learner is an algorithm $\mathcal{A}$ is a function that takes as input a finite sequence and returns a grammar; that is, $\mathcal{A}(p[i]) = G$, where G is a finite representation for a stringset. This G is called the learner's **hypothesis** given $p[i]$.

hypothesis

- A learner is said to **converge** on a presentation p if there is some n such that for all $m > n$, $\mathcal{A}(p[n]) = \mathcal{A}(p[m])$.

converge

Data	$p(0)$	$p(1)$	$p(2)$	...	$p(n)$	$p(n+1)$	$p(n+2)$	...	$p(m)$	...
Hyp.	G_0	G_1	G_2	...	G_n	G_n	G_n	...	G_n	...

Figure 3: Convergence

- A class $\mathcal{C}$ is **ILPD-learnable** if there is some algorithm $\mathcal{A}_\mathcal{C}$ such that for *any* stringset $L \in \mathcal{C}$, given *any* positive presentation p of L, $\mathcal{A}_\mathcal{C}$ converges to a grammar G such that $L(G) = L$.

ILPD-learnable

- Let's show that the Finite class is ILPD-learnable.

- How is ILPD learning an idealization?

- What are the advantages of using ILPD as a criterion for learning?

4.2 Learning SL grammars

- The class of SL_k for a specific k is ILPD-learnable
- Consider a SL_k language $L_\star$ representable by some SL_k grammar $G_\star$. We are looking for a procedure that converges to a grammar G such that $L(G) = L_\star$ from positive examples of $L_\star$.

$$G_\star = \{CC, C\ltimes\}$$

	Data	Hypothesis
0	V	
1	$CVCV$	
2	$CVVCVCV$	
3	$VCVCV$	

- Let's check if the learner has **generalized**.
  Does the grammar it converges to accept $CVCVV$? How about $CVCVVC$?

generalization

- Let's call our algorithm $\mathcal{A}_{\mathrm{SL}_k}$.
  Assuming the target language is SL_k, in general, the hypothesis G_i at time step i is

  $$G_i = \mathcal{A}_{\mathrm{SL}_k}(p[i]) =$$

$\mathcal{A}_{\mathrm{SL}_k}$

- For a target $L_\star$, a **characteristic sample** for a learner $\mathcal{A}$ is a set $D \subseteq L_\star$ such that if a sequence $p[i]$ of a positive presentation of $L_\star$ contains D, then $\mathcal{A}$ is guaranteed to converge to $L_\star$.

  What is the characteristic sample for the SL_k learner for any $L_\star$ whose grammar is $G_\star$?

characteristic sample (de la Higuera, 2010)

- The **time complexity** of a learner $\mathcal{A}$ is the number of steps $\mathcal{A}$ takes to compute a hypothesis given some sequence $p[i]$, relative to the size of $p[i]$.

  The time complexity of $\mathcal{A}_{\mathrm{SL}_k}$ is **linear**, that is, the time it takes to run on any $p[i]$ is directly proportional to the size of $p[i]$. This means $\mathcal{A}_{\mathrm{SL}_k}$ is extremely **efficient**, and thus cognitively plausible.

time complexity

linear

efficient

- For extra practice, let's learn Pintupi. Note that $k = 3$. What is the initial hypothesis? At what point do we converge?

t	datum	hypothesis
0	$\acute{\sigma}$	
1	$\acute{\sigma}\sigma$	
2	$\acute{\sigma}\sigma\sigma$	
3	$\acute{\sigma}\sigma\acute{\sigma}\sigma$	
4	$\acute{\sigma}\sigma\acute{\sigma}\sigma\sigma$	
5	$\acute{\sigma}\sigma\acute{\sigma}\sigma\acute{\sigma}\sigma$	

4.3 The limits of SL learning

- SL_k is ILDP-learnable, but *SL in general is not*. This follows from the following two facts:
  - Any class $\mathcal{C}$ such that $\text{FIN} \subsetneq \mathcal{C}$ is not learnable from positive data only (Gold, 1967).
  - $\text{FIN} \subsetneq \text{SL}$
- How about the long-distance assimilation pattern from Chumash?

