Theoretical Computational Linguistics: Learning Theory

Jeffrey Heinz

October 19, 2017

Contents

1 Analyzing Learning Computationally **5**
- 1.1 Strings and stringsets 5
- 1.2 The membership problem 5
- 1.3 Learning problems 7
- 1.4 Generalizing a little bit 8
- 1.5 Classifying membership problems 8

Appendices **13**
- 1.A Enumerating Σ^* 13
- 1.B Non-enumerable stringsets 14

2 Identification in the Limit from Positive Data **15**
- 2.1 Identification in the limit 15
  - 2.1.1 Identification in the limit from positive data 16
  - 2.1.2 The Strictly k-Piecewise Stringsets 17
  - 2.1.3 The Strictly k-Local Stringsets 19
  - 2.1.4 Strictly k-Local Treesets 21

3 Identification in the Limit: General Results **27**
- 3.1 Identification in the limit from positive and negative data 27
- 3.2 Variations on a theme 28
- 3.3 Important Results 28

4 Generalizing Strictly Local Learning **31**

Chapter 1

Analyzing Learning Computationally

1.1 Strings and stringsets

A string is a finite sequence of symbols from some set of symbols Σ. The set of all possible strings is often noted Σ^*. The asterisk is a symbol due to Kleene, who is one of the great computer scientists of the twentieth century. It is often called the 'Kleene star.' Generally, the presence of the Kleene star on a set denotes the **free monoid** of a set, which is the set of all finite sequences of length zero or more from that set. The unique string of length zero is often denoted λ or ϵ.

$\vdots$

aaa, aab, aba, abb, baa, bab, bba, bbb

aa, ab, ba, bb

a, b

λ

Figure 1.1: Strings of increasing length with $\Sigma = \{a, b\}$.

Here are some examples of sets of strings, also called **formal languages**, or **stringsets**.

1. Let $\Sigma = \{$`a, b, c, ..., z, .`$\}$. Then there is a subset of Σ^* which includes all and only the grammatical sentences of English (modulo capitalization).

2. Let $\Sigma = \{$`Advance-1cm, Turn-R-5°`$\}$. Then there is a subset of Σ^* which includes all and only the ways to get from point A to point B.

Exercise 1. Provide some more examples of stringsets relevant to linguistics.

1.2 The membership problem

The **membership problem** is the problem of deciding whether a string belongs to a set. The problem can be stated thusly: Given a set of strings S and *any* string $s \in \Sigma^*$, output whether $s \in S$. Is there an algorithm that solves this problem for a given S?

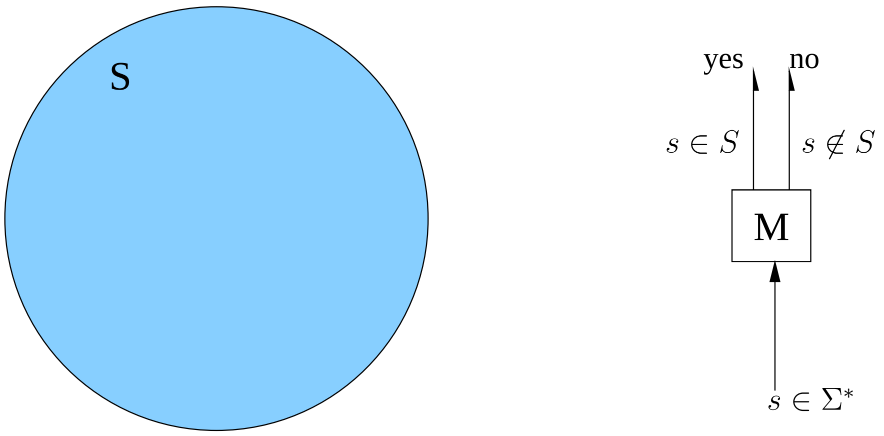
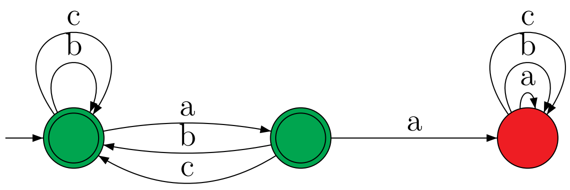


Figure 1.2: The membership problem

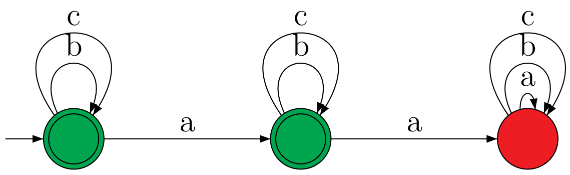
Example 1. A string belongs to S if it does not contain `aa` as a substring.

$s \in S$	$s \notin S$
abba	baab
abccba	aaccbb
babababa	ccaaccaacc
...	...



Example 2. A string belongs to S if it does not contain `aa` as a subsequence.

$s \in S$	$s \notin S$
cabb	baab
babccbc	babccba
bbbbbb	bbacccccccccaccc
...	...



Exercise 2. These finite-state machines are not the only algorithmic solutions to these membership problems? Provide other algorithms which solve these two membership problems.

1.3 Learning problems

There are many ways to define the problem of learning a stringset. Here are two informal ones just to get started.

1. For any set S from some given collection of sets: Drawing finitely many examples from S, output a program solving the membership problem for S.

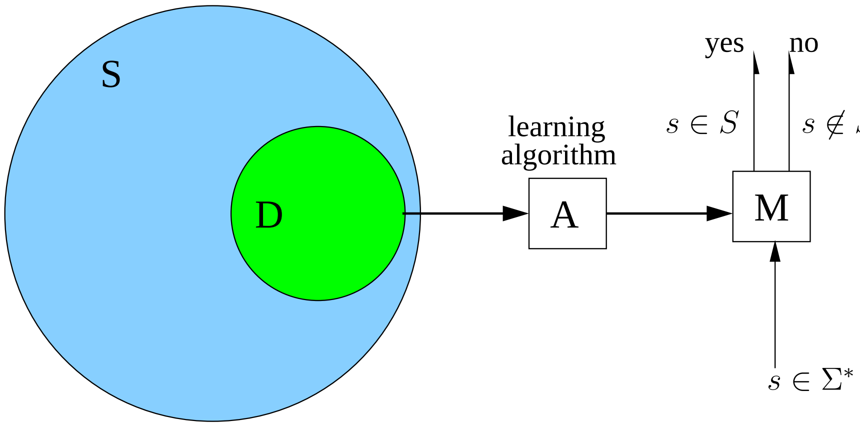


Figure 1.3: A Learning Problem with Only Positive Evidence

2. For any set S from some given collection of sets: Draw finitely many strings labeled as to whether they belong to S or not, output a program solving the membership problem for S.

These definitions are too informal. What does it mean to draw finitely many examples? Do I want the algorithm to succeed for any finite sample of strings providing information about the language? Why or why not?

