Computational Linguistics:

Language Learning

Edward P. Stabler
Lx212 notes, draft 2012-12-06 19:11

Contents

Chapter 0 Introduction

(0) **Some traditional questions.**

- Is language acquisition more like learning from evidence or more like growth or triggering?
- Are specific properties of human syntax innately given at the outset, in every human learner?

And linguists are particularly interested in the questions,

- How do properties of the language learning shape the languages we speak?
- How can linguistic ability be acquired as rapidly and reliably as it is, from readily available data?

Questions like these generate heated disputes, but they are typically posed in these vague terms that obscure the real issues. We will introduce some ways of thinking about learning that will allow much clearer and more interesting questions to be formulated and (some of them) answered.

(1) In the past 10 years or so, a remarkable consensus emerged about appropriate measures of learning complexity, measures related to how much evidence might be required to distinguish one hypothesis from alternatives. "VC dimension" and a number of other, independently proposed measures turn out to be equivalent, showing that a number of independent traditions have homed in on essentially the same idea about learning. Having a criterion like this is important because it allows us to to define the kind of learning process we should be searching for. (Blumer et al., 1989; Mukherjee et al., 2004)

(2) With this measure, it is easy to establish the learnability of finitely parameterized grammars, OT grammars and harmonic grammars that assume a given, finite list of constraints. But these results are unsatisfying if you doubt that the number of parameters needed to define human languages (with their lexicons included) is small enough to be treated as a finite, given list. And in certain cases it can be difficult or impossible to apply these methods directly to unanalyzed data of the sort plausibly available to the human learner. (Chomsky, 1981; Pinker, 1982; Heinz, Kobele, and Riggle, 2009; Bane, Riggle, and Sonderegger, 2010)

(3) Recent results establish the learnability of certain aspects of linguistic structure from data more likely to be perceptually available, without assuming a given, finite parameterization (Angluin, 1982; Clark and Thollard, 2004). In phonology, it appears that at least some of the patterns may be in "sub-regular" classes that allow efficient learning (Frank and Satta, 1998; Jäger, 2002; Heinz, 2010), and

(4) some of these ideas for learning regular languages have been extended to the more complex patterns in syntax. In particular, we now have learners for significant classes of context-free (Clark and Eyraud, 2007) and non-context-free languages definable by simple, formal 'minimalist grammars' (Yoshinaka and Clark, 2010; Yoshinaka, 2010).

0.0 Method

(5) Chomsky'65 *Aspects of the Theory of Syntax* (Chomsky, 1965)

- Identify factors underlying linguistic behavior...

(p5) The study actual linguistic performance, we must consider the interaction of a variety of factors, of which the underlying competence of the speaker is only one. In this respect, study of language is no different from empirical investigation of other complex phenomena.

- universal regularities hide in irregular details

(p6) [Although] traditional and structuralist grammars... may contain full and explicit lists of exceptions and irregularities, they provide only examples and hints concerning the regular and productive syntactic processes ... [It] is to be supplemented by a universal grammar that ... expresses the deep-seated regularities which, being universal, are omitted from the grammar itself.

(p6) . . . The problem for the linguist, as well as for the child learning the language, is to determine from the data of performance the underlying system of rules that has been mastered by the speaker-hearer and that he puts to use in actual performance.

(1981,p10) In the general case of theory construction, the primitive basis can be selected in any number of ways,. . . But in the case of UG . . . we want the primitives to be concepts that can plausibly be assumed to provide a preliminary, prelinguistic analysis of a reasonable selection of presented data, that is, to provide the data that are mapped by the language faculty to a grammar. . . [Restricting] the class of grammars mad accessible in principle by UG. . . to account for the fact that knowledge of language is acquired on the basis of the evidence available.

- Linguistic structure not a mirror of thought

(p6) [A] reason for the failure of traditional grammars, particular or universal, to attempt a precise statement of regular processes of sentence formation and sentence interpretation lay in the widely held belief that there is a "natural order of thoughts" that is mirrored by the order of words. Hence, the rules of sentence formation do not really belong to grammar but to some other subject in which the "order of thoughts is studied."

- Grammar as some kind of recursive system

(p6) But the fundamental reason for this inadequacy of traditional grammars is a more technical one. . . the technical devices for expressing a system of recursive processes were simply not available until much more recently.

- 'Implementation' in performance model may be nontrivial

(p9) [A] generative grammar is not a model for a speaker or a hearer. It attempts to characterize in the most neutral possible terms the knowledge of the language that provides the basis for actual use of language by a speaker-hearer. . . [It] does not, in itself, prescribe the character or functioning of a perceptual model or a model of speech production.

- Identify factors underlying behavior
- Universal regularities hide in irregular details

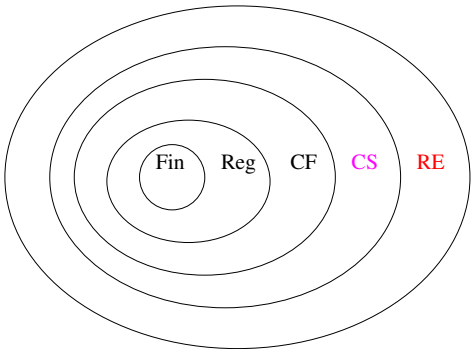
  (universals restricting the possible languages)
- Linguistic structure not a mirror of thought
- Grammar as some kind of recursive system
- 'Implementation' of grammar may be nontrivial

- Causal factors in language structure restricted
- The *kind* of recursive system is a language universal
- We want learners for that kind of recursive system

0.1 Kinds of recursive systems

0.1.1 The Chomsky hierarchy

(6) The classical presentation:



grammar rules

Fin: $S \to a_0 \dots a_n$

Reg: $A \to aB, A \to \epsilon$

CF: $A \to X$

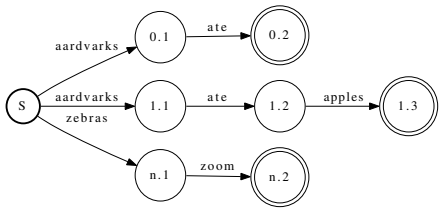
CS: $X \to Y$ with $|X| < |Y|$

RE: $X \to Y$

(7) **Fin:** $S \to a_0 \dots a_n$**, a list**

Example: S → aarvarks ate
S → aarvarks ate apples
S → zebras zoom

Each such grammar G corresponds to a 'list automaton'

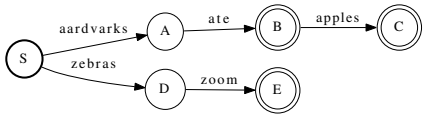


This is a simple kind of 'finite state automaton', FSA.

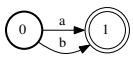
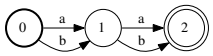
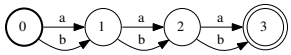
(8) **Fin: $A \to aB, A \to \epsilon$ without cycles**

- FSA state A is cyclic if, from A, A can be reached in ≥ 1 steps
- Category A is recursive iff, from A, ... A... derivable in ≥ 1 steps. (cf Aho&Ullman'72,p153)

Example: S → aardvarks A S → zebras D
A → ate B D → zoom E
B → apples C E → ϵ
B → ϵ
C → ϵ



(9) L(list fsa) = L(acyclic fsa) = Fin ('weakly' equivalent)
But the machines/grammars are different!

|sentence|	acyclic fsa	size	la size
1		2	3
2		3	9
3		4	25
n		n+1	$n2^n + 1$

Diffs in size of minimal, weak equivalents can be exponential.

(10) **Why English ∉ Fin**

- English has infinitely many sentences, since for any s ∈English, I can create a longer one.
  Worry: This arg only works for precise concepts like *prime number*, not for concepts with indefinite boundaries like *heap* or *bald person*
- '[T]he assumption that language is infinite is made for the purpose of simplifying the description' (Chomsky, 1956).

  Of course, *any* model of human cognition will make simplifications, and thus be inadequate in certain ways... (Fitch and Frederic 2012)
  Worry: Apparently contradicts first idea, raises questions about what 'simplifications' are allowed.

(11) **Why English ∉ Fin**: second try

(Chomsky'65,p3) Linguistic theory is concerned primarily with an ideal speaker-hearer . . . unaffected by such grammatically irrelevant conditions as memory limitations. . .

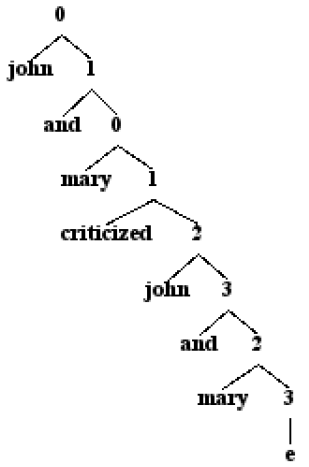
A two-factor account:

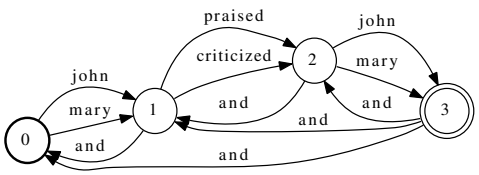
- English language recognizers have recursive, cyclic categories, so the language is infinite.
- English language recognizers have finite lifetimes, attention, but the factors influencing this are irrelevant to the factors determining linguistic structure.

Yes, our account of English is incomplete and incorrect in various ways, but these claims are not approximate, but true, and commonplace. Cf. claims about a pendulum, or a car engine, or a bodily organ.

0.1.2 Regular grammars and languages: recognizable with finite memory

(12) **Reg:** $A \to aB, A \to \epsilon$ (mem finite, but recursion allowed!)





size(A)=4 but |L(A)|=∞

$0 \to$ john 1
$0 \to$ mary 1
$1 \to$ and 0
$1 \to$ praised 1
$1 \to$ criticized 1
$2 \to$ and 1
$2 \to$ john 3
$2 \to$ mary 3
$3 \to \epsilon$
$3 \to$ and 2
$3 \to$ and 1
$3 \to$ and 0

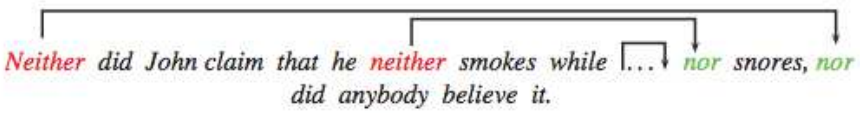
(13) **Eng ∉ Reg**: the 'supra-regular' hypothesis

(Fitch, Frederici, and Hagoort, 2012) Among linguists, psycholinguists and computer scientists today, the supra-regular hypothesis is nearly universally accepted. . . It has become a truism that natural language requires supra-regular resources, which are thus presumed to be present in some form in the human mind and implemented by human brains: this is not an issue debated in the recent literature. It is thus a peculiar historical fact that, until very recently, neither neuroscientists nor experimental and animal psychologists have shown any interest in this issue. (p1927)

Supra-regular distinctiveness hypothesis: . . . humans are unusual (or perhaps unique) in possessing supra-regular processing power. (p1929)

(14) **Why English ∉ Reg**

(Jäger and Rogers, 2012) The crucial insight here is that English has *center embedding* constructions. . .



As far as the grammar of English goes, there is no fixed upper bound on the number of levels of embedding. Consequently, English grammar allows for a potentially *unlimited number* of nested dependencies of *unlimited size*.

Isn't this shaky evidence? Why not assume a fuzzy boundary around 2 embeddings, with 3 marginal, but 10 out? – that would be representable with a regular grammar

(15) **Why AGL language $\notin$ Reg**: again

By testing the learner on examples of an artificial language (A^nB^n) with 2 embeddings, can we tell if the learner has acquired a grammar allowing more than 2?

(Jäger&Rogers'12) To get evidence that the learner has done this, one needs to include strings [allowing more than 2 embeddings]... [Tests of between 1 and 3 embeddings] seems very near the boundary of practicality for most experiments involving living organisms.

This situation seems quite serious! Is the only evidence that the grammar accepts more than n embeddings only obtainable by checking whether the learner can handle sentences with that many embeddings? If that were true, then it's absolutely clear that we would have no evidence for unbounded embedding of this sort in human languages, as we worried when we looked at *neither... nor* above!

(16) **When does a physical computer use a language $\notin$ Reg?**

The answer is not in your formal language theory text that I have ever seen!

(Hopcroft and Ullman, 1979, p14) The computer itself can be viewed as a finite state system, although doing so turns out to be not as useful as one would like.

(Gurevich, 1988, p412) ... the classical theory of finite-state machines is not adequate to deal with real computers... there are too many states.

... assume infinite when it's useful? When is that?

(17) **Why English $\notin$ Reg**: second try

(Fitch and Frederici, 2012, p1940) Let us assume that human language use *could* be modeled by a FSA, augmented with transition probabilities... such a Markov grammar would require an enormous number of parameters.

- not necessarily enormous unless each is independent of others
- why assume probabilistic grammars with infinitely many require fewer parameters?

(18) **Why English $\notin$ Reg**: third try

(Chomsky, 1965, p3) Linguistic theory is concerned primarily with an ideal speaker-hearer ... unaffected by such grammatically irrelevant conditions as memory limitations...

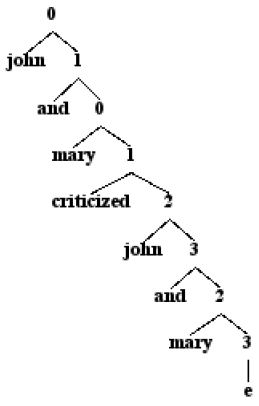
- that is, influences on analysis $\neq$ influences on memory limitations, so outside the domain of the grammar. **A factored account.**

(Miller, 1967) constituent structure languages are more natural, easier to cope with, than regular languages... The hierarchical structure of strings generated by constituent-structure grammars is characteristic of much other behavior that is sequentially organized; it seems plausible that it would be easier for people than would the left-to-right organization characteristic of strings generated by regular grammars

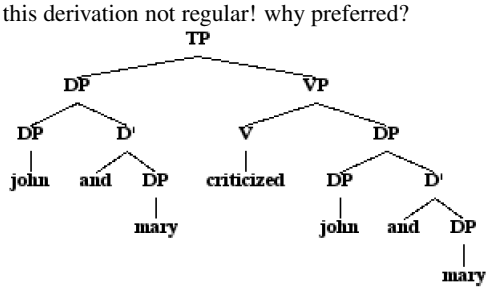
- that is, a wide variety of evidence supports the idea that we are computing structures (for short, intelligible sentences) not definable by regular grammars. **Tis the main point! This is why the supra-regular hypothesis about human language is uncontroversial.**

So we should ask first: what kind of grammar G is being used? Then, we could consider the parasitic question: what is L(G)? So (1) humans use non-regular grammars, and then (2), because of (1), the best theories assume we implement parsers for grammars G such that *neither... nor* embeddings are not bounded.