- The learning algorithm $\mathcal{A}_{\text{SL}_k}$ **only learns SL_k languages**. We know exactly **what patterns it can learn** and **what patterns it cannot learn**, and **on exactly what data**. **This is the advantage of studying learning with formal grammatical inference.**

4.4 Connection to learning phonotactics

- Learning mechanisms for the TSL and SP languages (recall from Fig 2) are based on very similar mechanisms.
  References:
  - Learning TSL languages – Heinz et al. (2011); Jardine and Heinz (2016); Jardine and McMullin (2017)
  - Learning SP languages – Heinz (2010); Heinz and Rogers (2013)

5 Learning with strictly local automata

> "It is always a pleasant surprise when two formalisms, introduced with different motivations, turn out to be equally powerful, as this indicates that the underlying concept is a natural one."

Engelfriet and Hoogeboom (2001, p. 216)

5.1 Strictly local acceptors

- Another formalism for studying formal languages are **automata**. Automata are abstract machines that perform computations in some kind of well-defined way.

automata

- A (deterministic) **finite-state acceptor (FSA)** is
  - An input alphabet Σ
  - A finite set Q of **states**
  - A single initial state $q_0 \in Q$
  - A set $F \subseteq Q$ of **final states**
  - A **transition function** $\delta : Q \times \Sigma \to Q$

finite-state acceptor (FSA)

states Q

final states F

transition function δ

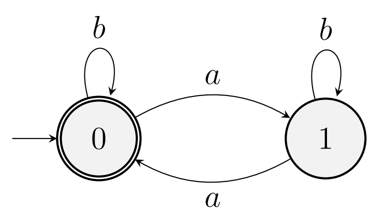


Figure 4: FSA for the language of strings over $\{a, b\}$ that contain an even number of as. States are circles with accepting states marked with double circles, arrows mark transition from state to state, and the state 0 with an unlabeled incoming arrow with no source state is the initial state.

- The REG class is exactly those languages that are describable by FSAs.

Recall REG from Fig. 1

- A **strictly k-local FSA (SL$_k$FSA)** is a FSA that has exactly one state per $k-1$ factor of Σ^*.

strictly k-local FSA (SL$_k$FSA)

- SL$_k$FSAs describe exactly the SL$_k$ languages.

5.2 Learning SLFSAs

- FSAs are useful because there are a number of learning techniques that make use of them.

For an overview, see Heinz et al. (2016), Chapter 3.

- We're going to study a lesser-used, but incredibly useful, technique of 'transition-filling,' described (originally?) in Heinz and Rogers (2013).
- Let's add to our FSAs an **output function** and a **ending function** that output to the boolean values $\{\top, \bot\}$
  - An **output function** $\omega : Q \times \Sigma \rightarrow \{\top, \bot\}$
  - A **ending function** $\epsilon : Q \rightarrow \{\top, \bot\}$

output function ω

ending function ϵ

- These work the same as usual acceptors, just that acceptance is instead based on **only traversing** $\top$ **transitions**, and **ending on** $\top$ **states**.

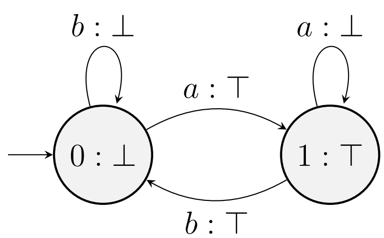
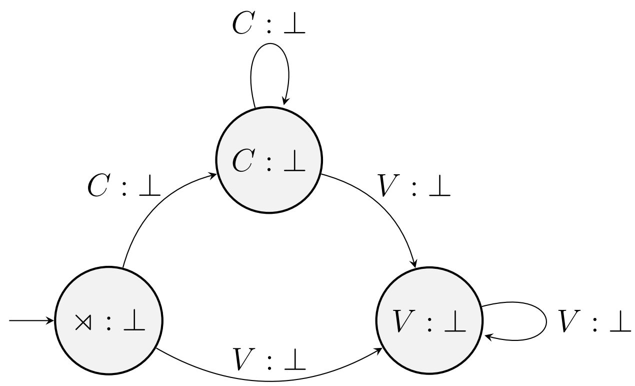


Figure 5: FSA for the language $(ab)^n$ with output function (marked on transitions after the colon) and ending function (marked on states after the colon).