Exercise 3. Improve the above definitions by making clearer what 'drawing' and 'output' mean. Try to write it formally if you can.

Exercise 4. Recall Osherson *et al.* (1986):
"Of special interest are the circumstances under which these hypotheses stabilize to an accurate representation of the environment from which the evidence is drawn. Such stability and accuracy are conceived as the hallmarks of learning. Within learning theory, the concepts 'evidence,' 'stabilization,' 'accuracy,' and so on, give way to precise definitions."
How do your improved definitions address these concepts?

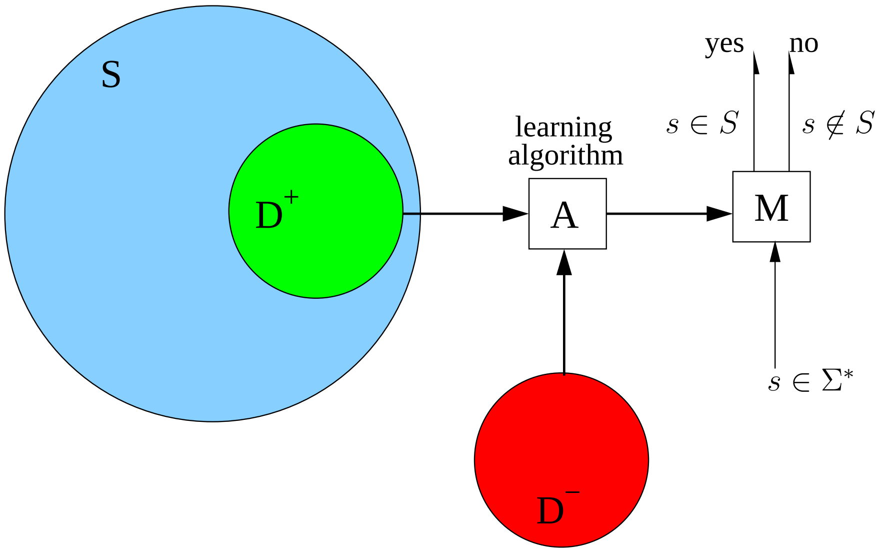


Figure 1.4: Learning Problem with Positive and Negative Evidence

1.4 Generalizing a little bit

The functional perspective lets us generalize the foregoing a little bit. From a functional perspective a stringset S is associated with a function $f : \Sigma^* \to \{0, 1\}$. But we may be interested in other types of functions which have Σ^* for a domain.

function	Notes
$f : \Sigma^* \to \{0, 1\}$	Binary classification
$f : \Sigma^* \to \mathbb{N}$	Maps strings to numbers
$f : \Sigma^* \to [0, 1]$	Maps strings to real values
$f : \Sigma^* \to \Delta^*$	Maps strings to strings
$f : \Sigma^* \to \wp(\Delta^*)$	Maps strings to sets of strings

Exercise 5. Provide some specific examples of functions like the ones above relevant to linguistics.

Exercise 6. Try to write another definition of learning. It can be for any of the types of functions shown.

1.5 Classifying membership problems

A basic question to ask is whether every stringset has a solution to the membership problem. Perhaps it is surprising to learn that the answer is No. In fact, as a consequence of work on computability in the mid-twentieth century it is known that most stringsets have no solution to the membership problem.

1. Enumerations of Σ^*.
2. No enumeration of $\wp(\Sigma^*)$.
3. Programs are of finite length so programs can be represented as strings of finite length.
4. This means every program is an element of Σ^*.
5. Consequenty there are at most countably many stringsets S which have programs which solve the membership problem of S (see 1).
6. But there are uncountably many stringsets (elements of $\wp(\Sigma^*)$, (see 2)).
7. So most stringsets have no solution to the membership problem.
8. Consequently any conceivable learning problems which targets a a non-enumerable stringset S has no solution because the learning algorithm cannot ultimately output a program which solves the membership problem for S. No such program exists or can exist!

The Chomsky Hierarchy provides additional classification of membership problems which have solutions (i.e. stringsets) (Chomsky, 1956; Hopcroft and Ullman, 1979).

Of particular interest is the class of regular languages, which may be defined as the class of stringsets whose membership problem can be solved by a computational device whose memory requirements are bounded and thus crucically do not grow without bound with respect to the length of the strings. Finite-state acceptors are one way to represent such computations.

As you may know, zooming in on the class of regular languages reveals some more structure (McNaughton and Papert, 1971; Thomas, 1982).

These subregular classes are shown below from a model-theoretic perspective (Rogers *et al.*, 2010, 2013).