(19) Consider the regular and context free derivations:



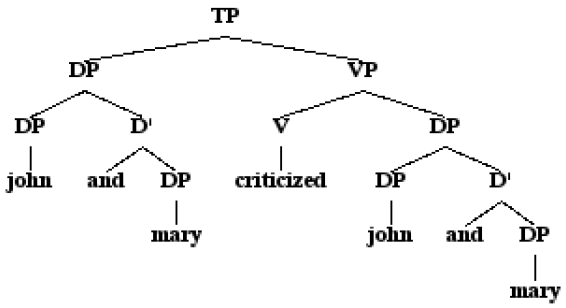
this derivation not regular! why preferred?



- same DPs in diff positions
- new name $\Rightarrow$ both positions
- boundary effects (e.g. click)
- semantic compositionality,...

0.1.3 Context free grammars and languages

(20) **CF:** $A \rightarrow X$ (context free grammar)



TP → DP VP
VP → V DP
V → criticized
DP → DP D'
D' → and DP
DP → john
DP → mary

This grammar G not regular, but L(G) is.

- Not simpler than (12) in the 'false but useful approximation' sense, but in a way that makes it more believable.

(21) We have these hypotheses:

(H1) G(English) $\notin$ Reg $\Leftarrow$ clear!
(H2) L(English) $\notin$ Reg

The strong evidence for H2 is not our judgments about (our 'mastery' of) *neither-nor* and other center-embeddings, but H1.

ARGUMENT SKETCH: Notice that the structure of the embedded clause, given something like (H1) looks very similar to the simple sentence:

the claim that [the claim is funny] is funny
the claim is funny

If we say it is the same, then L(English) $\notin$ Reg.

(22)
- 3 factors: G(HL) + memory limits + lifespan
- G(HL) non-Reg, supporting, for example, L(English) non-Reg

These claims are not approximate, but true, and commonplace. Cf. a computer parses arithmetic expressions.

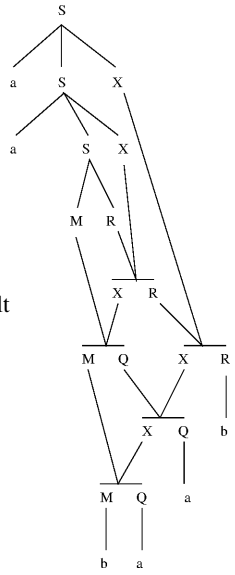
0.2 Context sensitive grammars and languages

(23) **CS:** $X \rightarrow Y$ **with** $|X| \leq |Y|$

S → aSX	S → bSY	S → LQ	S → MR
QX → XQ	RX → XR	QY → YQ	RY → YR
LX → LQ	MX → MQ	LY → LR	MY → MR
L → a	M → b	Q → a	R → b

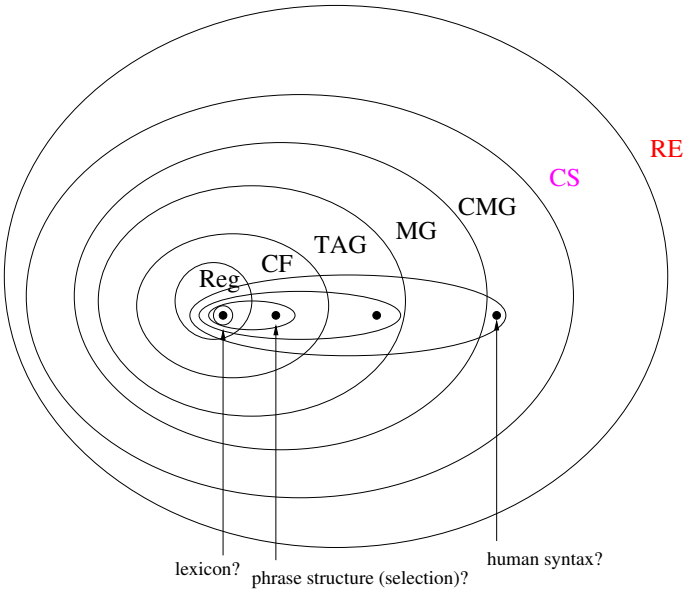
This defines the non-CF language $\{xx | \, x \in \{a, b\}^+\}$ (Mateescu and Salomaa, 1997, Ex.2.2)

Deriving aabaab:



These grammars can define difficult languages, and they are difficult to reason about.

(24) **Refining the Chomsky hierarchy**



(MCS): HLs are weakly and strongly 'mildly context sensitive' (Joshi, 1985)

Strong MCS = HLs are weakly and strongly TAG languages

Weak MCS = HLs are weakly and strongly MG languages

(The smaller loops indicate some classes that we have learners for – we will get to them soon.)

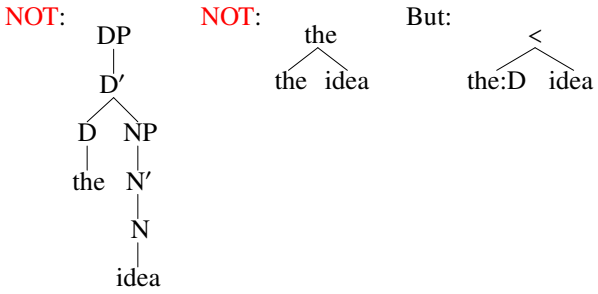
(25) Michaelis and Harkema showed that MGs and multiple context free grammars (MCFGs) define exactly the same languages, (Michaelis, 1998, 2001b; Harkema, 2001), and it was already known Seki et al. (1991) that MCFGs define the same languages as 'set-local' TAGs and certain other formalisms.

(26) Jäger&Rogers'12 suggest that "Most experts…assume at this time that all natural languages are MG languages" (Jäger and Rogers, 2012, p.1961), but there are arguments that human grammars have copying and that the string languages of human languages sometimes show this power, putting them in the class that we call copying MG languages (CMG).

(27) MG,CMG grammars are naturally regarded as defining languages in 2 steps (Mönnich, 1998; Michaelis, Mönnich, and Morawietz, 2000; Morawietz, 2003; Kobele, Retoré, and Salvati, 2007).

0.2.1 Minimalist grammars (MG)

(28) 'bare phrase structure' (Stabler, 1997, 2010; Michaelis, 2001a)



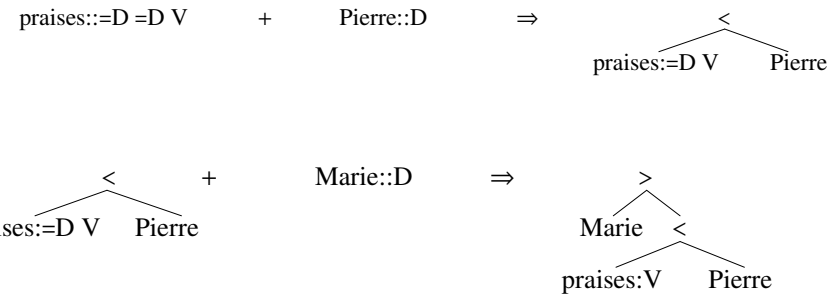
The < "points toward" the **head** of the phrase.
The largest subtree with a given head is a **maximal** projection.

(29) lexical items: Vocabulary paired with feature sequences

every,some,student,...	(vocabulary)
C, T, D, N, V, P,...	(categories)
=C, =T, =D, =N, =V, =P,...	(selectors)
+wh, +case, +focus,...	(licensors)
-wh, -case, -focus,...	(licensees)

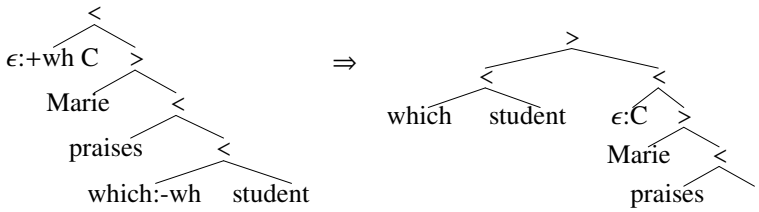
Examples: **Marie::D**
who::D -wh
praises::=D =D V
ϵ::=I +wh C

(30) **External merge (•)** complements on right, additional selected elements on left



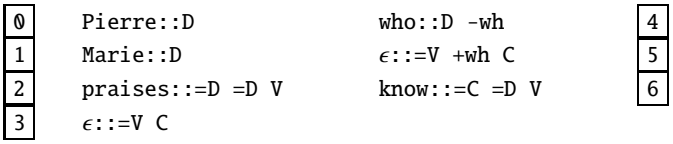
(2 features deleted, and :: in lexical items changes to : in derived structures)

(31) **Internal merge (∘)** in a tree whose head has first feature +f, move maximal -f subtree specifier position:

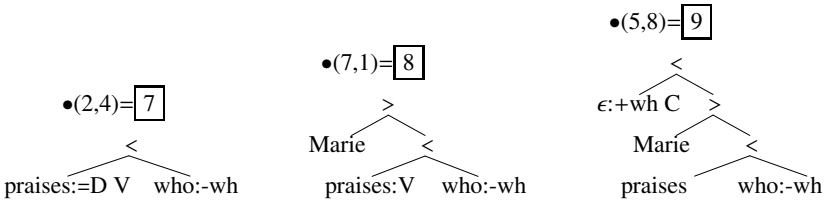


(SMC) ∘ applies only when exactly 1 head has -f first feature

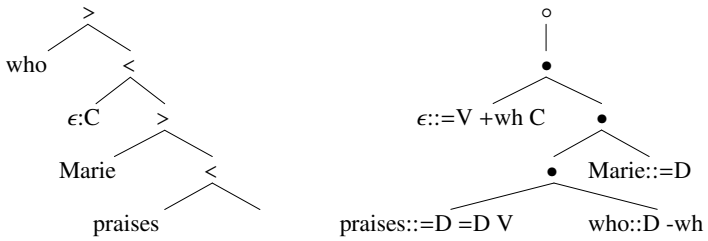
(32) **example grammar**:



steps 1,2,3



step 4 **all 4 steps**

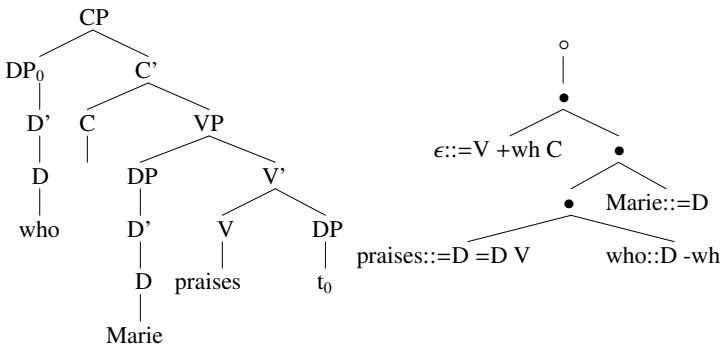


∘(9)**=derived tree** **derivation tree**

(completed derivation with 1 feature left; 8 features checked in total)

The mapping from the derivation tree on the right to the derived tree on the left is easy to calculate! (Kobele, Retoré, and Salvati, 2007)

step 4 **all 4 steps**

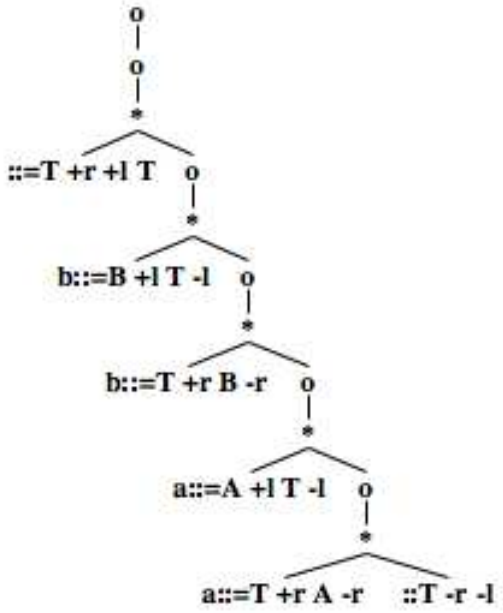


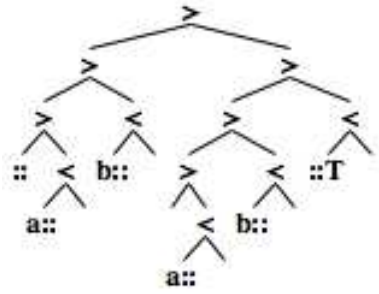
derived tree **derivation tree**

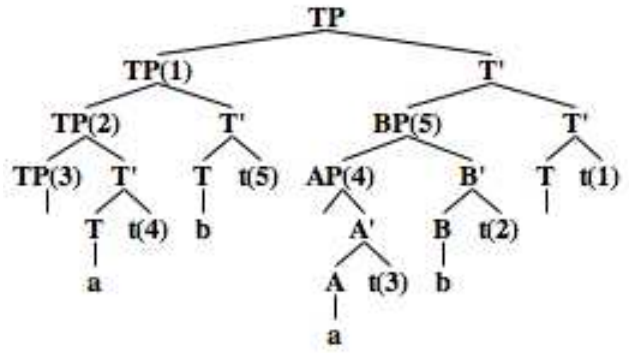
The mapping from the derivation tree on the right to the derived tree on the left is easy to calculate!

(33) A grammar for the non-CF language $\{xx | \ x \in \{a, b\}^*\}$

ϵ::T −r −l	ϵ::=T +r +l T
a::=T +r A −r	b::=T +r B −r
a::=A +l T −l	b::=B +l T −l







(34) **MG in 2 finite state steps**: this account is detailed in Kobele, Retoré, and Salvati (2007)

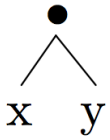
0. form derivation (trivial!)
1. check derivation (at interfaces?) (FS dbutt)
2. map to PF/LF (FS dmbutt)

nb: There are many ways to do 0,1,2, and of course these steps could be interleaved or folded together in an implementation. Standard grammars have these steps all folded together.

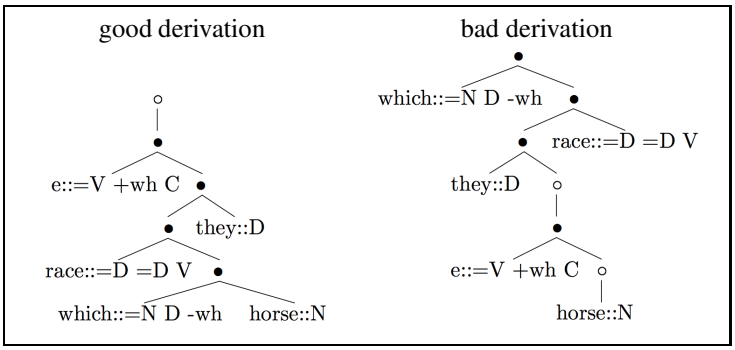
FS d(m)butt = finite state deterministic (multiple) bottom-up tree transducer

0. form derivation: merge

Form derivation by combining any two lexical items or derivations:



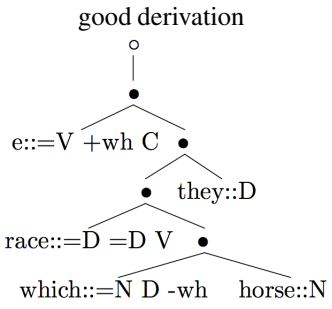
or by 'combining' with something already in the derivation:

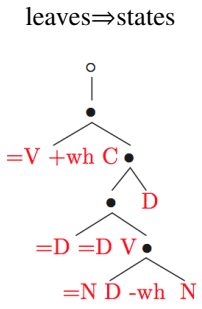


1. check derivation: 'deterministic bottom-up tree transducer'

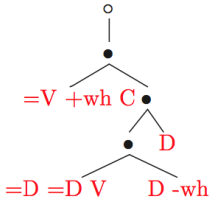
- At the leaves, let the 'state' be the features of the lexical items.
- Compute 'state' of internal nodes by feature checking.
- Derivation is good iff the state of the root is *C* (or whatever counts as an acceptable phrasal category)

Checking the good derivation:

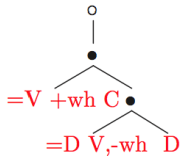




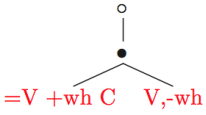
now up the tree...