- As **all SL$_k$ transducers** share the **same structure** of transitions, we can learn by the following procedure:
  - initial hypothesis is the SL$_k$ representation of the empty language.[3]
  - we set to $\top$ the output of any transition that is taken by any string in the data presentation.
  - we set to $\top$ the output of any state that any string in the data presentation ends on.
- Let's see how this works with an example. Below is the standard SL$_k$ structure for the alphabet $\{C, V\}$. How does the above procedure change the outputs on the transitions?

[3] =the empty set, {}

	data
0	CV
1	$CVCV$
2	$CVCVCV$
3	$VCVCV$



6 Input-strictly local functions and learning processes

6.1 Subsequential functions

- Consider an **input alphabet** Σ and an **output alphabet** Γ.
- A **relation** is some set $R \subseteq \Sigma^* \times \Gamma^*$, that is strings in Σ^* paired with strings in Γ^*.

  {(an, an), (anda, anda), (anta, anda), (lalalalampa, lalalalamba),...}

- R is a **function** iff $(w, v) \in R$ and $(w, u) \in R$ implies $v = u$. We'll only be considering functions today.
- We can create **subsequential finite-state transducers (SFSTs)** by taking our output and ending functions and changing their outputs from values in $\{\top, \bot\}$ to strings in Γ^*.[4]
  - An input alphabet Σ and an output alphabet Γ
  - A finite set Q of states
  - A single initial state $q_0 \in Q$
  - A set $F \subseteq Q$ of final states
  - A transition function $\delta : Q \times \Sigma \to Q$
  - An output function $\omega : Q \times \Sigma \to \Gamma^*$
  - A ending function $\epsilon : Q \to \Gamma^*$
- Let's do some examples.

input alphabet
output alphabet
relation
function
subsequential finite-state transducers (SFSTs)

[4] I mean *subsequential* in the sense of Schützenberger and Mohri; other authors (e.g. Filiot and Reynier 2016) use the term *sequential* for the same class.

a. Write a FST for (a) interpreted *non-iteratively*. Examples are given below.

(a) $a \;\rightarrow\; b \,/\, b\,\_\_$

aba	$\mapsto$	abb
$baba$	$\mapsto$	$bbbb$
$baaaa$	$\mapsto$	$bbaaa$

b. Write a FST for (b) interpreted *iteratively*.

(b) $a \;\rightarrow\; b \,/\, b\,\_\_$

aba	$\mapsto$	abb
$baba$	$\mapsto$	$bbbb$
$baaaa$	$\mapsto$	$bbbbb$

c. Write a FST for rule in (c), interpreted *non-iteratively*.

(c) $a \;\rightarrow\; b \,/\, \_\_\, b$

aba	$\mapsto$	bba
$baba$	$\mapsto$	$bbba$
$aaaab$	$\mapsto$	$aaabb$

d. In Kikongo, the liquid /l/ becomes [n] after another nasal: Ao (1991)

(d) $l \;\rightarrow\; n \,/\, m \ldots \_\_$

$tala$	$\mapsto$	$tala$
$mala$	$\mapsto$	$mana$
$matala$	$\mapsto$	$matana$

- Functions describable with SFSTs are called **(left-)subsequential functions**. Functions describable with SFSTs working in reverse are called the **right-subsequential functions**.

  (left-)subsequential functions

  right-subsequential functions
- Recent work has looked at sub-classes of the subsequential functions and as it relates to phonology. References:
  - Right/left-subsequential functions – Mohri (1997); Heinz and Lai (2013); Heinz (2018)
  - **Input strictly local (ISL) functions, (right- and left-)output strictly local (OSL) functions** – Chandlee (2014); Chandlee et al. (2015); Chandlee and Heinz (2018)

    input strictly local (ISL)

    (right-/left-)output strictly local (OSL)

- For a formal definition of a version of SFSTs that can deal with optionality, see Beros and de la Higuera (2016).

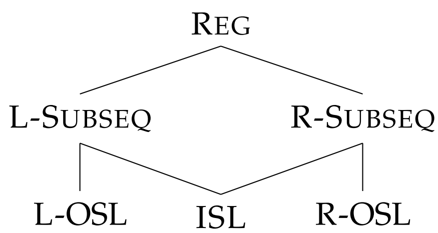


Figure 6: Hierarchy of subregular function classes related to phonology.