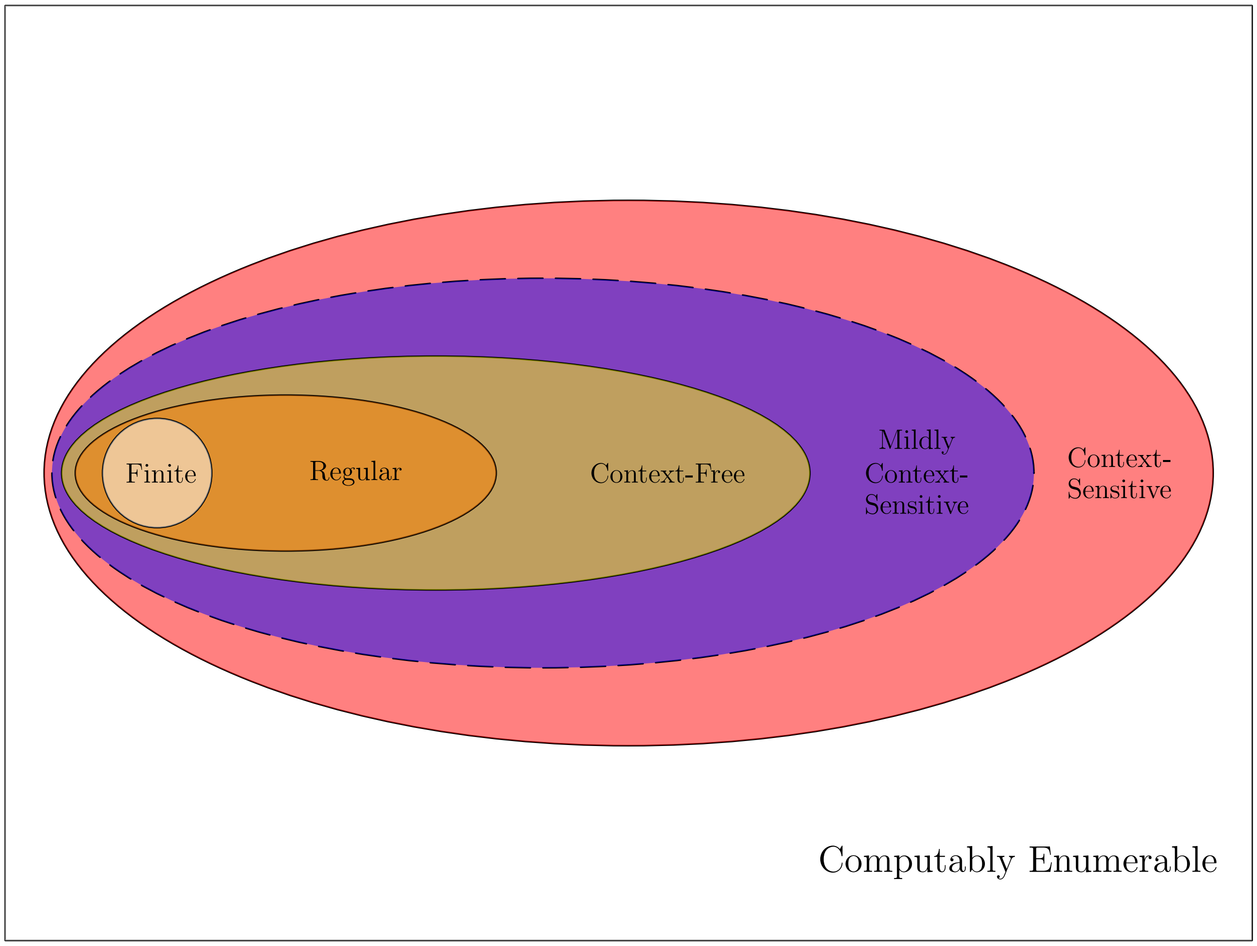


Figure 1.5: The Chomsky Hierarchy

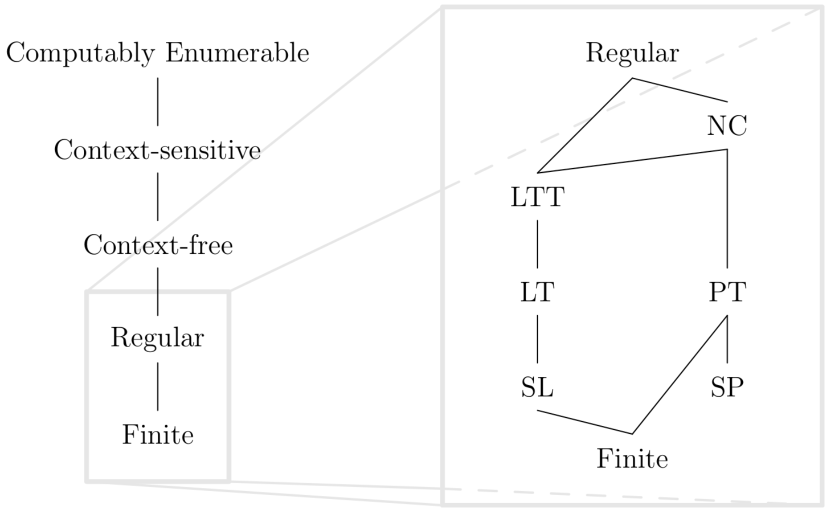


Figure 1.6: Room at the bottom.

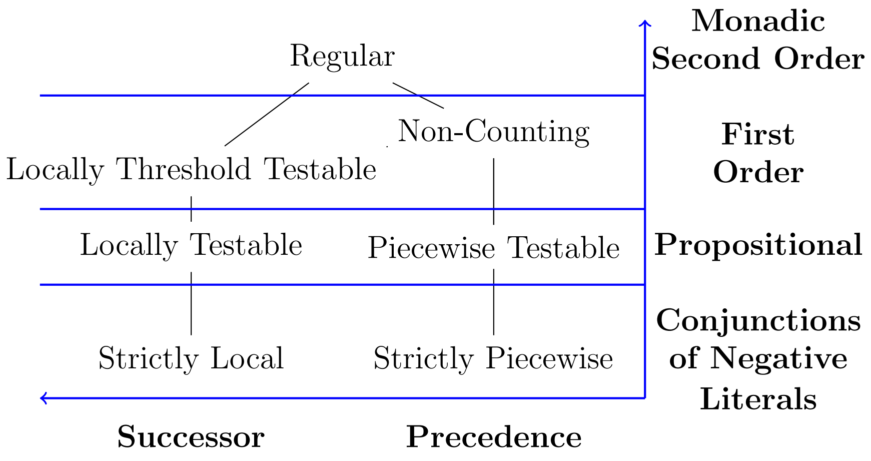


Figure 1.7: Subregular Hierarchies.

Appendix

1.A Enumerating Σ^*

The usual way to enumerate strings in Σ^* is to order them first by their length and then within strings of the same length to order them in dictionary order, as shown below.

0	λ	3	c	6	ac	...
1	a	4	aa	7	ba	
2	b	5	ab	8	bb	

Figure 1.A.1: Enumerating Σ^* with $\Sigma = \{a, b, c\}$.

A natural question that arises is what is the nth string in this enumeration? What effective procedure yields the nth string?

One way to find the nth string is to build a tree of all the strings in a "breadth-first" fashion. The first few steps are shown below.

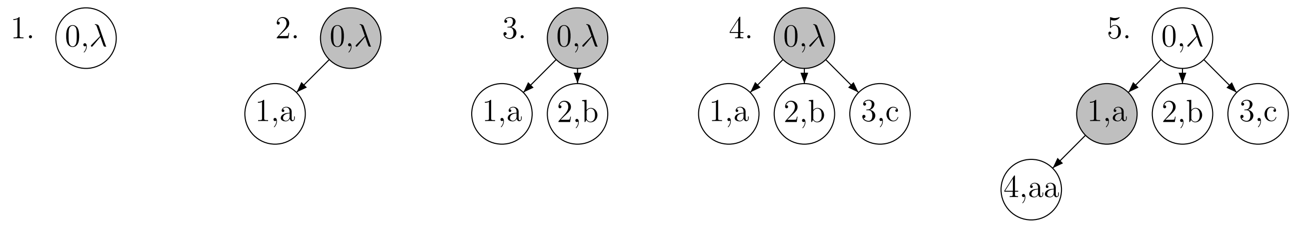


Figure 1.A.2: Enumerating Σ^* with $\Sigma = \{a, b, c\}$.

The procedure for Σ could be stated as follows. Remember we know there are k elements in Σ, and we can assume they are ordered. We are given as input a number n and we want to output the nth string in the enumeration of Σ^*.

1. Set a counter variable c to 0.
2. BUILD a node labeled $(0, \lambda)$.
3. If $0 = n$ then OUTPUT λ and STOP.
4. Otherwise, ADD $(0, \lambda)$ to the QUEUE.
5. REMOVE the first element (m, w) from the QUEUE.

6. Set variable i to 1.
7. Let a be the ith symbol in Σ.
8. Increase c by 1.
9. BUILD a node labeled $(c, w \cdot a)$ as a daughter to (m, w).
10. If $c = n$ then OUTPUT $w \cdot a$ and STOP.
11. Otherwise, ADD this daughter node to the end of the QUEUE.
12. Increase i by 1.
13. If $i > k$ then go to step 5. Otherwise, go to step 7.