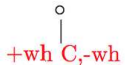
DP features



V' features



VP features

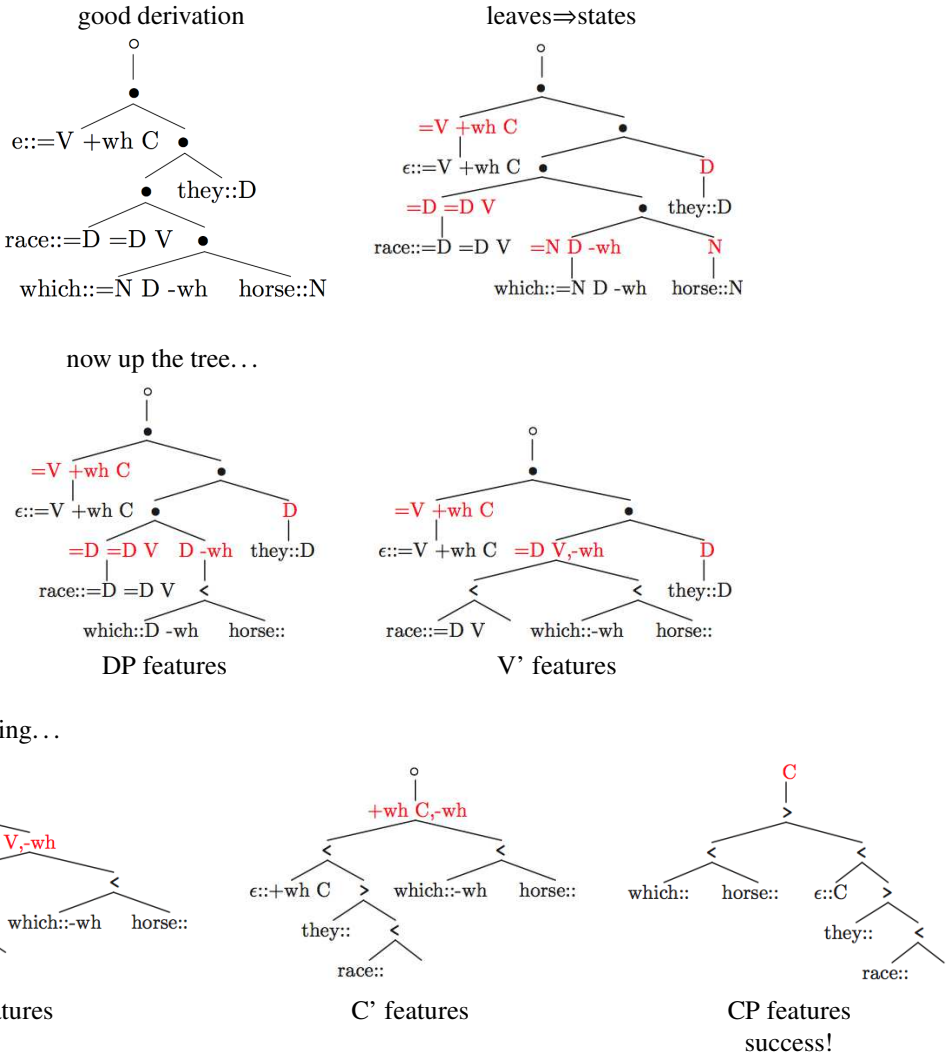


C' features

C

CP feature
Success!

(35) **2. (Ck tree in course of) mapping to PF/LF**



0.3 Summary

- The Chomsky hierarchy is like a map of all possible languages, with regions defined by the kinds of rules you can use to define the languages there (*rules that determine analyses of the expressions in those languages*).

  We will explore various regions of this map, in some detail, in this class!

- When a learner is born/created in a community using a language somewhere on this map, some language in the Chomsky hierarchy, is there a way for the learner to determine, from listening to the language, where the language is on this map? We will see that, in general, there is no way to do this, without 'universal' assumptions about what is to be learned. This is the puzzle we will be struggling with.

- One big division in the Chomsky hierarchy is between the languages that can be recognized by finite devices (with a finite amount of memory) – these are the **regular languages**, the **finite state languages** – vs. languages that cannot be recognized with finite memory.

- The language of arithmetic is non-regular. It includes expressions like

  $$5 + ((2 \times 3)/4)$$

  but not expressions like

  $$))5 + \times 2/3 - (+4(.$$

  One very basic way to show this is to observe that the open and close parentheses must match in arithmetic. An expression can begin with n open parentheses, for any n, but it is only well-formed if each open parenthesis has a matching close parenthesis. Obviously, you can't do this with finite memory, because when n gets high enough, the finite memory will not be able to count that high!

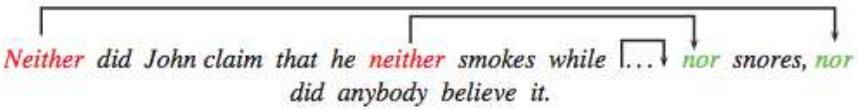
  Similarly for the slightly simpler language $a^n b^n$. In this language, for every a, there must be a corresponding b. Obviously, a finite device cannot enforce that restriction for all n.

  (The Myhill-Nerode theorem formalizes this reasoning - coming later!)

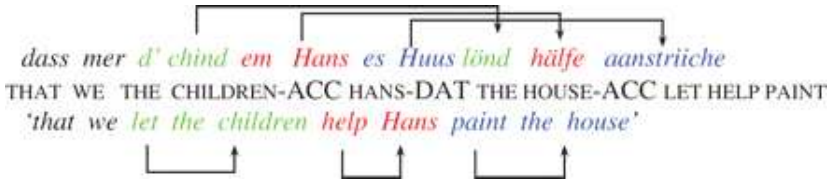
A 'standard' story: (to be rejected below!)

s1. Although finite devices cannot recognize non-regular languages, it is *simpler* and more *useful* to regard certain human (and laptop computer) languages as non-regular. (This is now a consensus position.)

s2. The main evidence for the non-regularity of English is the existence of $a^n b^n$ patterns, such as we see in sentences of the form



s3. Chomsky (1956) argued that context free grammars were not appropriate for human languages, but the existence languages with non-context-free string sets was not uncontroversially established until 1985 with the publication of a paper showing a crossing dependency pattern in Bambara compounding Culy (1985), and another showing a crossing dependency pattern Shieber (1985) in Swiss-German:



(I think even these 'uncontroversial' results are misunderstood, as will be discussed later. But this is the conventional picture.)

Both of the pictures above are from Jäger and Rogers (2012). Compare, e.g. Peters (1987), which seems to offer a version of this same story but then comes close to the grammar-based alternative we recommend below.

Problems for the previous perspective:

P1. When is it correct to say that a device is using a non-regular language?

(This question is directly relevant, because we also want to know: how could a learner detect whether the language in use is a regular language?)

The standard answers are not satisfying! Assume the language is non-regular whenever...

A1. a regular grammar would need *lots* of states.

A2. a non-regular grammar would be *significantly simpler*.

A3. the language users shows *mastery* of embeddings $a^n b^n$ for *sufficiently high n*.

A4. the language users can *potentially* handle $a^n b^n$ for any n.

P2. Humans can handle nested *neither*^{n} ... *nor*^{n} ... only to about $n = 2$ or 3. This seems very poor evidence of 'mastery' of the non-regular pattern.

Furthermore, this is evidence about English only. Many other languages have similar patterns, but do all human languages have them? Standard discussions provide no argument for this.

P3. Swiss-German speakers can handle crossing dependencies $O_1 O_2 \ldots O_n V_1 V_2 \ldots V_n$ only to about $n = 2$ or 3. This seems very poor evidence of 'mastery' of the non-CF pattern.

Furthermore, this is evidence about Swiss-German only. Do many other languages have similar patterns? Standard discussions provide no argument for this.

The following different perspective reinterprets the standard claims so that they make more sense.

An alternative perspective, differing in the way the basic terms are used, and in the way evidence bears on claims about language users, can be given in something like the following way:

ES1. Laptop computers and all other artifacts, like humans and all other organisms, are finite state machines.

ES2. **X uses L** $=_{df}$ **X uses some G** such that **L=L(G)**.

X uses G $=_{df}$ at least sometimes, in appropriate conditions for fluent language use, **the analyses that X computes for utterances are those defined by G**. **X implements G.**

Human languages have property P $=_{df}$ normal, fluent humans use grammars G such that each such G – or the associated string language $L(G)$ – has property P.

ES3. We can now claim, without contradicting ES1, that human languages are non-regular, and we can explain persuasively why this is a consensus without appealing to 'mastery' of $a^n b^n$ patterns for high n. See (19)

- A parser that uses a stack to process $a^n b^n$ in a standard way, but only up to some fixed depth n (e.g. 2 or 3), is finite state, a 'recognizer' for only a regular language, but it uses a non-regular language in our sense.
- Your laptop computer uses the non-regular language of arithmetic, in this sense.
- The claim that human languages are non-regular is an empirical claim about the analyses computed (with consequences, as we will see, for the memory that must be available in the course of that computation).
- If all other aspects of grammar and processing failed to support the idea that we use a non-regular grammar, then the fact that we can handle *neither*^{n} ... *nor*^{n} ... to $n = 2$ or 3 would not support it either!

(simplicity,usefulness) If human languages have property P, and one kind of grammar ensures that by having lots of rules for many instances of P, where the another kind of grammar can define P simply, the latter is usually going to be more believable. E.g. if each instance is covered by a separate rule, we would expect grammars convering some instances but not others.

(potential,idealization) With an abstract mathematical definition of a machine, the assumption of infinite memory makes perfect sense. This is what we use when we say that for some grammar G for a human or computer language, L(G)$\notin$Reg. There is no implication that humans or laptops are infinite. There is no implication of any real potential for adding more memory to any human or laptop.

ES4. We will also claim, without contradicting ES1, that human languages are non-CF, and we can explain persuasively why this is a consensus without appealing to 'mastery' of $a_1 \ldots a_n b_1 \ldots b_n$ patterns for high n.

Stabler (2013) formalizes Chomsky's 1956 argument for this – coming later. (Contrast s3.)

Puzzle to be solved in this class: how can a language learner tell $L \notin Fin$? $L \notin Reg$? $L \notin CF$?

Appendix A: the Chomsky hierarchy more carefully

(36) In studying human language, we idealize, abstracting away from less relevant factors. So for a simple start, let's think of a language as a (non-empty) set of expressions built from some (finite, non-empty) vocabulary Σ. But, of course, what we are actually interested in is how linguistic structure is assigned to anything – this will show in, *inter alia*, our lack of interest in the elements of Σ.

In learnability theory, we extend this standard idealization, thinking of what can be learned as a set of languages. But what we are actually interested in is how we come to recognized linguistic structure in anything, and particularly in how we generalize from our limited experience.

The sets of expressions are a useful idealization in the study of the computational mechanisms for coming to recognize structure.

(37) Chomsky noticed in the 1950's that we can distinguish sets of languages according to the complexity of the patterns that can occur in those languages. This idea is one of the fundamental ones in the theory of computation.

(38) A **rewrite grammar** $G = \langle Cat, \Sigma, S, R\rangle$ where

a. Cat (categories) is a finite nonempty set;

b. Σ (vocabulary) is a finite nonempty set such that $Cat \cap \Sigma = \emptyset$;

c. $S \in Cat$ (the "start" or "sentence" category);

d. $R \subseteq ((\Sigma \cup Cat)^* \times Cat \times (\Sigma \cup Cat)^*) \times (\Sigma \cup Cat)^*$ a finite set (the rewrite rules)

(39) Let ϵ be the empty sequence. For any set S, let $S^\epsilon = S \cup \{\epsilon\}$

(40) We often write $x \rightarrow y$ to indicate that $\langle x, y\rangle \in R$;

(41) For any $w, z \in (\Sigma \cup Cat)^*$, we say $w \Rightarrow z$ iff there are $x, y, u, v \in (\Sigma \cup Cat)^*$ such that

a. $w = uxv$,

b. $z = uyv$, and

c. $x \rightarrow y$.

(42) The relation $\Rightarrow^+$ is the transitive closure of $\Rightarrow$.

The relation $\Rightarrow^*$ is the reflexive, transitive closure of $\Rightarrow$.

(43) Given a rewrite grammar G, $L(G) = \{s \in \Sigma^* | \ S \Rightarrow^* s\}$.

(44) A rewrite grammar G is

a. a **list grammar** iff $R \subseteq (\{S\} \times \Sigma^*)$;

b. a **regular (right branching, right linear) grammar** iff $R \subseteq ((Cat \times (\Sigma \times Cat)) \cup (Cat \times \{\epsilon\}))$;

c. a **context free grammar** iff $R \subseteq Cat \times (\Sigma \cup Cat)^*$;

d. a **context sensitive grammar** iff $R \subseteq \{\langle x, y\rangle | \ x, y \in (\Sigma \cup Cat)^*, \ |x| \leq |y|\}$, allowing $S \rightarrow \epsilon$ iff S does not occur on the right side of any rule.

(45) a. The set of languages generated by list grammars is the set $\mathscr{L}_{fin}$ of finite languages;

b. The set of languages generated by regular (right branching) grammars is the set $\mathscr{L}_{fs}$ of regular, or finite state languages;

c. The set of languages generated by context free grammars is the set $\mathscr{L}_{cf}$ of context free languages;

d. The set of languages generated by context sensitive grammars is the set $\mathscr{L}_{cs}$ of context sensitive languages;

e. The set of languages generated by all the grammars is the set $\mathscr{L}_{re}$ of recursively enumerable languages.