6.2 Input strictly local functions

- The ISL functions are exactly those whose SFSTs have states who represent $k-1$ suffixes.
- Examples (a) and (c) above are ISL, but the others are not.
- 94% of the processes in P-Base (Mielke, 2004) are ISL (Chandlee and Heinz, 2018).

6.3 Learning input strictly local functions

- The following is based on the "transition-filling" technique from Jardine et al. (2014).
- The target is an ISL function f; input data is from $d \subset f$
- A **prefix** of w is a string u s.t. $w = uv$ for some string v **prefix**
- Let $u^{-1}w = v$ s.t. $w = uv$ (if u is a pref. of w, undefined otherwise)

- The **common prefixes** of a set L is the set **common prefixes**

$$\mathrm{cmnprfs}(L) = \{u \mid \forall w \in L, u \text{ is a prefix of } w\}$$

- The **longest common prefix (lcp)** of a set L of strings is **longest common prefix (lcp)**

$$\mathrm{lcp}(L) = w \in \mathrm{cmnprfs}(L) \text{ s.t. } \forall v \in \mathrm{cmnprfs}(L), |w| \geq |v|$$

- For d we define d^p as d^p

$$d^p(w) \overset{\text{def}}{=} \text{lcp}(d(w\Sigma^*))$$

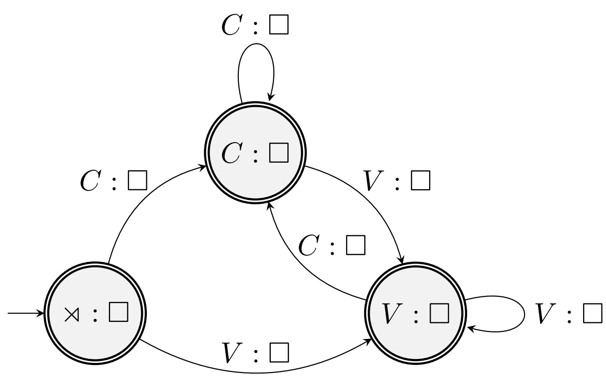
- For w, we define d_w as d_w

$$d_w(u) = d^p(w)^{-1}d(wu)$$

- Then d^p_w is d^p_w

$$d^p_w(u) \overset{\text{def}}{=} \text{lcp}(d_w(u\Sigma^*))$$

- We then start with a 'blank' ISL$_k$ SFST like the one below.



- We then fill the output for the transition on σ from state q as

$$\sigma : d^p_w(\sigma)$$

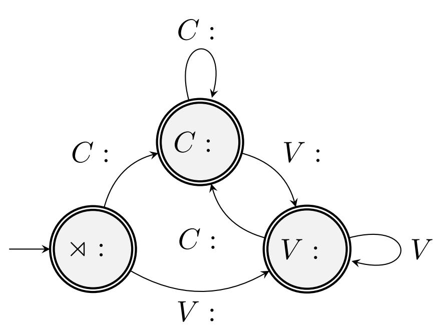
for some w that reaches q, and likewise the ending transition as

$$q : d^p(w)^{-1}d(v)$$

for any w, v that reach q.

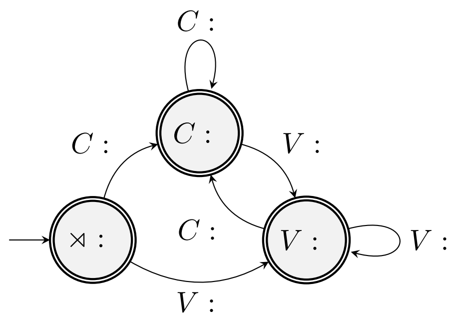
- Let's work on some examples using the above machine:
  - $V \to \emptyset \,/\, V$ ___

(CVC, CVC)
(CVV, CV)
$(CVCCV, CVCCV)$
$(CCVCC, CCVCC)$
$(CCCVCV, CCCVCV)$
$(CVVCV, CVCV)$
(V, V)