The general form of this algorithm is very useful. Recall that an enumeration of Σ^* is also an enumeration of all programs! This means we could try running some set of inputs X on all the programs to find a program that gives a certain output. Basically, in steps 3 and 10 we would check to see how the program w behaves on the inputs in X. If the behavior is what we like, we output this program and stop. Otherwise we continue to the next program!

1.B Non-enumerable stringsets

This is where Alëna Aksenova's handout comes in.

Chapter 2

Identification in the Limit from Positive Data

A definition of a learning problem requires specifying the instances of the problem and specifying what counts as correct answers for these instances. This means thinking carefully about an interaction between three items: the learning targets, the learning algorithm, and the input to the learning algorithm, which can be thought of as the available evidence.

This is difficult because we have to confront the question "Which inputs is it reasonable to expect the learning algorithm to succeed on?" For example, if we are trying to identify a stringset S which is of infinite size but the evidence for S contains only a single string $s \in S$ then we may feel this places an unreasonable burden on the learning algorithm. What is at stake here was expressed by Charles Babbage:

> On two occasions I have been asked [by members of Parliament], "Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?" I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question. *as quoted in de la Higuera (2010, p. 391)*

It's unfair to expect a summation algorithm to succeed if the input is wrong. More generally, how do we define learning in such a way so that the input to the algorithm is not "wrong". What does it mean to have input of sufficient quality in learning? We want to only consider instances of the learning problem that are reasonable or fair. But nailing that down precisely is hard! In fact, what we will see is that this is an ongoing issue and there are many attempts to address it. The issue is a live one today.

2.1 Identification in the limit

Gold (1967) provided some influential definitions of learning. He called his approach **identification in the limit**. He provided not one, but several definitions, and he compared what kinds of stringsets were learnable in these paradigms.

No one I know knows what happened to Gold. He seems to have disappeared from academia in the 1980s.

Gold conceptualized learning as a never-ending process unfolding in time. Evidence is presented piece by piece in time to the learning algorithm. The learning algorithm outputs a program with each piece of evidence it receives based on its experience up to the present moment. As time goes on, the programs the learning algorithm outputs must be identical and must solve the membership problem for the target stringset.

Time t	1	2	3	4	...	n	...
Evidence at time t	e(1)	e(2)	e(3)	e(4)	...	e(n)	...
Input to Algorithm at time t	e⟨1⟩	e⟨2⟩	e⟨3⟩	e⟨4⟩	...	e⟨n⟩	...
Output of Algorithm at time t	G(1)	G(2)	G(3)	G(4)	...	G(n)	...

Figure 2.1: A schema of the Identification in the Limit learning paradigm

Let us explain the notation in the figure. The notation "e(n)" means the evidence presented at time n. This notation is functional which means evidence can be understood as a function with domain $\mathbb{N}$.

The notation "e⟨n⟩" refers to the sequence of evidence up to the nth one. For example, e(3) means the finite sequence "e(1), e(2), e(3)." In mathematics, angle brackets are sometimes used to denote sequences so some mathematicians would write this sequence as $\langle\, \mathrm{e}(1), \mathrm{e}(2), \mathrm{e}(3)\, \rangle$.

The notation "G(n)" refers to the program output by the algorithm with input e⟨n⟩. If A is the algorithm, and we wish to use functional notation so that $A(i) = o$ means "on input i, algorithm A outputs o" then G(n)=$A(e\langle n\rangle)$.

There are two important ideas in this paradigm. First, a successful learning algorithm is one that converges over time to a correct generalization. At some time point n, the algorithm must output the same program and this program must solve the membership problem for S. This means the algorithm can make mistakes, *but only finitely many times.*

Second, which infinite sequences of evidence learners must succeed on? Which are the ones of sufficient quality? Gold defined required these sequences to be representative of the target stringsets. *Each* possible piece of evidence *occurs at some point* in the unfolding sequence of evidence. Lest we think this is too good to be true, recall that the input to the learner at any given point n in time is the *finite* sequence e⟨n⟩, and that to succeed, it is only allowed to make finitely many mistakes.

2.1.1 Identification in the limit from positive data

The box below precisely defines the paradigm when learning from *positive* data. Let us define the "evidence" when learning from positive data more precisely. A **positive presentation** of a stringset S is a function $\varphi : \mathbb{N} \to S$ such that φ is onto. Recall that a function f is onto provided for every element y in its co-domain there is some element x in its domain such that $f(x) = y$. Here, this means for every string $s \in S$, there is some $n \in \mathbb{N}$ such that $\varphi(n) = s$.

Definition 1 (Identification in the limit from positive data)**.**

```
1 Algorithm $A$ *identifies in the limit from positive data* a class of stringsets $C$ provided
2     for all stringsets $S \in C$,
3         for all positive presentations $\varphi$ of $S$,
4             there is some number $n \in \mathbb{N}$ such that
5                 for all $m > n$,
6                 • the program output by $A$ on $\varphi\langle m\rangle$ is the same as the the program
7                   output by $A$ on $\varphi\langle n\rangle$, and
8                 • the program output by $A$ on $\varphi\langle m\rangle$ solves the membership problem
9                   for $S$.
```

Here is breakdown of what these lines mean.

Line 1 Establishes the name of the relationship between an algorithm A and a collection of stringsets C provided the definition holds.

Line 2 The algorithm must succeed for all $S \in C$.

Line 3 The algorithm must succeed for all positive presentations φ of S.

Line 4 It succeeds on φ for S if there is a point in time n

Line 5 such that for all future points in time m,

Lines 6-7 the output of A converges to the same program, and

Lines 8-9 the output of A correctly solves the membership problem for S.

This paradigm is also called **learning from text.**

2.1.2 The Strictly k-Piecewise Stringsets

Example 3. Here we present an algorithm and prove that it identifies the Strictly k-Piecewise (SP$_k$) stringsets in the limit from positive data. SP stringsets were proposed to model aspects of long-distance phonotactics Heinz (2010a), motivated on typological and learnability grounds. The learning scheme discussed here exemplifies more general ideas Heinz (2010b); Heinz *et al.* (2012).

The notion of *subsequence* is integral to SP stringsets. Informally, a string u is subsequence of string v if one is left with u after erasing zero or more letters in v. For example, *ab* is a subsequence of *cccccaccccccccbcccccc*. Formally, u is a subsequence of v ($u \sqsubseteq v$)

provided there are strings $x_1, x_2, \dots x_n$ and strings $y_0, y_1, \dots y_n$ such that $u = x_1x_2\dots x_n$ and $v = y_0x_1y_1x_2y_2\dots x_ny_n$. It is the y_i strings that erased in v to leave u.

A stringset S is Strictly Piecewise if and only if it is closed under subsequence. In other words, if $s \in S$ then every subsequence of s is also in S.