(46) The Chomsky hierarchy specifies proper inclusion relations among collections of languages as follows:

$\mathscr{L}_{re}$, recursively enumerable languages

$\subset$

$\mathscr{L}_{rec}$, recursive languages

$\subset$

$\mathscr{L}_{cs}$, context sensitive languages

$\subset$

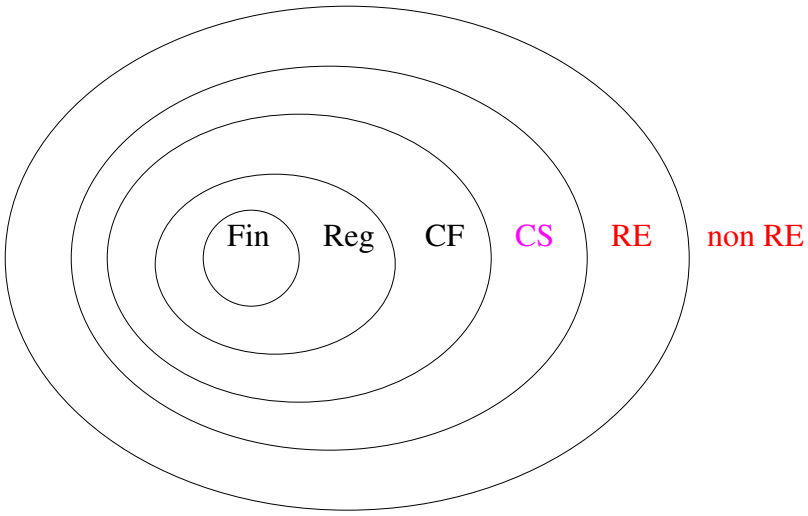
$\mathscr{L}_{cf}$, context free languages

$\subset$

$\mathscr{L}_{fs}$, finite state languages

$\subset$

$\mathscr{L}_{fin}$, finite languages



Many more details and proofs of all of these results can be found in, for example, (Salomaa, 1973; Harrison, 1978; Hopcroft and Ullman, 1979). The early Chomsky papers (Chomsky, 1963; Miller and Chomsky, 1963) are dated, but still interesting.

Appendix B: enumerations

(47) An enumeration is a sequence, a list, a function whose domain is the positive integers or $\mathbb{N}$.

A finite sequence is a function whose domain is $\{0, 1, \ldots, n\}$ for some positive integer n.

(48) An enumerable set is one whose members can be enumerated.[1] ...By courtesy, we regard as enumerable the empty set, $\emptyset$, which has no members.

Sometimes an enumerable set is called "denumerable."

(49) For any vocabulary Σ (i.e. for any finite, nonempty set), Σ^* is enumerable.

There are many enumerations of Σ^*. Let's define one.

Let $\Sigma^0 = \{\epsilon\}$.

Let $\Sigma^1 = \{\langle x\rangle |\ x \in \Sigma\}$.

For $k \geq 1$, let $\Sigma^{k+1} = \{\langle x_1, \ldots, x_k, x_{k+1}\rangle |\ \langle x_1, \ldots x_k\rangle \in \Sigma^k \text{ and } x_{k+1} \in \Sigma\}$.

Notice that each of the sets $\Sigma^0, \Sigma^1, \ldots$ is finite.

Define a total order $<$ on Σ (e.g. "alphabetical" or "numeric" order).

Then extend the order to Σ^* as follows: $x < y$ iff either

a. $|x| < |y|$ or

b. $|x| = |y|$ and there is some i such that for all $0 \leq j < i$, the j'th element of x is the same as the j'th element of y and the $i'th$ element of $x <$ the $i'th$ element of y.

To enumerate Σ^* in order, list all elements of each of $\Sigma^0, \Sigma^1, \ldots$, in order.

Every element of Σ^* appears in this sequence at some finite point.

(50) The set of positive rational numbers is enumerable.

There are many other ways to get an enumeration, but here is one. The following matrix has all of the positive rational numbers in it:

$$\begin{array}{cccccc}
\frac{1}{1} & \frac{2}{1} & \frac{3}{1} & \frac{4}{1} & \frac{5}{1} & \cdots \\
\frac{1}{2} & \frac{2}{2} & \frac{3}{2} & \frac{4}{2} & \frac{5}{2} & \cdots \\
\frac{1}{3} & \frac{2}{3} & \frac{3}{3} & \frac{4}{3} & \frac{5}{3} & \cdots \\
\frac{1}{4} & \frac{2}{4} & \frac{3}{4} & \frac{4}{4} & \frac{5}{4} & \cdots \\
\frac{1}{5} & \frac{2}{5} & \frac{3}{5} & \frac{4}{5} & \frac{5}{5} & \cdots \\
\cdots & & & & &
\end{array}$$

And listing elements of the previous matrix in the the following order, we will get to each entry at some finite point:

$$\begin{array}{cccccc}
1 & 2 & 6 & 7 & 15 & \cdots \\
3 & 5 & 8 & 14 & \cdots & \\
4 & 9 & 13 & \cdots & & \\
10 & 12 & \cdots & & & \\
11 & \cdots & & & & \\
\cdots & & & & &
\end{array}$$

The fact that this enumeration lists each rational number infinitely often does not matter; the thing we require for this to count as an enumeration of the rationals is just that every rational number appear (at some finite position).

(51) Using the same method, we see that for any vocabularies Σ_1, Σ_2, the set $\Sigma_1^* \times \Sigma_2^*$ is enumerable.

(52) The set of all grammars is enumerable.

Once again, there are many ways to do this, but let's define one enumeration in a little bit of detail. Assume that the elements of Σ are alphabetic, and order them alphabetically.

The set *Cat* for each grammar is, by definition of grammar, finite and disjoint from Σ. We can think of each such set as a finite subset of $\mathbb{N}$, totally ordered by $\leq$. We can extend this order to $\mathbb{N} \cup \Sigma$ by and ordering all of Σ before $\mathbb{N}$.

We can then order $((\Sigma \cup Cat)^* \times Cat \times (\Sigma \cup Cat)^*)$ in just the way we ordered Σ in (49) – these are the left sides of the rules of rewrite grammars.

[1] This is the first sentence of Boolos and Jeffrey (1980). The first chapters of this book are a good place to go for more background on enumerability. Other good places (though more advanced) are the first chapter of Kleene (1952) or part I of Smullyan (1993).

We similarly order $(\Sigma \cup Cat)^*$ – the right sides of the rewrite rules.

Since rewrite rules are nothing more than left-side/right-side pairs, we can use the construction in (50) to enumerate them all. Assuming that the categories *Cat*, vocabulary Σ and start symbol S are given, we can then enumerate grammars with 0 symbols, grammars with 1 symbol, grammars with 2 symbols, and so on, to get all grammars.[2]

(53) The set $\wp(\mathbb{N})$ of all sets of natural numbers is not enumerable. (Cantor)

Proof: Suppose for contradiction that $E = \langle s_1, s_2, s_3, \ldots \rangle$ is an enumeration of $\wp(\mathbb{N})$.

Define the set $nondiag(E) = \{n \in \mathbb{N} | \ n \notin s_n\}$.

(For example, if s_3 happens to be the even numbers, then $3 \in nondiag(E)$.)

Since E is a list of all sets of integers, for some i, $nondiag(E) = s_i$.

By the definition of $nondiag(E)$, $i \in nondiag(E)$ iff $i \notin s_i$.

But then since $nondiag(E) = s_i$, $i \in nondiag(E)$ iff $i \notin nondiag(E)$. ↯

(54) (Corollary) The set of all languages is not enumerable.

[2] We could let *Cat* and Σ be infinite enumerable sets too, and still enumerate all grammars, again using a variant of the strategy in (50).

Appendix C: coding the enumerations

(55) Writing a program can be a useful test to make sure you understand what we are describing. Programmers usually care about whether a program terminates, and we will too, even though we are setting that worry aside for now.

(56) It is easy to write a program that enumerates (i.e. lists) the natural numbers

$$\mathbb{N} = \{0, 1, 2, \dot{}\}$$

In 'pseudocode' we could say:

```
i = 0
while i ≥ 0 :
   print i
   i = s(i)
```

(57) Most programming languages will do the decimal notation for you, so you can simply say something like:

```
for i = 0 to ∞ :
  print i
```

Most languages do not have the symbol ∞, but in Python you can write that program this way:

```
from itertools import *  # for count

for i in count(start=0,step=1):
    print i
```

To allow us to interupt gracefully at any point, it is nicer to add a check, like this:

```
from itertools import *  # for count

for i in count(start=0,step=1):
    print i
    ck = raw_input('more? ')
    if len(ck)>0 and ck[0] == 'n':
        break
```

(58) Clearly we can also enumerate the non-positive integers:

$$Neg = \{0, -1, -2, \ldots\}$$

```
from itertools import *  # for count

for i in count(start=0,step=-1):
    print i
    ck = raw_input('more? ')
    if len(ck)>0 and ck[0] == 'n':
        break
```

(59) To enumerate $\mathbb{N} \cup \mathbb{N}eg$, though, we cannot simply concatenate the two previous programs! The obvious simple idea, prominent in Cantor's famous proofs, is to, in effect, do the two enumerations at once. So for example, we could say:

```
from itertools import *  # for count

for i in count(start=0,step=-1):
```

```
    print i
    print (-1 * i)
    ck = raw_input('more? ')
    if len(ck)>0 and ck[0] == 'n':
        break
```

(60) Now let's consider how to enumerate all possible grammars. The trick is that we want to enumerate grammars of size 0,1,2,... as the number of categories is also increasing 0,1,2,... So we cannot use a program of this form:

```
from itertools import *  # for count

for size in count(start=0,step=-1):
    for no_of_cats in count(start=0,step=-1):
        list all grammars of size, number_of_cats
```

The problem is that for `size=0`, we already would have infinitely many cycles through the inner loop! So we use some version of Cantor's trick. The following code written by Jos Tellings elegantly enumerates all possible grammars:

```
# Jos Tellings
# file: Tellings-enumerateG.py
# Oct 7, 2012

from itertools import *
alph = ['a','b'] # fix alphabet
flag = False

# from itertools docs:
def powerset(iterable):
    "powerset([1,2,3]) --> () (1,) (2,) (3,) (1,2) (1,3) (2,3) (1,2,3)"
    s = list(iterable)
    return chain.from_iterable(combinations(s, r) for r in range(len(s)+1))

# we are diagonalizing on the size of NT and the max size of the rules
# (if a --> b is a rule, the size of that rule is defined as |a| + |b|)
for k in count(start=1, step=1):
        for m in range(1,k):
                size = k-m         # size of NT
                length = m        # max length of rules
                symb = map(str,range(size)) # convert numbers to strings
                symb[0:0] = alph # set of NT + alphabet
                lijst = [] #initialize
                for mu in range(length+1): # all lengths up to max length
                        rules = product(symb,repeat=mu)
                        for z in rules:
                                word = ''.join(z) #create a string
                                for w in range(1,mu+1):
                                        lijst.append([word[0:w],word[w:mu]]) # split in all possible ways
                                                                            # (left side is non-empty)
                print 'size NT =', size, ', max length =', m, ', no of such grammars =', 2**len(lijst)
                everything = powerset(lijst) #all possible combinations of rules
                for u in everything:
                        grammar = [alph,range(size),list(u)]
                        print grammar
                        stop = raw_input("Enter = continue, q+Enter = stop, \
                                          c+Enter= go to next set of grammars ") #flow control
                        if stop=="q":
```

```
                                flag = True #break out of all loops
                                break
                        elif stop=="c": break #break only one loop
                if flag: break
        if flag: break
```

References

References

Aho, Alfred V. and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling. Volume 1: Parsing*. Prentice-Hall, Englewood Cliffs, New Jersey.

Angluin, Dana. 1982. Inference of reversible languages. *Journal of the Association for Computing Machinery*, 29:741–765.

Bane, Max, Jason Riggle, and Morgan Sonderegger. 2010. The VC dimension of constraint-based grammars. *Lingua*, 120(5):1194 – 1208.

Blumer, Anselm, Andrzei Ehrenfeucht, David Haussler, and Manfred K. Warmuth. 1989. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36:929–965.

Boolos, George S. and Richard C. Jeffrey. 1980. *Computability and Logic, 2nd Edition*. Cambridge University Press, NY.

Chomsky, Noam. 1956. Three models for the description of language. *IRE Transactions on Information Theory*, IT-2:113–124.

Chomsky, Noam. 1963. Formal properties of grammars. In R. Duncan Luce, Robert R. Bush, and Eugene Galanter, editors, *Handbook of Mathematical Psychology, Volume II*. Wiley, NY, pages 323–418.

Chomsky, Noam. 1965. *Aspects of the Theory of Syntax*. MIT Press, Cambridge, Massachusetts.

Chomsky, Noam. 1981. *Lectures on Government and Binding*. Foris, Dordrecht.

Clark, Alexander and Rémi Eyraud. 2007. Polynomial identification in the limit of substitutable context-free languages. *Journal of Machine Learning Research*, 8:1725–1745.

Clark, Alexander and Franck Thollard. 2004. PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5:473–497.

Culy, Christopher. 1985. The complexity of the vocabulary of Bambara. *Linguistics and Philosophy*, 8(3):345–352.

Fitch, W. Tecumseh and Angela D. Frederici. 2012. Artificial grammar learning meets formal language theory: An overview. *Philosophical Transactions of the Royal Society B*, 367:1933–1955.

Fitch, W. Tecumseh, Angela D. Frederici, and Peter Hagoort. 2012. Pattern perception and computational complexity: Introduction. *Philosophical Transactions of the Royal Society B*, 367:1925–1932.

Frank, Robert and Giorgio Satta. 1998. Optimality theory and the generative complexity of constraint violability. *Computational Linguistics*, 24:307–315. http://aclweb.org/anthology/J/J98/J98-2006.pdf.

Gurevich, Yuri. 1988. Algorithms in the world of bounded resources. In Rolf Herkin, editor, *The Universal Turing Machine: A Half-Century Survey*. Oxford University Press, Oxford, pages 407–416.

Harkema, Henk. 2001. A characterization of minimalist languages. In *Logical Aspects of Computational Linguistics*, LNCS 2099, pages 193–211, Springer, NY.

Harrison, Michael A. 1978. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, Massachusetts.

Heinz, Jeffrey. 2010. Learning long-distance phonotactics. *Linguistic Inquiry*, 41(4):623–661.

Heinz, Jeffrey, Gregory M. Kobele, and Jason Riggle. 2009. Evaluating the complexity of optimality theory. *Linguistic Inquiry*, 40(2):277–288.

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

Jäger, Gerhard. 2002. Gradient constraints in finite state OT: The unidirectional and the bidirectional case. In Joseph Greenberg, editor, *More than Words: A Festschrift for Dieter Wunderlich*. Akademie Verlag, Berlin, pages 299–325.

Jäger, Gerhard and James Rogers. 2012. Formal language theory: Refining the Chomsky hierarchy. *Philosophical Transactions of the Royal Society B*, 367:1956–1970.

Joshi, Aravind K. 1985. How much context-sensitivity is necessary for characterizing structural descriptions. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Processing: Theoretical, Computational and Psychological Perspectives*. Cambridge University Press, NY, pages 206–250.

Kleene, Stephen Cole. 1952. *Introduction to Metamathematics*. North Holland, NY.