– $\emptyset \rightarrow V \,/\, C \,\_\_\, \{C, \#\}$

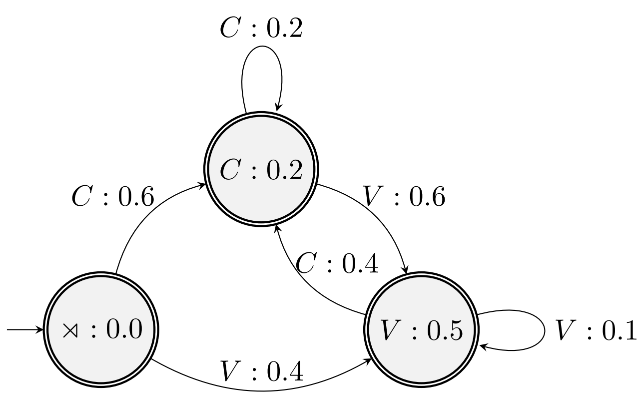
$(CVC, CVCV)$
(CVV, CVV)
$(CVCCV, CVCCV)$
$(CCVCC, CCVCCV)$
$(CCCVCV, CCCVCV)$
$(CVCV, CVCV)$
$(CVVCV, CVVCV)$
(V, V)



- This algorithm ILPD-learns any class whose functions share a state & transition structure
- This very general idea extends to other kinds of learning:
  - Learning URs (Hua et al. in progress)
  - Learning OSL (Chandlee et al., 2015) tier-based OSL (Burness and McMullin, 2019) use a similar (yet distinct) method
  - Learning for optional ISL processes uses the same basic idea (Heinz in progress) based on Beros and de la Higuera (2016)

6.4 Strictly Local Distributions

- We can replace strings in Γ^* with numbers between 0 and 1

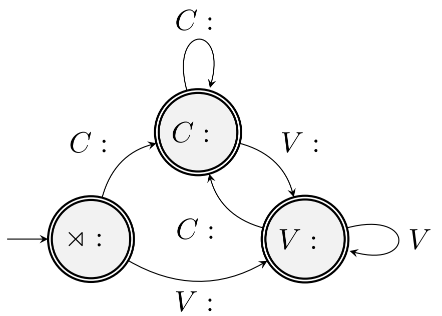


- Take a sequence of data. We begin with 0s on the transitions, and:
  - add 1 to each transition whenever it is traversed
  - divide each count by the times a state has been visited

(=the total number of outgoing transitions + times ended on each state)

- In the limit, this is guaranteed to **approach** the distribution from which the sequence is drawn
- Let's try an example:

  CVC
  CVV
  $CVCCV$
  $CVCVC$
  $CVCV$
  $CVVCV$



- Related work:
  - Learning strictly piecewise distributions: Heinz and Rogers (2010)
  - Learning SL distributions over features: Heinz and Koirala (2010)

7 Looking ahead

- Open questions:
  - Learning using logic and non-string models (Strother-Garcia et al., 2016)
  - Learning using features (Chandlee et al., 2019)
  - Learning URs (Hua et al., in progress)
  - Learning optionality (Heinz et al., in progress)
  - ...
- Alëna Aksënova's *kist* package codes a lot of this in Python, available at https://github.com/alenaks/SigmaPie

References

Ao, B. (1991). Kikongo nasal harmony and context-sensitive underspecification. *Linguistic Inquiry*, 22(2):193–196.

Applegate, R. B. (1972). *Ineseño Chumash grammar*. PhD thesis, University of California, Berkeley.

Beros, A. and de la Higuera, C. (2016). A canonical semi-deterministic transducer. *Fundamenta Informaticae*, pages 431–459.

Burness, P. and McMullin, K. (2019). Efficient learning of output tier-based strictly 2-local functions. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 78–90, Toronto, Canada. Association for Computational Linguistics.

Chandlee, J. (2014). *Strictly Local Phonological Processes*. PhD thesis, University of Delaware.

Chandlee, J., Eyraud, R., and Heinz, J. (2015). Output strictly local functions. In Kornai, A. and Kuhlmann, M., editors, *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 14)*, pages 52–63, Chicago, IL.