A theorem shows that every SP stringset S has a basis in a finite set of strings (Rogers *et al.*, 2010). These strings can be understood as *forbidden* subsequences. That is any string $s \in \Sigma^*$ containing any one of the forbidden subsequences is not in S. Conversely, any string s which $\underline{\text{does not}}$ contain $\underline{\text{any}}$ forbidden subsequence belongs to S.

The same theorem shows that a SP stringset S can be defined in terms of a finite set of *permissible* subsequences. Because the set is finite, there is a longest string in this set. Let its length be k. In this case, any $s \in \Sigma^*$ belongs to S if and only if $\underline{\text{every}}$ one of its subsequences of length k or less is permissible.

In other words we can define SP$_k$ stringsets as follows. Let a grammar G be a finite subset of Σ^* and let k be the length of a longest string in G. Let $\texttt{subseq}_k(s) = \{u \mid u \sqsubseteq s, |u| \leq k\}$. The "language of the grammar" $L(G)$ is defined as the stringset $\{s \mid \texttt{subseq}_k(s) \subseteq G\}$. We are going to be interested in the collection of stringsets SP$_k$, defined as those stringsets generated from grammars G with a longest string k. Formally,

$$\text{SP}_k \overset{\text{def}}{=} \{S \mid G \subseteq \Sigma^{\leq k}, L(G) = S\} \ .$$

This is the collection C of learning targets.

For all $S \in \text{SP}_k$, all presentations φ of S, and all time points $t \in \mathbb{N}$ define A as follows:

$$A\big(\varphi\langle t\rangle\big) = \begin{cases} \texttt{subseq}_k(\varphi(t)) & \text{if } t = 1 \\ A(\varphi\langle t-1\rangle) \cup \texttt{subseq}_k(\varphi(t)) & \text{otherwise} \end{cases}$$

One can prove that algorithm A identifies in the limit from positive data the collection of stringsets SP$_k$.

Exercise 7. Prove algorithm A identifies in the limit from positive data the collection of stringsets SP$_k$.

For any presentation ϕ and time t, define k-SPIA (Strictly k-Piecewise Inference Algorithm) as follows

$$k\text{-SPIA}\big(\varphi\langle t\rangle\big) = \begin{cases} \varnothing & \text{if } t = 0 \\ k\text{-SPIA}(\varphi\langle t-1\rangle) \cup \texttt{subseq}_k(\varphi(t)) & \text{otherwise} \end{cases}$$

Note that we are being a little sloppy here. Technically, the output of k-SPIA given some input sequence is a set of subsequences G, not a program. What we really mean with the above is that $k-$SPIA outputs a program which uses G to solve the membership problem for $L(G) = \{w \mid \texttt{subseq}_k(w) \subseteq G\}$. This program looks something like this.

1. Input: any word w.
2. Check whether $\texttt{subseq}_k(w) \subseteq G$.

3. If so, OUTPUT Yes, otherwise OUTPUT No.

All $k-$SPIA does is update this program simply by updating the contents of G.

Theorem 1. *For each k, $k-$SPIA identifies in the limit from positive data the collection of stringsets SP_k.*

Proof Consider any $k \in \mathbb{N}$. Consider any $S \in \text{SP}_k$. Consider any positive presentation φ for S. It is sufficient to show there exists a point in time t_ℓ such that for all $m \geq t_\ell$ the following holds:

1. k-SPIA$(\langle m \rangle) = k-$SPIA$(\langle t_\ell \rangle)$ (convergence), and
2. k-SPIA$(\langle m \rangle)$ is a program that solves the membership probem for S.

Since $S \in \text{SP}_k$, there is a finite set $G \subseteq \Sigma^{\leq k}$ such that $S = L(G)$.

Consider any subsequence $g \in G$. Since $g \in G$ there is some word $w \in S$ which contains g as a k-subsequence. Since G is finite, there are finitely many such w, one for each g in G. Because φ is a positive presentation for S, there is a time t where each of these w occurs. For each w let t be the first occurence of w in φ. Let t_ℓ denote the latest time point of all of these time points t. Next we argue that for all time points m larger than this t_ℓ, the output of $k-$SPIA correctly solves the membership problem for S and does not change.

Consider any $m \geq t_\ell$. The claim is that k-SPIA$(\langle m \rangle) = k-$SPIA$(\langle t_\ell \rangle) = G$. For each g in G, a word containing g as a subsequence occurs at or earlier than t_ℓ and so $g \in k-$SPIA$(\langle m \rangle)$. Since g was arbitrary in G, $G \subseteq k-$SPIA$(\langle m \rangle)$.

Similarly, for each $g \in k-$SPIA$(\langle m \rangle)$, there was some word w in φ such that w contains g as a subsequence. Since φ is a positive presentation for S, w is in S. Since w belongs to S, $\texttt{subseq}_k(w) \subseteq G$ and so g belongs to G. Since g was arbitrary in k-SPIA$(\langle m \rangle)$ it follows that k-SPIA$(\langle m \rangle) \subseteq G$.

It follows k-SPIA$(\langle m \rangle) = G$.

Since m was arbitrarily larger than t_ℓ we have both convergence and correctness.

Since φ was arbitrary for S, S arbitrary in SP_k and k arbitrary, the proof is concluded. □

2.1.3 The Strictly k-Local Stringsets

Here we present an algorithm and prove that it identifies the Strictly k-Local (SL_k) stringsets in the limit from positive data. The first proof of this result was presented by Garcia *et al.* (1990), though the Markovian principles underlying this result were understood in a statistical context much earlier. The learning scheme discussed there exemplifies more general ideas (Heinz, 2010b; Heinz *et al.*, 2012).

The notion of *substring* is integral to SL stringsets. Formally, a string u is substring of string v ($u \trianglelefteq v$) provided there are strings $x, y \in \Sigma^*$ and $v = xuy$. Another term for substring is *factor*. So we also say that u is a factor of v. If u is of length k then we say u is a k-factor of v.

A stringset S is Strictly k-Local if and only if there is a number k such that for all strings $u_1, v_1, u_2, v_2, x \in \Sigma^*$ such that if $|x| = k$ and $u_1xv_1, u_2xv_2 \in S$ then $u_1xv_2 \in S$. We say S is closed under suffix substitution (Rogers and Pullum, 2011).

A theorem shows that every SL_k stringset S has a basis in a finite set of strings (Rogers and Pullum, 2011). These strings can be understood as *forbidden* substrings. Informally, this means any string s containing any one of the forbidden substrings is not in S. Conversely, any string s which $\underline{\text{does not}}$ contain $\underline{\text{any}}$ forbidden substring belongs to S.

The same theorem shows that a SL stringset S can be defined in terms of a finite set of *permissible* substrings. In this case, s belongs to S if and only if $\underline{\text{every}}$ one of its k-factors is permissible.

We formalize the above notions by first defining a function the $\texttt{factor}_k$, which extracts the substrings of length k present in a string, or those present in a set of strings. If a string s is of length less than k then $\texttt{factor}_k$ just returns s.