Kobele, Gregory M., Christian Retoré, and Sylvain Salvati. 2007. An automata-theoretic approach to minimalism. In *Model Theoretic Syntax at 10. ESSLLI'07 Workshop Proceedings*.

Mateescu, Alexandru and Arto Salomaa. 1997. Aspects of classical language theory. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Volume 1: Word, Language, Grammar*. Springer, NY, pages 175–251.

Michaelis, Jens. 1998. Derivational minimalism is mildly context-sensitive. In *Proceedings, Logical Aspects of Computational Linguistics, LACL'98*, pages 179–198, Springer, NY.

Michaelis, Jens. 2001a. *On Formal Properties of Minimalist Grammars*. Ph.D. thesis, Universität Potsdam. *Linguistics in Potsdam 13*, Universitätsbibliothek, Potsdam, Germany.

Michaelis, Jens. 2001b. Transforming linear context free rewriting systems into minimalist grammars. In *Logical Aspects of Computational Linguistics*, LNCS 2099, pages 228–244, Springer, NY.

Michaelis, Jens, Uwe Mönnich, and Frank Morawietz. 2000. Derivational minimalism in two regular and logical steps. In *Proceedings of the 5th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, pages 163–170.

Miller, George A. 1967. Project grammarama. In *Psychology of Communication*. Basic Books, NY.

Miller, George A. and Noam Chomsky. 1963. Finitary models of language users. In R. Duncan Luce, Robert R. Bush, and Eugene Galanter, editors, *Handbook of Mathematical Psychology, Volume II*. Wiley, NY, pages 419–492.

Mönnich, Uwe. 1998. TAGs M-constructed. In *TAG+ Workshop, Institute for Research in Cognitive Science, University of Pennsylvania*.

Morawietz, Frank. 2003. *Two Step Approaches to Natural Language Formalisms*. de Gruyter, Berlin.

Mukherjee, Sayan, Partha Niyogi, Tomaso Poggio, and Ryan Rifkin. 2004. Learning theory: stability is sufficient for generalization and necessary and sufficient for consistency of Empirical Risk Minimization. *Advances in Computational Mathematics*, 25(1-3):161–193.

Peters, P. Stanley. 1987. What is mathematical linguistics? In Walter J. Savitch, Emmon Bach, William Marsh, and Gila Safran-Naveh, editors, *The Formal Complexity of Natural Language*. Reidel, Dordrecht.

Pinker, Steven. 1982. A theory of the acquisition of lexical interpretive grammars. In Joan Bresnan, editor, *The mental representation of grammatical relations*. MIT Press, Cambridge, Massachusetts.

Salomaa, Arto. 1973. *Formal Languages*. Academic, NY.

Seki, Hiroyuki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

Shieber, Stuart M. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–344.

Smullyan, Raymond M. 1993. *Recursion Theory for Metamathematics*. Oxford University Press, Oxford.

Stabler, Edward P. 1997. Derivational minimalism. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics*, Lecture Notes in Computer Science 1328. Springer-Verlag, NY, pages 68–95.

Stabler, Edward P. 2010. Computational perspectives on minimalism. In Cedric Boeckx, editor, *Oxford Handbook of Minimalism*. Oxford University Press, Oxford, pages 617–641.

Stabler, Edward P. 2013. Two models of minimalist, incremental syntactic analysis. *Topics in Cognitive Science*, 5(3):611–633.

Yoshinaka, Ryo. 2010. Polynomial-time identification of multiple context-free languages from positive data and membership queries. In *Proceedings of the 10th International Colloquium on Grammatical Inference*, LNCS 6339, Springer-Verlag, Berlin.

Yoshinaka, Ryo and Alexander Clark. 2010. Polynomial time learning of some multiple context-free languages with a minimally adequate teacher. In *Proceedings of the 15th Conference on Formal Grammar*.

Chapter 1 Learning a language from examples: Gold paradigm

1.0 Basics

(0) Gold (1967) and Blum and Blum (1975) established one perspective from which language can be studied. This perspective is developed in various ways in our texts (Jain et al., 1999; Kanazawa, 1998; Laird, 1988).

We will consider this perspective first, for two reasons:

- It provides a simple and completely clear perspective on some aspects of learning.
- It allows us to focus our attention on the critical step: generalization.

(As noted earlier, by "generalization," I will mean conjecturing that the environment contains more than has been seen – this may not be the standard usage, but it's what I will mean.)

(1) We will think of languages as sets of expressions together with a non-expression #, where an expression is an element of Σ^* for some (finite,non-empty) vocabulary Σ.

(2) A **(positive) text** T is an infinite sequence whose elements are expressions and #.

Since we regard an infinite sequence as a function with domain $\mathbb{N} = \{0, 1, \ldots\}$, we let $T(n)$ be the $n'th$ member of T counting up from 0.

Let $T[n]$ be the length n initial sequence of T. So $T[n] = \langle T(0), T(1), \ldots T(n-1)\rangle$.

(3) Let the **content of text** T, $content(T)$ be the set of expressions that appear in T. We say that **text** T **is for language** L iff $content(T) = L$. Note that # is never part of the content.

Then, for every text T, and for every $x \in content(T)$, there is some finite i such that $x = T(i)$.

(4) A **learner** is a (possibly partial) function ϕ from finite sequences of expressions and # to grammars, where each grammar G defines a language $L(G)$.

We now define a certain kind of learning, which is sometimes called **identification from positive text**, as follows.

(5) If $x \in Dom(\phi)$ we say ϕ is **defined on** x, or $\phi(x)\downarrow$.

(6) ϕ **converges** on T, $\phi(T)\downarrow$, iff for some i, $\phi(T[j]) = \phi(T[i])$ for all $j \geq i$.
In this case, we define $\phi(T)$ to be $\phi(T[i])$.

(7) ϕ **identifies** T iff $\phi(T)\downarrow$ and $content(T) = L(\phi(T))$.

(8) ϕ **identifies** L iff ϕ identifies all texts for L.

(9) ϕ **identifies** a class of languages $\mathscr{L}$ iff for all $L \in \mathscr{L}$, ϕ identifies L.[1]

(10) A class of languages $\mathscr{L}$ is **identifiable** iff some learner identifies $\mathscr{L}$.

Sometimes, when the context makes our intention clear, we will call such a class **learnable**.

1.1 First results

Theorem 1 The classes $\mathscr{L} = \emptyset$ and $\mathscr{L} = \{L\}$ for any r.e. language L are identifiable.

(11) The previous theorem needs the qualification "r.e" (recursively enumerable) since we are assuming that languages are represented by grammars (or machines) that recursively enumerate them.

[1] I will use the term 'class' for sets of languages, just to avoid confusions with languages themselves which are sets of sequences over an alphabet Σ. And I'll try to use $\mathscr{L}$ rather than L to denote such classes.

Theorem 2 The class $\mathscr{L}_{fin}$ of finite languages is identifiable. (Gold, 1967)

Proof: Suppose that we have an enumeration of grammars for all the r.e. languages. Then define a learner ϕ_e as follows: for any text T, $\phi_e(T[i])$ is the first grammar G in the enumeration such that $L(G) = content(T[i])$.

Consider any $L \in \mathscr{L}_{fin}$ and any text T for L. Then there is some i such that $content(T[i]) = L$. In this case, by our definition of ϕ_e, $\phi_e(T[j]) = \phi_e(T[i])$ for all $j \geq i$, so ϕ_e converges on t. And again by our definition of ϕ_e, $L(\phi_e(T[i])) = content(T)$. □

(12) Notice that we did not define the enumeration of grammars precisely; but it can be done without too much trouble.

(13) Notice also that nothing in the definition of learnability implies that learners are necessarily computable functions, and so we did not need to explain how the function ϕ_e could be computed. In this case, the decision about whether $L(\phi_e(T[i])) = content(T)$ is obviously not computable in general.

So if we are interested in a computable learner, we could, for example, define an effective enumeration of grammars for just the finite languages, and choose the first of these that matches the sample. Or we could let $\phi_e(T[i])$ be the grammar $\{S \rightarrow x |\ x \in content(T[i])\}$.

(14) Gold (1967) calls the learner ϕ_e defined in the proof above an **identification-by-enumeration** learner. Notice that nothing in ϕ_e's values signals when it has converged on the right hypothesis.

(15) It is interesting to contrast learning functions which announce their convergence ("Aha!" or, for short "∅"), in the following way:

a. on any text T there is a unique i such that $L(\phi(T[i])) = \emptyset$, and

b. on any text T if $L(\phi(T[i])) = \emptyset$ then for all $j > i$, $\phi(T[i+1]) = \phi(T[j])$.

Osherson, Weinstein, and Stob (1986) call such learners **self-monitoring**.

Theorem 3 No self-monitoring learner identifies the class $\mathscr{L}_{fin} - \emptyset$ of non-empty finite languages. (Freivald and Wiehagen, 1979)

Proof: For contradiction, assume that ϕ is a self-monitoring learner that identifies $\mathscr{L}_{fin} - \emptyset$. Consider any finite language $L \in \mathscr{L}_{fin} - \emptyset$ and any text T where $content(T) = L$. Let i be the unique point at which $L(\phi(T[i])) = \emptyset$. Then by assumption, $L(\phi(T[i+1])) = L(\phi(T[m]))$ for all $m > i$. Since $L \in \mathscr{L}_{fin} - \emptyset$, there are infinitely many strings $x \notin L$. Take one such string x, and consider the text $T' = T[i+1]xxx\ldots$. That is, T' is the sequence $T[i+1]$ followed by an infinite sequence of the sentence x. Then $content(T') \in \mathscr{L}_{fin}$. By definition of the self-monitoring ϕ, $L(\phi(T'[i])) = \emptyset$, and so for all $j > i$, $L(\phi(T[i+1])) = L(\phi(T'[i+1])) = L(\phi(T[j])) = L(\phi(T'[j])) = L$. So $L(\phi(T')) \neq content(T')$, and so ϕ does not identify T' – a contradiction! ↯ □

Exercise 1 It is a triviality that no self-monitoring learner identifies any class $\mathscr{L}$ that contains $\emptyset$, since this learner would have to have a unique "aha" point where it takes the value $\emptyset$, and it would also need to take that value at all later points in the text – a contradiction.
So suppose we modify the definition so that a learner ϕ is "self-monitoring" iff

1. on any text T there is a unique i such that $\phi(T[i])$ is undefined, and
2. on any text T if $\phi(T[i])$ is undefined, then for all $j > i$, $\phi(T[i+1]) = \phi(T[j])$.

With this definition, show that there is no self-monitoring learner identifies any class $\mathscr{L}$ that contains $\emptyset$ and another language. ○

Exercise 2 If (total) learner ϕ converges on text $T\ldots$

a. then $\{\phi(T[n]) | n \in \mathbb{N}\}$ is finite. Prove this. Show that the converse is false.

b. then if ϕ identifies T, then $L(\phi(T[n]) = content(T)$ for all but finitely many $n \in \mathbb{N}$. Prove this.
Show that the converse is false.
That is, it is not the case that (if ϕ identifies T, then $L(\phi(T[n]) = content(T)$ for all but finitely many $n \in \mathbb{N}$) implies that ϕ converges on text T.

○

The previous exercise establishes

Corollary 1 No self-monitoring learner identifies the class $\mathscr{L}_{fin}$ of finite languages.

(16) So although the learner ϕ_e cannot tell when it has converged on a text for $L \in \mathscr{L}_{fin}$, no learner that identifies $\mathscr{L}_{fin}$ could do that. The learner ϕ_e is as good as possible in another respect as well: no learner can learn finite languages faster than ϕ_e does.

Definition 1 Suppose ϕ identifies L and let T be a text such that $content(T) = L$. Let the **convergence point** for ϕ in T, $cp(\phi, T)$ be the least i such that for all $j \geq i$, $\phi(T[j]) = \phi(T[i])$.
Given learners ϕ, ϕ' that identify some class $\mathscr{L}$, let's say that ϕ' is **uniformly faster** than ϕ on $\mathscr{L}$ iff

1. for all texts T such that $content(T) \in \mathscr{L}$, $cp(\phi', T) \leq cp(\phi, T)$, and
2. for some text T such that $content(T) \in \mathscr{L}$, $cp(\phi', T) < cp(\phi, T)$.

○

Theorem 4 No learner ϕ that identifies $\mathscr{L}_{fin}$ is uniformly faster than ϕ_e on $\mathscr{L}_{fin}$. (Gold, 1967)

Proof: Suppose for contradiction that there is a learner ϕ which identifies $\mathscr{L}_{fin}$ uniformly faster than ϕ_e. Then there is some text T such that $content(T) \in \mathscr{L}_{fin}$ and $cp(\phi, T) < cp(\phi_e, T)$. So at the point $i = cp(\phi, T)$ where ϕ correctly identifies $content(T)$, ϕ_e has not yet converged. In fact, by the definition of ϕ_e, $content(T[i]) = L(\phi_e(T[i])) \neq L(\phi_e(T)) = L$, since ϕ_e never conjectures two different grammars for the same language. On the other hand, ϕ has converged, that is, $\phi(T(i)) = \phi(T)$ and $L(\phi(T)) = L$. Now consider another text T' such that $T'[i] = T[i]$ and $content(T') = content(T'[i])$. Then $content(T') \in \mathscr{L}_{fin}$ and $cp(\phi_e, T') = i$. Since $T'[i] = T[i]$, we know that $\phi(T'[i]) = \phi(T[i])$ and so $L(\phi(T'[i])) \neq L$. It follows that $cp(\phi, T') > i$, and so ϕ is not uniformly faster than ϕ_e. ↯ □

1.2 Single value languages

(17) In learnability theory, we often think of languages as sets of strings, and texts as sequences of strings, but the previous section shows that many other learning problems will have a structure which is similar in the relevant respects.

(18) Consider the "language" which is a set of pairs of strings:

$$\langle x, y \rangle.$$

For example, if Σ had symbols to represent phonetic structures (PF) and logical forms (LF), we could have a "language" which was a set of PF-LF pairs.

Definition 2 $\mathscr{L}_{sv}$ is the collection of r.e. languages L such that L is a function. That is, a set L of pairs is in $\mathscr{L}_{sv}$ if and only if each first element of a pair is paired with a unique second element. These languages are "single valued."

$\mathscr{L}_{svt}$ is the collection of "single valued" r.e. languages which are "total" in the sense that every string occurs as a first element of a pair. ○

(19) Assuming that the languages in $\mathscr{L}_{svt}$ are r.e. means that each of these languages has a grammar, a grammar in which the expressions derived code pairs of expressions. This coding of pairs can be done in many ways: one way is just to designate a special symbol $*$ (in a vocabulary of size > 2) that separates the left expression s from the right expression t in derived sentences $s * t$.
Furthermore, the grammars generating expressions of this form are enumerable. For example, take our earlier enumeration of all grammars and remove all the grammars that do not conform. (Of course, this is not a computable procedure.)