Chandlee, J., Eyraud, R., Heinz, J., Jardine, A., and Rawski, J. (2019). Learning with partially ordered representations. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 91–101, Toronto, Canada. Association for Computational Linguistics.

Chandlee, J. and Heinz, J. (2018). Strictly locality and phonological maps. *LI*, 49:23–60.

Chomsky, N. (1965). *Aspects of Theory of Syntax*. Cambridge, Massachusetts: MIT Press.

de la Higuera, C. (2010). *Grammatical Inference: Learning Automata Grammars*. Cambridge University Press.

Engelfriet, J. and Hoogeboom, H. J. (2001). MSO definable string transductions and two-way finite-state transducers. *ACM Transations on Computational Logic*, 2:216–254.

Filiot, E. and Reynier, P. (2016). Transducers, logic, and algebra for functions of finite words. *ACM SIGLOG News*, 3(3):4–19.

Gold, M. E. (1967). Language identification in the limit. *Information and Control*, 10:447–474.

Hansen, K. and Hansen, L. (1969). Pintupi phonology. *Oceanic Linguistics*, 8:153–170.

Heinz, J. (2010). Learning long-distance phonotactics. *LI*, 41:623–661.

Heinz, J. (2018). The computational nature of phonological generalizations. In Hyman, L. and Plank, F., editors, *Phonological Typology*, Phonetics and Phonology, chapter 5, pages 126–195. De Gruyter Mouton.

Heinz, J., de la Higuera, C., and van Zaanen, M. (2016). *Grammatical Inference for Computational Linguistics*. Number 28 in Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Heinz, J. and Koirala, C. (2010). Maximum likelihood estimation of feature-based distributions. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, pages 28–37, Uppsala, Sweden. Association for Computational Linguistics.

Heinz, J. and Lai, R. (2013). Vowel harmony and subsequentiality. In Kornai, A. and Kuhlmann, M., editors, *Proceedings of the 13th Meeting on Mathematics of Language*, Sofia, Bulgaria.

Heinz, J., Rawal, C., and Tanner, H. G. (2011). Tier-based strictly local constraints for phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 58–64, Portland, Oregon, USA. Association for Computational Linguistics.

Heinz, J. and Rogers, J. (2010). Estimating strictly piecewise distributions. In *Proceedings of the 48th Annual Meeting of the ACL*. Association for Computational Linguistics.

Heinz, J. and Rogers, J. (2013). Learning subregular classes of languages with factored deterministic automata. In Kornai, A. and Kuhlmann, M., editors, *Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13)*, pages 64–71, Sofia, Bulgaria. Association for Computational Linguistics.

Jardine, A., Chandlee, J., Eyraud, R., and Heinz, J. (2014). Very efficient learning of structured classes of subsequential functions from positive data. In *Proceedings of the 12th International Conference on Grammatical Inference (ICGI 2014)*, JMLR Workshop Proceedings, pages 94–108.

Jardine, A. and Heinz, J. (2016). Learning tier-based strictly 2-local languages. *Transactions of the Association for Computational Linguistics*, 4:87–98.

Jardine, A. and McMullin, K. (2017). Efficient learning of tier-based strictly k-local languages. In Drewes, F., Martín-Vide, C., and Truthe, B., editors, *Language and Automata Theory and Applications, 11th International Conference*, Lecture Notes in Computer Science, pages 64–76. Springer.

McMullin, K. (2016). *Tier-based locality in long-distance phonotactics: learnability and typology*. PhD thesis, University of British Columbia.

Mielke, J. (2004). P-Base 1.95. http://137.122.133.199/jeff/pbase.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

Rogers, J., Heinz, J., Fero, M., Hurst, J., Lambert, D., and Wibel, S. (2013). Cognitive and sub-regular complexity. In *Formal Grammar*, volume 8036 of *Lecture Notes in Computer Science*, pages 90–108. Springer.

Strother-Garcia, K., Heinz, J., and Hwangbo, H. J. (2016). Using model theory for grammatical inference: a case study from phonology. In Verwer, S., Menno van Zaane, n., and Smetsers, R., editors, *Proceedings of The 13th International Conference on Grammatical Inference*, volume 57 of *JMLR: Workshop and Conference Proceedings*, pages 66–78.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.