Formally, let $\texttt{factor}_k(s)$ equal $\{u \mid u \trianglelefteq s, |u| = k\}$ whenever $k \leq |s|$ and let $\texttt{factor}_k(s) = \{s\}$ whenever $|s| < k$. We expand the domain of this function to include sets of strings as follows: $\texttt{factor}_k(S) = \bigcup_{s\in S} \texttt{factor}_k(s)$.

To formally define SL_k grammars, we introduce the symbols $\rtimes$ and $\ltimes$, which denote left and right word boundaries, respectively. These symbols are introduced because we also want to be able to forbid specific strings at the beginning and ends of words, and traditionally strictly local stringsets were defined to make such distinctions (McNaughton and Papert, 1971). Then let a grammar G be a finite subset of $\texttt{factor}_k(\{\rtimes\}\Sigma^*\{\ltimes\})$.

The "language of the grammar" $L(G)$ is defined as the stringset $\{s \mid \texttt{factor}_k(\rtimes s \ltimes) \subseteq G\}$. We are going to be interested in the collection of stringsets SL_k, defined as those stringsets generated from grammars G with a longest string k. Formally,

$$\text{SL}_k \overset{\text{def}}{=} \{S \mid G \subseteq \texttt{factor}_k(\{\rtimes\}\Sigma^*\{\ltimes\}), L(G) = S\} \ .$$

This is the collection C of learning targets.

For all $S \in \text{SL}_k$, for any presentation ϕ and time t, define k-SPIA (Strictly k-Local Inference Algorithm) as follows

$$k\text{-SLIA}\big(\varphi\langle t\rangle\big) = \begin{cases} \varnothing & \text{if } t = 0 \\ k\text{-SLIA}(\varphi\langle t-1\rangle) \cup \texttt{factor}_k(\rtimes\varphi(t)\ltimes) & \text{otherwise} \end{cases}$$

Exercise 8. Prove algorithm k-SLIA identifies in the limit from positive data the collection of stringsets SL_k.

Note that we are being a little sloppy here. Technically, the output of k-SLIA given some input sequence is a set of subsequences G, not a program. What we really mean with the above is that $k-$SLIA outputs a program which uses G to solve the membership problem for $L(G) = \{w \mid \texttt{subseq}_k(w) \subseteq G\}$. This program looks something like this.

1. Input: any word w.
2. Check whether $\texttt{factor}_k(\rtimes w \ltimes) \subseteq G$.

3. If so, OUTPUT Yes, otherwise OUTPUT No.

All $k-$SLIA does is update this program simply by updating the contents of G.

Theorem 2. *For each k, $k-$SLIA identifies in the limit from positive data the collection of stringsets SL_k.*

Proof Consider any $k \in \mathbb{N}$. Consider any $S \in \mathrm{SL}_k$. Consider any positive presentation φ for S. It is sufficient to show there exists a point in time t_ℓ such that for all $m \geq t_\ell$ the following holds:

1. k-SLIA$(\langle m\rangle) = k-$SLIA$(\langle t_\ell\rangle)$ (convergence), and
2. k-SLIA$(\langle m\rangle)$ is a program that solves the membership probem for S.

Since $S \in \mathrm{SL}_k$, there is a finite set $G \subseteq \Sigma^{\leq k}$ such that $S = L(G)$.

Consider any factor $g \in G$. Since $g \in G$ there is some word $w \in S$ which contains g as a k-factor. Since G is finite, there are finitely many such w, one for each g in G. Because φ is a positive presentation for S, there is a time t where each of these w occurs. For each w let t be the first occurence of w in φ. Let t_ℓ denote the latest time point of all of these time points t. Next we argue that for all time points m larger than this t_ℓ, the output of $k-$SLIA correctly solves the membership problem for S and does not change.

Consider any $m \geq t_\ell$. The claim is that k-SLIA$(\langle m\rangle) = k-$SLIA$(\langle t_\ell\rangle) = G$. For each g in G, a word containing g as a factor occurs at or earlier than t_ℓ and so $g \in k-$SLIA$(\langle m\rangle)$. Since g was arbitrary in G, $G \subseteq k-$SLIA$(\langle m\rangle)$.

Similarly, for each $g \in k-$SLIA$(\langle m\rangle)$, there was some word w in φ such that w contains g as a factor. Since φ is a positive presentation for S, w is in S. Since w belongs to S, $\mathtt{factor}_k(w) \subseteq G$ and so g belongs to G. Since g was arbitrary in k-SLIA$(\langle m\rangle)$ it follows that k-SLIA$(\langle m\rangle) \subseteq G$.

It follows k-SLIA$(\langle m\rangle) = G$.

Since m was arbitrarily larger than t_ℓ we have both convergence and correctness.

Since φ was arbitrary for S, S arbitrary in SL_k and k arbitrary, the proof is concluded. □

2.1.4 Strictly k-Local Treesets

Trees are like strings in that they are recursive structures. Informally, trees are structures with a single 'root' which dominates a sequence of trees.

Defining Trees and Treesets

Formally, trees extend the dimensionality of string structures from 1 to 2 (Rogers, 2003). Like strings, we assume a set of symbols Σ. This is often partioned into symbols of different types depending on whether the symbols can only occur at the leaves of the trees or not. We don't make any such distinction here.

Definition 2 (Trees)**.**

Base Case: If $a \in \Sigma$ then $a[\ ]$ is a tree.

Inductive Case: If $a \in \Sigma$ and $t_1, t_2, \dots t_n$ is a string of trees of length n then $a[t_1 t_2 \dots t_n]$ is a tree.

Also, a tree $a[]$ is called a *leaf.* We denote set of all possible trees with $\mathbb{T}^2_\Sigma$. A *treeset* T is a subset of $\mathbb{T}^2_\Sigma$.

(This notation follows Rogers (2003) wherein $\mathbb{T}^d_\Sigma$ denotes tree-like structures with Σ being the set of labels and d being the dimensionality. Since strings are of dimension 1, this means the set of all strings Σ^* is equivalent to $\mathbb{T}^1_\Sigma$.)

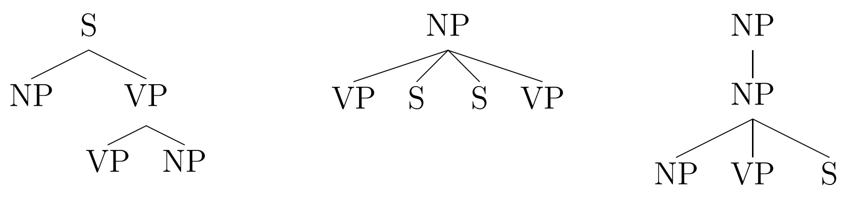
Here are some examples.

Example 4. Let $\Sigma = \{$NP, VP, S$\}$. Then the following are trees.