(20) Notice that the languages in $\mathscr{L}_{svt}$ have the following nice property: if two languages $L, L' \in \mathscr{L}_{svt}$ differ, then there is at least one first element s such that $s * t \in L$ (for some t) but $s * t' \in L'$ where $t \neq t'$. Given this fact about $\mathscr{L}_{svt}$, it is easy to establish the following result:

Theorem 5 $\mathscr{L}_{svt}$ is identifiable.

Proof: Let $\phi(T[i]) = G$ where G is the first grammar in the enumeration such that $content(T[i]) \subseteq L(G)$.

Since each language L in $\mathscr{L}_{svt}$ is represented by a machine that appears at some point in the enumeration, and this function will differ from all others at some point, we are guaranteed that in any text T there is a finite point i at which $\phi(T[i])$ will be the correct guess, and the guess will never change after that. □

This is another kind of "induction-by-enumeration."[2]

[2]It turns out that the collection of all "single valued" languages, $\mathscr{L}_{sv}$, is not identifiable (Osherson, Weinstein, and Stob, 1986, Cor.2.5A). Totality guarantees that the texts will contain, at some finite point, values that distinguish any two functions that differ; in $\mathscr{L}_{sv}$ we have no such guarantee, because the relevant values might be undefined.

1.3 Languages defined by constraints

The learner ϕ_e just keeps a record of what has been seen so far. Since many linguistic theories make use of constraints of various kinds, it is an interesting to consider learners that pay attention to what has not been seen so far. Let's define some simple language classes based on ruling out some of the possible structures, letting $|L|$ represent the number of elements in L:

Definition 3 Let $\mathscr{L}_{co-1}$ be the class of languages $\{\Sigma^* - \{x\} | \ x \in \Sigma^*\}$.
For any number $k \geq 0$, let $\mathscr{L}_{co-k}$ be the class of languages $\{\Sigma^* - L | \ |L| = k\}$.
Let $\mathscr{L}_{co-fin}$ be the class of languages $\{\Sigma^* - L | \ L \in \mathscr{L}_{fin}\}$. ○

Notice that, like $\mathscr{L}_{fin}$, all of these classes are infinite (except for $\mathscr{L}_{co-0}$ which is a singleton set, $\{\Sigma^*\}$). But, unlike $\mathscr{L}_{fin}$, every member of every one of these classes is an infinite language.

A grammar for a language in $\mathscr{L}_{co-1}$ could be regarded as a "filter" represented by a rule of the form

$$*x$$

where x is any string. A grammar for any language in one of the classes $\mathscr{L}_{co-k}$ cold be given by exactly k filters of this kind. And a grammar in $\mathscr{L}_{co-fin}$ would have any number of filters of this kind. That is, $\mathscr{L}_{co-fin} = \bigcup_{k \geq 0} \mathscr{L}_{co-k}$.

Theorem 6 $\mathscr{L}_{co-1}$ is identifiable.

Proof: Consider any enumeration of Σ^*, and let ϕ be defined as follows. For any sequence of strings $T[i]$,

$$\phi(T[i]) = *x$$

where x is the first string in the enumeration that does not occur in $T[i]$.

Now we just need to show that, given any text T for any language $L \in \mathscr{L}_{co-1}$, ϕ identifies T. Since $L \in \mathscr{L}_{co-1}$, L is defined by some filter $*x$. Since there are just finitely many elements that occur before x, there will be some finite point i such that $T[i]$ contains all of those elements. At that point, by the definition of ϕ, $\phi[T(i)] = *x$. Furthermore, it is easy to see that at all later points $j > i$, $\phi(T[j]) = *x$ because x will never occur in the text. □

The previous result generalizes immediately:

Theorem 7 For any k, $\mathscr{L}_{co-k}$ is identifiable.

One learner for this class is the function that, at each point, conjectures that the first k elements of Σ^* not seen so far are the ones excluded.

The previous results do not generalize to $\mathscr{L}_{co-fin}$, though. In fact, we can show:

Conjecture 1 The class $\mathscr{L}_{co-1} \cup \mathscr{L}_{co-2}$ is not identifiable. ○

This claim is intuitive: at each point in a text, the learner has no way of knowing whether one or two elements are excluded. In particular, suppose a learner guessed correctly in some case that the language was defined by $*x$. The problem is that if from there on the text were really one for the co-2 language $*x, *y$, the learner would not see the error. To prove this, though, requires that we show that the learner would never notice the mistake, and that is slightly tricky. To show this, the result in the following section will be extremely useful.

1.4 Exercises

The first 3 of the following exercises are from Osherson, Weinstein, and Stob (1986).

1. Suppose that some particular r.e. language L is given. Show that the class of languages $\{L \cup D | \ D \in \mathscr{L}_{fin}\}$ is learnable.
2. Let $\mathscr{L} = \{L, L'\}$ be a set of r.e. languages such that $L \subset L'$. Prove that no self-monitoring learner identifies $\mathscr{L}$.
3. We saw that $\mathscr{L}_{co-1}$ is identifiable. Show that no self-monitoring learner identifies $\mathscr{L}_{co-1}$.
4. Following Jäger and Rogers (2012), consider any alphabet Σ with 2 or more symbols in it and 2 special "boundary" symbols $\rtimes$ and $\ltimes$ not in Σ to denote the beginning and end of a string respectively. And for any string s, let the k-factors (or n-grams) of s by all substrings of length k in $\rtimes s \ltimes$. So for example,

$$\text{2-factors}(aba) = \{\rtimes a, ab, ba, a \ltimes\}.$$

Now let a k-factor grammar G be a set of k factors, that is, $G \subseteq (\Sigma \cup \{\rtimes, \ltimes\})^k$, with

$$L(G) = \{x \in \Sigma^* | \; k\text{-factors}(x) \subseteq G\}.$$

A k-factor grammar is called k-strictly-local, SL_k.

For example, for the 2-gram grammar $G1 = \{\rtimes a, ab, bb, bc, c\ltimes\}$, it is easy to see that $L(G1) = ab^+c$, so this is an SL_2 language. For any n, let the class of SL_k languages $\mathscr{L}_{SL_k}$ be the set of languages that are generated by SL_k grammars.

a. Show that the class $\mathscr{L}_{SL_2}$ of SL_2 languages includes some finite languages and some infinite ones.

b. Show that the class $\mathscr{L}_{SL_2}$ is finite.

c. Show that the class $\mathscr{L}_{SL_2}$ is learnable.

d. Consider this learner

$$\phi_{\mathrm{SL2}}(T[i]) = \{xy | \; xy \text{ is a 2-factor in } T[i]\}.$$

Show that this learner identifies any SL_2 language in the limit from positive data.

5. Heinz (2010) proposes that long distance phonotactics are patterns definable by what are sometimes called strictly 2-piecewise languages; cf. (Rogers et al., 2009). Let's say that the $xy \in \Sigma^2$ is a 2-piece of a string $s \in \Sigma^*$ iff an occurrence of x strictly precedes an occurrence of y in s. So for example,

$$2\text{-pieces}(aba) = \{ab, aa, ba\}.$$

Let's say that a precedence grammar $G \subseteq \Sigma^2$ and

$$L(G) = \{x \in \Sigma^* | \; 2\text{-pieces}(x) \subseteq G\}.$$

So then every precedence language contains ϵ and the one symbol strings Σ. For example, $L(\emptyset) = \{\epsilon\} \cup \Sigma$ and $L(\{aa\}) = \Sigma \cup a^*$. Now consider the learner

$$\phi_{prec}(T[i]) = \{xy | \; xy \text{ is a 2-piece in } T[i]\}.$$

Show that this learner identifies any precedence language in the limit from positive data.

1.5 Locking sequences

Here we introduce a notion that is very useful for reasoning about how learners behave on texts: the notion of a locking sequence for a learner ϕ on a language L. This is a finite sequence such that no extension of that sequence with elements of L can make ϕ change its mind. That is, it is not merely that none of the following elements in a given text make ϕ change, but no elements of L can make ϕ change. Surprisingly, whenever ϕ identifies L, there is a locking sequence of this kind. We make this idea precise as follows:

Definition 4 Given a learner ϕ that identifies a language L, a sequence t is said to be a **locking sequence for ϕ and L** iff

1. t is a finite sequence of expressions and #,
2. $content(t) \subseteq L$,
3. $L(\phi(t)) = L$, and
4. for every finite sequence s such that $content(ts) \subseteq L$, $\phi(ts) = \phi(t)$.

○

Theorem 8 If L is any (recursively enumerable) language and ϕ identifies L, then there is a locking sequence t for ϕ and L. (Blum and Blum, 1975)

Proof: Let L be any (recursively enumerable) language that ϕ identifies. For contradiction, assume that for every finite t such that $content(t) \subseteq L$ and $L(\phi(t)) = L$, there is a finite sequence s such that $content(ts) \subseteq L$ but $\phi(ts) \neq \phi(t)$.
We will deduce a contradiction by constructing a text R for L on which ϕ does not converge.
Let $U = x_0 x_1 \ldots$ be any text such that $content(U) = L$. Since ϕ identifies U, there is some finite $U[k]$ such that $L(\phi(U[k])) = L$.
We now define a set of sequences $R = \{r^0, r^1, \ldots\}$, where each r^{i+1} extends r^i:

$$r^0 = U[k]$$
$$r^{i+1} \begin{cases} = r^i x_i & \text{if } L(\phi(r^i)) \neq L \\ = r^i s x_i & \text{otherwise, where } s \text{ is finite, } content(s) \subseteq L, \text{ and } \phi(r^i s) \neq \phi(r^i) \end{cases}$$

We know that, at each step, the s mentioned in this definition exists by our assumption. Notice that for all $i < j$, r^i is a prefix of r^j. Also notice that every r^i is at least i elements long. So we can define a text $R = y_1 y_2 \ldots$ as follows: for all $i > 0$, y_i is the i'th element of r^i. Notice that $content(R) = L$, since R contains every element of U and all of the sequences s are such that $content(s) \subseteq L$. However, ϕ does not converge on R, because for every $i \geq 0$, either $L(\phi(R[i])) \neq L$ or for some $j > i$, $\phi(R[i]) \neq \phi(R[j])$. ↯ □

1.6 Exercise

Thanks to Karen Campbell for suggesting this one:

1. Given a learner ϕ that identifies a language L, let's say that a sequence t is a **reverse locking sequence for ϕ and L** iff
   a. t is a finite sequence of expressions and #,
   b. $content(t) \subseteq L$,
   c. $L(\phi(t)) = L$, and
   d. for every finite sequence s such that $content(st) \subseteq L$, $\phi(st) = \phi(t)$.

   OK, here are the questions – prove that your answers are right:
   i. Is every locking sequence for ϕ and L a reverse locking sequence?
   ii. Let's call t a super locking sequence for ϕ and L iff it is both a locking and a reverse locking sequence for ϕ and L. Is it the case that for every ϕ and L there is at least one super locking sequence?

1.7 Languages defined by constraints, part 2

Locking sequences provide exactly what we need to get an easy proof of our earlier Conjecture 1:

Theorem 9 The class $\mathscr{L}_{co-1} \cup \mathscr{L}_{co-2}$ is not identifiable.

Proof: Suppose for contradiction that this class is identifiable. Then there is some learner ϕ that identifies the class. Consider any $L \in \mathscr{L}_{co-1}$. By Theorem 8 there is a locking sequence t for L and ϕ. Since L is infinite and $content(t)$ is finite, there is an $x \in L - content(t)$. Consider any text T' for the language $L' = (L - \{x\}) \in \mathscr{L}_{co-2}$. Since $content(T') \subseteq L$, and since t is a locking sequence for ϕ and L, $\phi(tT')$ converges on a grammar for L, but this is incorrect since $L \neq content(tT') = (L - \{x\})$. So ϕ does not identify this text, so it does not identify $(L - \{x\})$.↯ □

We can immediately draw some more general conclusions, using the general and obvious fact:

Theorem 10 If $\mathscr{L}$ is not identifiable, no superset of $\mathscr{L}$ is identifiable.

Proof: If $\mathscr{L}'$ is identifiable, then there is some ϕ which identifies every language in $\mathscr{L}'$. Hence it identifies every language in $\mathscr{L}$. ↯ □

With this result, we have,

Corollary 2 $\mathscr{L}_{co-fin}$ is not identifiable.

We can now make a number of connections to traditional formal language theory, because of the following simple fact:

Theorem 11 $\mathscr{L}_{co-fin} \subset \mathscr{L}_{fs}$

Proof: This follows immediately from the following: (i) that every finite language is finite state, (ii) Σ^* is finite state, and (iii) if $L_1, L_2 \in \mathscr{L}_{fs}$ then $L_1 - L_2 \in \mathscr{L}_{fs}$.[3] □

Corollary 3 (Gold, 1967)
The class of finite state languages $\mathscr{L}_{fs}$ is not identifiable.
The class of context free languages $\mathscr{L}_{cf}$ is not identifiable.
The class of context sensitive languages $\mathscr{L}_{cs}$ is not identifiable.
The set of recursive languages $\mathscr{L}_{rec}$ is not identifiable.
The set of recursively enumerable languages $\mathscr{L}_{re}$ is not identifiable.

We also have the general result:

Theorem 12 It can happen that the union of two identifiable classes fails to be identifiable.

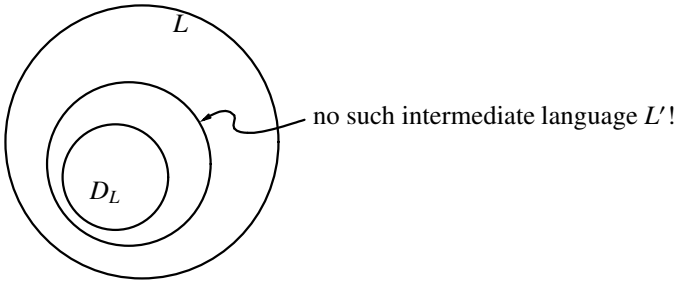
[3] These results are presented in any introductory text on formal language theory, such as Hopcroft and Ullman (1979) or Harrison (1978). We will come back to look at finite state languages and some of the other classes more carefully later.

1.8 Angluin's subset theorem

Various theorems about special cases appeared before Angluin homed in on the following very general "subset theorem."[4]

Theorem 13 Let $\mathscr{L}$ be a collection of (recursively enumerable) languages. Then $\mathscr{L}$ is identifiable iff for all $L \in \mathscr{L}$ there is a finite distinguished subset $D_L \subseteq L$ such that for all $L' \in \mathscr{L}$, if $D_L \subseteq L'$ then $L' \not\subset L$. (Angluin, 1980)

Angluin's theorem tells us that a collection of languages is identifiable just in case, for every language L in the collection, you can find a finite subset D_L such that no language L' that includes D_L is properly included in L:



Proof: ($\Rightarrow$) Suppose $\mathscr{L}$ is identified by ϕ. Then by Theorem 8, for any $L \in \mathscr{L}$, there is a locking sequence t for ϕ and L. Since locking sequences are finite, so is *content*(t). This finite subset of L will distinguish it: that is, we show that for all $L' \in \mathscr{L}$, if *content*$(t) \subseteq L'$, then $L' \not\subset L$.
Assume for contradiction that this is not so, that there is some $L' \in \mathscr{L}$ such that *content*$(t) \subseteq L'$ and $L' \subset L$. Then there is some text ts extending t such that *content*$(ts) = L'$, but since t is a locking sequence for ϕ and L, and since *content*$(ts') \subseteq L' \subset L$, $\phi(ts') = \phi(t)$ for any $s' = s_i,\ i \geq 0$. Thus ϕ fails to identify ts and hence fails to identify L'. ↯

($\Leftarrow$) Assume that for every $L \in \mathscr{L}$, there is a finite $D_L \subseteq L$ such that for all $L' \in \mathscr{L}$, $D_L \subseteq L'$ implies $L' \not\subset L$. Suppose that we have an enumeration of all possible grammars. For each $L \in \mathscr{L}$, we fix a particular finite $D_L \subseteq L$ such that for all $L' \in \mathscr{L}$, if $D_L \subseteq L'$ then $L' \not\subset L$.
Then for every text T, define $\phi(T[i])$ to be the first grammar G in the enumeration such that $D_{L(G)} \subseteq$ *content*$(T[i]) \subseteq L(G)$, if there is one. Otherwise, let $\phi(T[i])$ be the first grammar in the enumeration.
Now we show that this ϕ identifies $\mathscr{L}$. Consider any $L(G) \in \mathscr{L}$ where G is the first grammar in the enumeration that generates $L(G)$, and consider any text T such that *content*$(T) = L(G)$. Since T is a text for $L(G)$, the assumption tells us that there is a finite i such that:
a. $D_{L(G)} \subseteq$ *content*$(T[i])$, and
b. if $G' < G$, $L(G') \in \mathscr{L}$, and $L \not\subset L(G')$, then *content*$(T[i]) \not\subset L(G')$.
We know that there is such an i because, first, $D_{L(G)}$ is finite. And second, the number of $G' < G$ such that $L(G') \in \mathscr{L}$ and $L \not\subset L(G')$ is finite. Since for each such G' there is some string s' such that $s' \notin L(G')$ and $s' \in L$, we can let i be high enough that $T[i]$ contains such a string for each such G'.
It is now easy to see that ϕ will converge on the correct grammar at such a point i. By a and the definition of ϕ, for all $j \geq i$, $\phi(T[j]) = G$ unless there is a $G' < G$ such that $L(G') \in \mathscr{L}$, and $D_{L(G')} \subseteq$ *content*$(T[j]) \subseteq L(G')$.
But clearly, by the choice of i, $D_{L(G)} \subseteq$ *content*$(T[j]) \subseteq L(G)$, and by b, there is no earlier grammar $G' < G$ with these properties, so $\phi(T[j]) = G$. □

This main theorem can be used to establish both positive and negative results of special interest. The following theorem from Gold (1967) was given earlier as an exercise (maybe a rather challenging one), but now it can be proven as an easy corollary to Angluin's theorem.

Corollary 4 Any finite class of languages is identifiable.

Proof: Consider any finite collection of languages $L_1, L_2, \ldots, L_n$, and assume that they are here listed in such a way that if $j < i$, then $L_i \not\subseteq L_j$.
Now, consider each L_i in turn. For each earlier language, $L_j,\ j < i$, let let $x_{i,j}$ be some element that is in L_i but not in L_j. We define $D_{L_i} = \{x_{i,j} |\ j < i\}$.

[4] We follow Osherson, Weinstein, and Stob (1986) in presenting a simple version of Angluin's result which does not establish the existence of a computable learner, but only of a learner. de Jongh and Kanazawa (1996) call this a "watered down" version of Angluin's result, because it characterizes learnability rather than effective learnability. See Angluin (1980) for the original, stronger result, and some extensions in Lange and Zeugmann (1993); de Jongh and Kanazawa (1996).

Obviously, for each of the finitely many languages L_i, the corresponding subset will only have i elements in it – that is, finitely many!
Now suppose that there is some language L_k in our collection such that $D_{L_i} \subseteq L_k$. Clearly in this case, L_k is not one of the languages that appear earlier than L_i in the enumeration, and so it follows by our definition of the enumeration that $L_k \not\subset L_i$. So for each L_i we have a subset D_{L_i} as specified in Angluin's Theorem 13, and so the collection of languages is identifiable. □

Another way to see that any finite collection $L_1, L_2, \ldots, L_n$ is identifiable is to specify an adequate learner directly. Assume again that if $j < i$, then $L_i \not\subseteq L_j$, and assume that these languages are defined by grammars $G_1, G_2, \ldots, G_n$, respectively. Define ϕ so that for any t_i, $\phi(T[i])$ is just the first grammar G such that $content(T[i]) \subseteq L(G)$. This learner identifies the class.

The proof of Corollary 4 actually gives us a more general result:

Corollary 5 Consider any class $\mathscr{L}$ that can be enumerated $L_0, L_1, \ldots$ in such a way that if $j < i$, then $L_i \not\subseteq L_j$. $\mathscr{L}$ is identifiable.

We also derive some interesting negative results:

Corollary 6 Let $\mathscr{L}$ be a collection of (recursively enumerable) languages such that

i. $\mathscr{L}$ contains an infinite language L_∞, and
ii. for every $L_0 \subset L_\infty$ there is some $L \in \mathscr{L}$ such that $L_0 \subseteq L \subset L_\infty$.

Then $\mathscr{L}$ is not identifiable.

Proof: Assume $\mathscr{L}$ satisfies (i) and (ii). By (ii), every finite subset L_0 of L_∞ is such that for some $L \in \mathscr{L}$, $L_0 \subseteq L \subset L_\infty$, and so Theorem 13 applies immediately. □

Corollary 7 No strict superset of $\mathscr{L}_{fin}$ is identifiable. (Gold, 1967)

References

References

Angluin, Dana. 1980. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135.

Blum, L. and M. Blum. 1975. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155.

Freivald, R. and R. Wiehagen. 1979. Inductive inference with additional information. *Elektronische Informationsverarbeitung und Kybernetik*, 15:179–185.

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

Harrison, Michael A. 1978. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, Massachusetts.

Heinz, Jeffrey. 2010. Learning long-distance phonotactics. *Linguistic Inquiry*, 41(4):623–661.

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

Jäger, Gerhard and James Rogers. 2012. Formal language theory: Refining the Chomsky hierarchy. *Philosophical Transactions of the Royal Society B*, 367:1956–1970.

Jain, Sanjay, Daniel Osherson, James S. Royer, and Arun Sharma. 1999. *Systems that Learn: An Introduction to Learning Theory (second edition)*. MIT Press, Cambridge, Massachusetts.

de Jongh, Dick and Makoto Kanazawa. 1996. Angluin's theorem for indexed families of r.e. sets and applications. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory*.

Kanazawa, Makoto. 1998. *Learnable Classes of Categorial Grammmars*. CSLI Publications, Stanford, California.

Laird, Philip D. 1988. *Learning from Good and Bad Data*. Kluwer, Boston.

Lange, Steffen and Thomas Zeugmann. 1993. Language learning in dependence on the space of hypotheses. In *Proceedings of the sixth annual conference on Computational learning theory*, COLT'93, pages 127–136, ACM, NY.

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

Rogers, James, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Visscher, David Wellcome, and Sean Wibel. 2009. On languages piecewise testable in the strict sense. In *Proceedings of the 11th Meeting of the Assocation for Mathematics of Language*.

Chapter 2 Probabilistic criteria for distribution-free learning: PAC

(0) Suppose I have a fair coin that either has heads and tails as usual, or else has 2 heads. You can sample the results of flips of the coin. How many flips will ensure that you can be 99% sure whether the coin has 2 heads? A single tails will definitively answer the question, so all we need to worry about is a sequence of heads that happens even when tails is a possible outcome. We calculate:

$$(\tfrac{1}{2})^n \leq \tfrac{1}{100}$$
$$(\tfrac{1}{2^n}) \leq \tfrac{1}{100}$$
$$2^n \geq 100$$
$$n \geq \log_2 100$$
$$n \geq 6.6439$$
so, $\lceil 6.6439 \rceil = 7$ flips is enough

2.0 PAC learning

(1) The PAC (probably approximately correct) learning framework proposed by Valiant (1984) and developed especially simple perspective on probabilistic learning, with a very natural success criterion that applies for any distribution over the target space
Here we use the definitions from chapter 3 of Anthony and Biggs (1992), which make it clear how the distribution over examples determines both the error ϵ and confidence levels δ.[1]

(2) **Definition:** Consider any distribution μ on X and any target class $\mathscr{L} \subseteq \wp X$. Then for any target $L \in \mathscr{L}$ and any hypothesis $h \subseteq X$, the error of the hypothesis is the measure of the symmetric difference between the hypothesis and the target:

$$\operatorname*{error}_{\mu}(L, h) = \mu((L \backslash h) \cup (h \backslash L)).$$

(3) A distribution μ on X induces a distribution μ^m on X^m obtained by regarding each element of each sequence in X^m as drawn randomly and independently from X according to distribution μ.
For any target $L \subseteq X$, we can define the characteristic function $F_L : X \to \{0, 1\}$

$$F_L(x) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{otherwise.} \end{cases}$$

Let $S(m, L)$ be the set of sequences $x_1, \ldots, x_m \in X^m$ labeled according to whether each element is in L:

$$((x_1, F_L(x_1)), \ldots, (x_m, F_L(x_m))).$$

We consider learners ϕ that are given a data sequence $s \in S(m, L)$, and which output a hypothesis $\phi(s) \subseteq X$, or equivalently $\phi(s) : X \to \{0, 1\}$.

(4) (Some of our learners, given labeled, probabilistic texts, will only adjust their hypotheses in response to positive data.)

(5) **Definition:** $\mathscr{L}$ is PAC-learnable iff there is a learner ϕ such that, for any real numbers $0 < \epsilon, \delta < 1$, there is a positive integer $m_0 = m_0(\delta, \epsilon)$ such that for any target $L \in \mathscr{L}$ and any probability distribution μ on X, whenever $m \geq m_0$,

$$\mu^m\{s \in S(m, L) | \operatorname*{error}_{\mu}(L, \phi(s)) \leq \epsilon\} \geq 1 - \delta.$$

[1] Cf. also Blumer et al. (1989); Anthony and Biggs (1992); Kearns and Vazirani (1994); Pestov (2011).

So a PAC learner converges on the target by getting arbitrarily close, with enough data. Such a learner is 'efficient' iff the time required to compute $\phi(s)$ is bounded by some polynomial function of $(\frac{1}{\epsilon}, \frac{1}{\delta})$ Sometimes, an additional parameter is introduced, a natural measure of the size or dimensionality of the target space, in order to get bounds that are polynomial in $(\frac{1}{\epsilon}, \frac{1}{\delta}, size(L))$.

(6) NB: Sometimes the learner is given not only samples but also δ, ϵ, in other cases this is not necessary.

2.1 Learning finite sets of finite sets

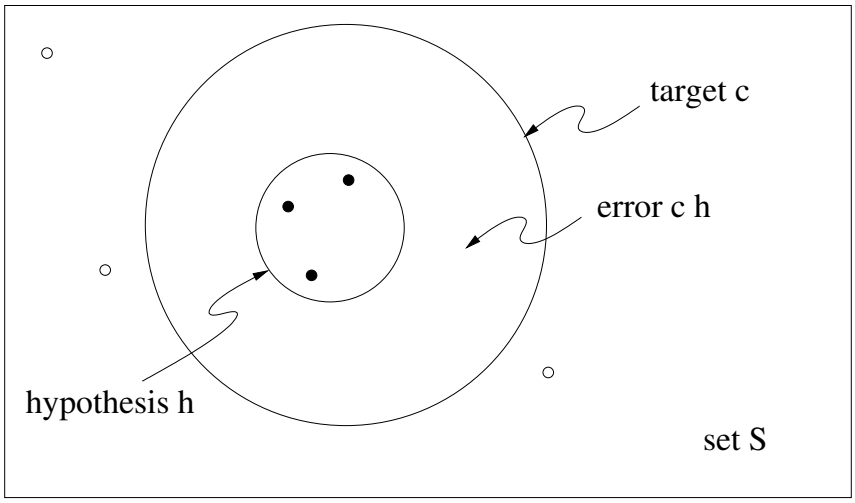
Let's start with the simplest kind of case – learning a finite set from a collection of finite sets. Recall that $\wp(S)$ is the powerset of S, that is, the set of all subsets of S. (The following result and proof is very slightly simpler than Kearns and Vazirani (1994, Thm1.2), and uses the same style of proof which comes also from Valiant (1984).)

(7) We will use the "union bound", which says that for any events A, B and any distribution μ, the probability that one or the other will happen is less than or equal to the sum of the probability of A and the probability of B.

$$\mu(A \cup B) \leq \mu(A) + \mu(B).$$

This is an equality when A and B are disjoint, but the inequality holds in all cases.

(8) **Thm.** Given any finite set S, the class $\wp(S)$ is PAC-learnable.



Showing positive samples • labeled 1 and negative samples ◦ labeled 0

Proof. Define a learner ϕ as follows. Given any target concept $c \in \wp(S)$ and any labeled sample s of length k, the learner's hypothesis $\phi(s) = \{x \in S \mid (x, 1) \in s\}$. This learner simply remembers all positive instances, never generalizing.

So by this definition of ϕ, the learner's hypothesis h is always a subset of c. Consequently, $error_\mu(c, h) = \mu(c - h) = \sum_{x \in c-h} \mu(x)$.

Since $|c| \leq |S|$ and so obviously $|c - h| \leq |S|$. If for all $x \in (c - h)$, $\mu(x) \leq \frac{\epsilon}{|S|}$, then $\mu(c - h) \leq \epsilon$.

Let's say that a set x is "important" if $\mu(x) > \frac{\epsilon}{|S|}$. Then if no $x \in (c - h)$ is important, $\mu(c - h) \leq \epsilon$.

For any *fixed* important element x, the probability of a sample of length m in which x does not occur is $\leq (1 - \frac{\epsilon}{|S|})^m$. Since there are at most $|S|$ important elements, by the union bound, the probability that a sample of length m lacks all the important elements is $\leq |S|(1 - \frac{\epsilon}{|S|})^m$.

Recalling the fact mentioned at the bottom of page 40 that for positive x, $(1 - x)^{m_0} \leq e^{-xm_0}$, and of course $e^{\log k} = k$, notice that if we let $m_0 = \frac{|S|}{\epsilon} \log \frac{|S|}{\delta}$ then

$$\begin{aligned} |S|(1 - \tfrac{\epsilon}{|S|})^{m_0} \quad \leq \quad |S|e^{-m_0 \frac{\epsilon}{|S|}} \quad &= \quad |S|e^{-(\frac{|S|}{\epsilon} \log \frac{|S|}{\delta})\frac{\epsilon}{|S|}} \\ &= \quad |S|e^{-\log \frac{|S|}{\delta}} \\ &= \quad |S|\tfrac{\delta}{|S|} \\ &= \quad \delta \end{aligned}$$

So for $m \geq m_0$, we have $|S|(1 - \frac{\epsilon}{|S|})^m \leq \delta$, so that $\mu(error(h) \geq \epsilon) \leq \delta$.