1. S[NP[ ] VP[VP[ ] NP[ ]]]]
2. NP[VP[ ] S[ ] S[ ] VP[ ]]
3. NP[NP[NP[ ] VP[ ] S[ ]]]

We might draw these structures as follows.

(1) (2) (3)



Note that the expression "a string of trees" in the definition of trees implies that our alphabet for strings is all the trees. Since we are defining trees, this may seem a bit circular. The key to resolving this circularity is to interleave the definition of the alphabet of the strings with the definition of trees in a zig-zag fashion. First we apply the inductive case for trees *once*, then we use those trees as an alphabet to define some strings of trees. Then we go back to trees and apply the inductive case again, which yields more trees which we can use to enlarge our set of strings and so on. While we do not go through the details here, this method essentially provides a way to enumerate the set of all possible trees.

Here are some useful definitions which give us information about trees.

Definition 3.

1. The *root* of a tree $a[t_1 \dots t_n]$ is a.

2. The *size* of a tree t, written $|t|$, is defined as follows. If $t = a[\ ]$, its size is 1. If not, then $t = a[t_1 t_2 \dots t_n]$ where each t_i is a tree. Then $|t| = 1 + |t_1| + |t_2| + \dots + |t_n|$.

3. The *depth* of a tree t, written $\texttt{depth}(t)$, is defined as follows. If $t = a[\ ]$, its depth is 1. If not, then $t = a[t_1 t_2 \dots t_n]$ where each t_i is a tree. Then $\texttt{depth}(t) = 1 + \texttt{max}\big\{\texttt{depth}(t_1), \texttt{depth}(t_2), \dots, \texttt{depth}(t_n)\big\}$ where $\texttt{max}$ takes the largest number in the set.

4. The *yield* of a tree t, written $\texttt{yield}(t)$, maps a tree to a string of its leaves as follows. If $t = a[\ ]$ then $\texttt{yield}(t) = a$. If not, then $t = a[t_1 t_2 \dots t_n]$ where each t_i is a tree. Then $\texttt{yield}(t) = \texttt{yield}(t_1) \cdot \texttt{yield}(t_2) \cdot \dots \cdot \texttt{yield}(t_n)$.

5. Tree t is a *subtree* of $t' = a[t_1 \dots t_n]$ provided there is i such that either $t = t_i$ or t is a subtree of t_i.

6. A tree $t = a[a_1[\ ] \dots a_n[\ ]\ ]$ is a *2-local tree* of tree t' ($t \trianglelefteq t'$) provided there exists a subtree s of t' such that $s = a[t_1 \dots t_n]$ and the root of each t_i is a_i.

7. A 1-*treetop* of $t = a[t_1 t_2 \dots t_n]$ is $a[\ ]$.

8. A k-*treetop* of $t = a[t_1 t_2 \dots t_n]$ is $a[s_1 s_2 \dots s_n]$ where each s_i is a $(k-1)$-treetop of t_i.

9. A tree $t = a[t_1 \dots t_n]$ is a k-*local tree* of tree t' ($t \trianglelefteq t'$) provided
   (a) t is of depth k
   (b) there exists a subtree s of t' such that $s = a[s_1 \dots s_n]$ and for each i, t_i is the $(k-1)$ treetop of s_i.

Strictly Local Treesets

The notion of *k-local tree* ($\trianglelefteq$) is integral to Strictly Local treesets.

As with SL stringsets, we define a function $\texttt{factor}_k$, which extracts the k-local trees present in a tree (or set of trees). If a tree t is of depth less than k then $\texttt{factor}_k$ just returns $\{t\}$.

Formally, let $\texttt{factor}_k(t)$ equal $\{s \mid s \trianglelefteq t, \texttt{depth}(s) = k\}$ whenever $k \leq \texttt{depth}(t)$ and let $\texttt{factor}_k(t) = \{t\}$ whenever $\texttt{depth}(t) < k$. We expand the domain of this function to include sets of strings as follows: $\texttt{factor}_k(T) = \bigcup_{t \in T} \texttt{factor}_k(t)$.

Then a SL$_k$ treeset grammar $G = (\Sigma_0, \Sigma_\ell, F_k)$, which will be interpreted as the symbols permissible as the roots, the symbols permissible as the leaves, and the permissible k-Local trees. So $\Sigma_0, \Sigma_\ell \subseteq \Sigma$ and F_k is a finite subset of $\texttt{factor}_k(\mathbb{T}^2_\Sigma)$.

The "language of the grammar" $L\big((\Sigma_0, \Sigma_\ell, F_k)\big)$ is defined as the treeset

$$\{t \mid \texttt{root}(t) \in \Sigma_0, \texttt{yield}(t) \in \Sigma_\ell^*, \texttt{factor}_k(t) \subseteq F_k, \}\ .$$

We are going to be interested in the collection of treesets SLT$_k$, defined as those treesets generated from grammars G where the depth of the largest permissible local tree is k. Formally,

$$\text{SLT}_k \overset{\text{def}}{=} \{T \mid \exists G \subseteq \Sigma \times \Sigma \times \texttt{factor}_k(\mathbb{T}^2_\Sigma), L(G) = T\}\ .$$

This is the collection C of learning targets.

Theorem 3. *For each k, the class k-SLT is identifiable in the limit from positive data.*

To make this result clear, let us remind ourselves what a positive presentation of data is. It means for each $t \in T$ there is some time point i such that $\phi(i) = t$. So the evidence here are tree structures, not the yields of tree structures.

Exercise 9. Prove the theorem.

Context-Free Grammars

Definition 4. A *context-free grammar* is a tuple $\langle T, N, S, \mathcal{R} \rangle$ where

- $\mathcal{T}$ is a nonempty finite alphabet of symbols. These symbols are also called the *terminal* symbols, and we usually write them with lowercase letters like $a, b, c, \ldots$
- $\mathcal{N}$ is a nonempty finite set of *non-terminal* symbols, which are distinct from elements of $\mathcal{T}$. These symbols are also called *category* symbols, and we usually write them with uppercase letters like $A, B, C, \ldots$
- S is the *start* category, which is an element of $\mathcal{N}$.
- A finite set of *production rules* $\mathcal{R}$. A production rule has the form

$$A \rightarrow \beta$$

  where β belongs to $(\mathcal{T} \cup \mathcal{N})^*$ and $A \in \mathcal{N}$. So β are strings of non-terminal and terminal symbols and A is a non-terminal.

Example 5. Consider the following grammar G_1:

- $\mathcal{T} = \{$***john***, ***laughed***, ***and***$\}$;
- $\mathcal{N} = \{$S, VP1, VP2$\}$; and
- $$\mathcal{R} = \left\{ \begin{array}{l} \text{S} \rightarrow \textbf{\textit{john}} \text{ VP1} \\ \text{VP1} \rightarrow \textbf{\textit{laughed}} \\ \text{VP1} \rightarrow \textbf{\textit{laughed}} \text{ VP2} \\ \text{VP2} \rightarrow \textbf{\textit{and laughed}} \text{ VP2} \\ \text{VP2} \rightarrow \textbf{\textit{laughed}} \end{array} \right\}$$

Example 6. Consider the following grammar G_2:

- $T = \{$a, b$\}$;
- $V = \{$S$\}$; and

- The production rules are

$$\mathcal{R} = \left\{ \begin{array}{l} \text{S} \rightarrow \text{aSb} \\ \text{S} \rightarrow ab \end{array} \right\}$$

The language of a context-free grammar (CFG) is defined recursively below.

Definition 5. The (partial) *derivations* of a CFG $G = \langle \mathcal{T}, \mathcal{N}, S, \mathcal{R} \rangle$ is written $D(G)$ and is defined recursively as follows.

1. *The base case:* S belongs to $D(G)$.
2. *The recursive case:* For all $A \rightarrow \beta \in \mathcal{R}$ and for all $\gamma_1, \gamma_2 \in (\mathcal{T} \cup \mathcal{N})^*$, if $\gamma_1 A \gamma_2 \in D(G)$ then $\gamma_1 \beta \gamma_2 \in D(G)$.
3. Nothing else is in $D(G)$.

Then the language of the grammar $L(G)$ is defined as

$$L(G) = \big\{ w \in \mathcal{T}^* \mid w \in D(G) \big\}.$$

Exercise 10. How does G_1 generate ***John laughed and laughed and laughed***?

Exercise 11. What language does G_2 generate?

Theorem 4. *The languages generated by context-free grammars are exactly the yields of the Strictly 2-Local treesets.*

Exercise 12. Explain how the 2-local trees in a tree relate to the production rules of a context-free grammar.

Chapter 3

Identification in the Limit: General Results

3.1 Identification in the limit from positive and negative data

A **positive and negative presentation** of a stringset S provides example strings not in S in addition to example strings in S. This can be formalized using the *characteristic function* of S. Every set S has a characteristic function with domain Σ^* defined as follows.

$$f_S(s) = \begin{cases} 1 & \text{iff } s \in S \\ 0 & \text{otherwise} \end{cases}$$

Characteristic functions are total functions, which means defined for all $s \in \Sigma^*$. Also recall, that we write $(x, y) \in f$ whenever $f(x) = y$. So we can think of f_S as a set of points where $(s, 0)$ means $s \notin S$ and $(s, 1)$ means $s \in S$.

Then a **positive and negative presentation** of a stringset S is a function $\varphi : \mathbb{N} \to f_S$ such that φ is onto. Here, this means for every string $s \in \Sigma^*$, there is some $n \in \mathbb{N}$ such that $\varphi(n) = (s, f_s(s))$.

Definition 6 (Identification in the limit from positive and negative data)**.**

```
Algorithm $A$ *identifies in the limit from positive and negative data* a class of stringsets
$C$ provided
    for all stringsets $S \in C$,
        for all positive and negative presentations $\varphi$ of $S$,
            there is some number $n \in \mathbb{N}$ such that
                for all $m > n$,
                • the program output by $A$ on $\varphi\langle m \rangle$ is the same as the the program
                  output by $A$ on $\varphi\langle n \rangle$, and
                • the program output by $A$ on $\varphi\langle m \rangle$ solves the membership problem
                  for $S$.
```

The only difference between the definition above and the one in Definition 1 is in line 3. This paradigm is also called **learning from an informant.**

3.2 Variations on a theme

Exercise 13. Suppose we want to learn a transformation from strings to strings? In other words a relation from Σ^* to Δ^*? How could the definitions above be changed?

Exercise 14. Suppose we want to learn a probability distribution over Σ^*. How could the definitions above be changed?

3.3 Important Results

1. Finite class of stringsets is identifiable in the limit from positive data
2. No superfinite class of stringsets is identifiable in the limit from positive data
3. The computably enumerable stringsets are identifiable in the limit from positive and negative data
    (a) This learner by enumeration is not efficient
4. The regular class of stringsets is efficiently identifiable in the limit from positive and negative data (RPNI)
5. Deterministic transductions are identifiable in the limit from positive data (OSTIA)
6. Determinsitic probability distributions are identifiable in the limit from positive data (RLIPS, ALEGRIA)

7. Corresponding extensions to regular tree languages and tree transductions
8. Gold's conclusions and critical analysis/reflection

Chapter 4

Generalizing Strictly Local Learning

1. SL distributions (n-gram models)
   (a) Different definition of learning (MLE)
2. ISL and OSL string-to-string functions
   (a) Adding variation to this?
   (b) Adding probabilities to this?
3. The proper notion of k-factor
   (a) subsequences are k-factors with the right reps
   (b) SL tree languages are k-factors with the right reps
   (c) k-factors of autosegmental structures?
   (d) Feature-based representations?
4. ISL and OSL tree transductions?

Bibliography

Chomsky, Noam. 1956. Three models for the description of language. *IRE Transactions on Information Theory* 113124. IT-2.

Garcia, Pedro, Enrique Vidal, and José Oncina. 1990. Learning locally testable languages in the strict sense. In *Proceedings of the Workshop on Algorithmic Learning Theory*, 325–338.

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

Heinz, Jeffrey. 2010a. Learning long-distance phonotactics. *Linguistic Inquiry* 41:623–661.

Heinz, Jeffrey. 2010b. String extension learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, 897–906. Uppsala, Sweden: Association for Computational Linguistics.

Heinz, Jeffrey, Anna Kasprzik, and Timo Kötzing. 2012. Learning with lattice-structured hypothesis spaces. *Theoretical Computer Science* 457:111–127.

de la Higuera, Colin. 2010. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press.

Hopcroft, John E., and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

McNaughton, Robert, and Seymour Papert. 1971. *Counter-Free Automata*. MIT Press.

Osherson, Daniel, Scott Weinstein, and Michael Stob. 1986. *Systems that Learn*. Cambridge, MA: MIT Press.

Rogers, James. 2003. wMSO theories as grammar formalisms. *Theoretical Computer Science* 293:291–320.

Rogers, James, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Visscher, David Wellcome, and Sean Wibel. 2010. On languages piecewise testable in the strict sense. In *The Mathematics of Language*, edited by Christian Ebert, Gerhard Jäger, and Jens Michaelis, vol. 6149 of *Lecture Notes in Artifical Intelligence*, 255–265. Springer.

Rogers, James, Jeffrey Heinz, Margaret Fero, Jeremy Hurst, Dakotah Lambert, and Sean Wibel. 2013. Cognitive and sub-regular complexity. In *Formal Grammar*, edited by Glyn Morrill and Mark-Jan Nederhof, vol. 8036 of *Lecture Notes in Computer Science*, 90–108. Springer.

Rogers, James, and Geoffrey Pullum. 2011. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information* 20:329–342.

Thomas, Wolfgang. 1982. Classifying regular events in symbolic logic. *Journal of Computer and Systems Sciences* 25:370–376.