(9) Since any binary relation on a finite set is also a finite set, it immediately follows that when the instance space is the set of pairs $X = C \times C$, the class $\wp(C \times C)$ is PAC learnable.

(10) **Exercise.** Generalize Thm (8) to show that any finite set of hypotheses is PAC learnable, whether the hypotheses themselves are finite or not. (If you're stuck or lazy, you can google to find this proof, e.g. Shai Ben-David's proof here https://cs.uwaterloo.ca/~shai/Chapters_4_CS886.pdf)

2.2 Learning rays

Anthony and Biggs (1992, §3.3) begin with this simple example.

(11) A **ray** is a set of real numbers greater than some arbitrary point θ. So we can think of a ray as a set

$$r_\theta = \{x \in \mathbb{R} | \ \theta \leq x\}$$

or as a function (the 'characteristic function' of this set) $r_\theta : \mathbb{R} \to \{0, 1\}$:

$$r_\theta(x) = \begin{cases} 1 & \text{if } \theta \leq x \\ 0 & \text{otherwise} \end{cases}$$

(12) Let's add an "empty ray" r_∞,

$$r_\infty(x) = 0 \text{ for all } x \in \mathbb{R}.$$

(13) The whole set of rays $\mathscr{L} = \{r_\theta | \ \theta \in \mathbb{R} \cup \{\infty\}\}$.

(14) Consider any μ on $X = \mathbb{R}$ and the target class of rays $\mathscr{L}$.
Obviously, this target class is bigger than any we have considered before!

(15) **NB:** We cannot learn a ray by memorizing the positive instances seen so far. That strategy "overfits" terribly! We need a much more expressive class of hypotheses with some kind of bias towards rays. (Cf. Gold learners for $\mathscr{L}_{fin}$ vs $\mathscr{L}_{co-1}$.)

(16) Define the learning function which for any sample $T[m]$ of length m has this value

$$\phi_{ray}((x_1, r_\theta(x_1)), \ldots, (x_m, r_\theta(x_m))) = \begin{cases} = r_h & \text{for } h = \min\{x_i | \ r_\theta(x_i) = 1\} \text{ if this set is nonempty} \\ = r_\infty & \text{otherwise.} \end{cases}$$

The situation $\lambda \leq \beta_0$ is illustrated in Figure 3.1.

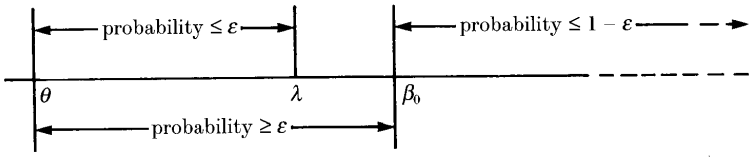


Figure 3.1: In this situation the error of the output is at most ϵ

(17) **Thm 3.3.1.** The set of rays $\mathscr{L}$ is PAC learnable.

Proof: Suppose $\delta, \epsilon, r_\theta, \mu$ are given, and suppose we have a sample $s \in S(m, r_\theta)$.

a. Let $\beta_0 = \sup\{\beta | \ \mu[\theta, \beta) < \epsilon\}$. Remember sup=lub, so consider all the β's that make $[\theta, \beta) < \epsilon$, then among the values larger than all of these, let β_0 be the least. In other words, we set β_0 to the maximum value such that we still have

$$\mu[\theta, \beta_0) \leq \epsilon.$$

And obviously then

$$\mu[\theta, \beta_0] \geq \epsilon.$$

b. By this definition, if $h \leq \beta_0$,

$$\operatorname*{error}_{\mu}(r_h) = \mu[\theta, h) \leq \mu[\theta, \beta_0) \leq \epsilon.$$

c. Since $\mu[\theta, \beta_0] \geq \epsilon$, the probability of a data point not in this interval is $\leq 1 - \epsilon$.

d. So the probability of selecting m data points not in this interval is $\leq (1 - \epsilon)^m$.

e. By the definition of ϕ, h is the least value seen in the data, so the probability that $h \leq \beta_0$ will be $\geq 1 - (1 - \epsilon)^m$

f. Since $h \leq \beta_0$ implies $\text{error}_\mu(r_h) \leq \epsilon$,

$$\mu^m\{s \in S(m, r_\theta) | \operatorname*{error}_{\mu}(r_h) \leq \epsilon\} \geq 1 - (1 - \epsilon)^m.$$

Since we want to find the conditions under which being within this ϵ error has probability greater than $1 - \delta$, what we want is conditions under which $(1 - \epsilon)^m \leq \delta$.

g. So if we let $m = \lceil \frac{1}{\epsilon} \log \frac{1}{\delta} \rceil$ then (using the two facts mentioned at the bottom of page 40)

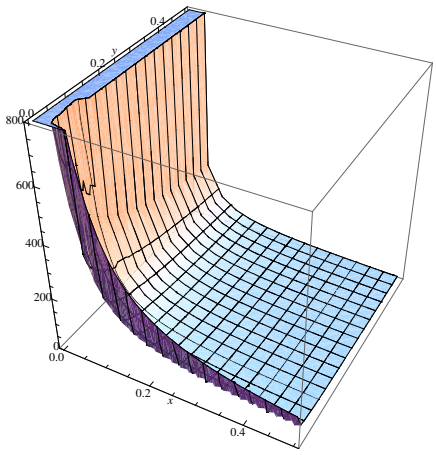
$$(1 - \epsilon)^m < e^{-\epsilon m} < e^{\log \delta} = \delta.$$

Since for any ϵ, δ, this m will suffice, ϕ is a PAC learner. □

(18) We can calculate exactly how many samples are needed for us to be 99% sure that at most 1% of our samples will be misclassified by our hypothesis:

```
> e
ans =  2.7183
> log(e)
ans =  1
> ceil(1.2)
ans =  2
> epsilon=0.99
epsilon =  0.99000
> delta=0.99
delta =  0.99000
> ceil ( (1/(1-epsilon)) * log (1/(1-delta)) )
ans =  461
>
```

We can get a quick overall picture from a plot of the required number of samples against $1 - \epsilon$ and $1 - \delta$:



But, for example, when we fix ϵ and just watch how the number of required samples increases as we halve δ (or vice versa), we see that the increase is not 'too rapid' –

```
> epsilon=0.99
epsilon =  0.99000
```

```
> delta=0.92
delta =  0.92000
> ceil ( (1/(1-epsilon)) * log (1/(1-delta)) )
ans =  253
> delta=0.96
delta =  0.96000
> ceil ( (1/(1-epsilon)) * log (1/(1-delta)) )
ans =  322
> delta=0.98
delta =  0.98000
> ceil ( (1/(1-epsilon)) * log (1/(1-delta)) )
ans =  392
> delta=0.99
delta =  0.99000
> ceil ( (1/(1-epsilon)) * log (1/(1-delta)) )
ans =  461
> delta=0.995
delta =  0.99500
> ceil ( (1/(1-epsilon)) * log (1/(1-delta)) )
ans =  530
> delta=0.9975
delta =  0.99750
> ceil ( (1/(1-epsilon)) * log (1/(1-delta)) )
ans =  600
```

2.3 Learning rectangles

(19) Following Kearns and Vazirani (1994, §1.1) and Blumer et al. (1989), consider the problem of learning rectangles that are aligned with the X and Y axes on the Cartesian plane $\mathbb{R} \times \mathbb{R}$.

That is, recalling the definition of closed intervals, the concepts are the products of closed intervals, so

$$C = \{[x_1, x_2] \times [y_1, y_2] | \; x_1, x_2, y_1, y_2 \in \mathbb{R}\} \subseteq \mathbb{R}^2.$$

(nb: this C is not closed under unions or complements, but it is closed under intersections)

(20) **Theorem:** The rectangles C are efficiently PAC learnable.

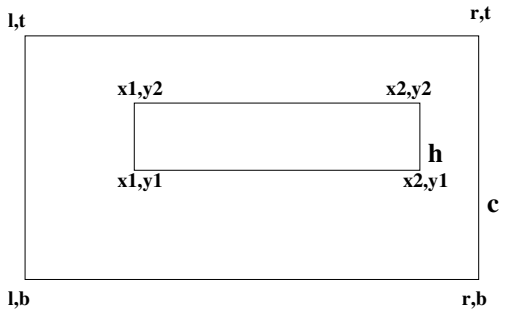
Proof: Let μ be an arbitrary probability measure on $\mathbb{R}^2$, and let $c \in C$ be an arbitrary rectangle.

For any sequence of i labeled samples $T[i] = \langle (v_1, b_1), (v_2, b_2), \ldots, (v_i, b_i) \rangle$ from $\mathbb{R}^2 \times 2$. Note that in this case each v_i is a vector in $\mathbb{R}^2$, a pair of real numbers, and that some elements of the sample might be positive, examples v_i where $b_i = 1$. So consider the corresponding set of positive examples, $P[i] = \{v_i | \langle v_i, 1 \rangle \in T[i]\}$. Now define

$$\phi(T[i]) \begin{cases} = \emptyset & \text{if } P[i] = \emptyset \quad \text{(no positive examples yet)} \\ = [x_1, x_2] \times [y_1, y_2] & \text{otherwise, where } x_1 = \min\{x_k | \, \langle x_k, y_k \rangle \in P[i]\}, \\ & \qquad\qquad x_2 = \max\{x_k | \, \langle x_k, y_k \rangle \in P[i]\}, \\ & \qquad\qquad y_1 = \min\{y_k | \, \langle x_k, y_k \rangle \in P[i]\}, \text{ and} \\ & \qquad\qquad y_2 = \max\{y_k | \, \langle x_k, y_k \rangle \in P[i]\} \end{cases}$$

For $c \in C$, we said the "error" of a given hypothesis h is $error_\mu(c, h) = \mu((c - h) \cup (h - c))$.

However, this learner never "overgeneralizes", so at every point i, $h \subseteq c$, and so $error_\mu(c, h) = \mu(c - h)$.



When $c = [l, r] \times [t, b]$ and $h = [x_1, x_2] \times [y_1, y_2]$, the set $c - h$ is the union of the following 4 sets (which overlap at the corners, just to make the math easier):

$$\begin{aligned} top = \quad & [l, r] \times [y_2, t] \\ bottom = \quad & [l, r] \times [b, y_1] \\ left = \quad & [l, x_1] \times [b, t] \\ right = \quad & [x_2, r] \times [b, t] \end{aligned}$$

Now if we can guarantee that the probability of a sample from any one of these four areas is at most $\epsilon/4$, then the probability of a point from the union of the four areas is at most ϵ.

And in this case, the probability that m independent draws miss each rectangle is at most $(1 - \epsilon/4)^m$, and so the probability that m draws miss all 4 rectangles is $4(1 - \epsilon/4)^m$.

So how large does m need to be to guarantee that $4(1 - \epsilon/4)^m \leq \delta$?

Using the fact (mentioned in the review on page 39) that

$$1 - x \leq e^{-x},$$

we see that

$$1 - \frac{\epsilon}{4} \leq e^{-\frac{\epsilon}{4}},$$

and so

$$(1 - \frac{\epsilon}{4})^m \leq e^{-m\frac{\epsilon}{4}},$$

and so m is large enough if

$$4(e^{-m\frac{\epsilon}{4}}) \leq \delta.$$

Now, dividing by 4

$$e^{-m\frac{\epsilon}{4}} \leq \frac{\delta}{4},$$

and taking the $\log_e$ of both sides

$$-m\frac{\epsilon}{4} \leq \log(\frac{\delta}{4}).$$

Multiplying by -1

$$m\frac{\epsilon}{4} \geq \log(\frac{4}{\delta}),$$

and then by $\frac{4}{\epsilon}$,

$$m \geq \frac{4}{\epsilon} \log(\frac{4}{\delta}).$$

So if we have at least $\frac{4}{\epsilon} \log(\frac{4}{\delta})$ samples, then with probability $1 - \delta$, the probability that the next sample will be an error is no more than ϵ.

These sample requirements are "efficient" in the sense that the number of samples needed is polynomial w.r.t. $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$.

(21) Remarkably, the previous results says that if we want to be at least 99% sure ($\delta = 0.01$) that our hypothesis has at least a 99% chance of being right on the next sample, it suffices to look at

$$\lceil (4/0.01) * \log(4/0.01) = 2396.6 \rceil = 2397 \text{ samples.}$$

(22) Looking more carefully at how many examples would be required to be 99% sure that our hypothesis is X% right,

confidence	⌊samples⌋
50%	831
60%	921
70%	1036
80%	1198
90%	1475
95%	1752
97.5%	2030
98.75%	2307
99.375%	2584
99.687%	2861

(23) **Exercise.**

a. Define a similar learner for axis-aligned rectangular solids in $\mathbb{R}^3$ and assess whether it is PAC.

b. Is the similar learner for axis-aligned hyper-rectangles in $\mathbb{R}^n$ PAC?

2.4 VC dimension

(24) Given $S \subseteq X$, if

$$\{L \cap S \mid L \in \mathscr{L}\} = \wp(S),$$

then S is **shattered** by $\mathscr{L}$.
That is, every subset of S is an element of the pointwise restriction of some L to S.

(25) The Vapnik-Chervonenkis dimension of $\mathscr{L}$,

$$VC(\mathscr{L}) = \max\{|S| : \ S \text{ is shattered by } \mathscr{L}\}.$$

That is, the VC dimension of $\mathscr{L}$ is the size of the largest set that $\mathscr{L}$ shatters.
If arbitrarily large sets can be shattered by $\mathscr{L}$, then $VC(\mathscr{L}) = \infty$.

(26) It is fairly easy to see these facts – try to sketch the proofs:

$VC(\mathscr{L}_{ray}) = 1$	
$VC(\mathscr{L}_{rect}) = 4$	see the proof sketch Kearns and Vazirani (1994, p.53)
$VC(\mathscr{L}_{lin\frac{1}{2}\mathbb{R}^n}) = n + 1$	(see if you can do the case $n = 2$)
$VC(\mathscr{L}_{\text{convex hulls in } \mathbb{R}^2}) = \infty$	(consider shattering circles)
$VC(\mathscr{L}_{\wp(\mathbb{R}\times\mathbb{R})}) = \infty$	
$VC(\mathscr{L}_{fin}) = \infty$	
$VC(\mathscr{L}_{reg}) = \infty$	
$VC(\mathscr{L}_{rev}) = \infty$	

(27) **Exercise.** Note that in $\mathbb{R}^1$, the linear half spaces correspond to the union of $\mathscr{L}_{ray}$ with the 'negative rays' $\mathscr{L}_{nray}$, where negative ray nr is determined by a maximum element θ:

$$nr_\theta = \{x \in \mathbb{R} \mid x \leq \theta\}.$$

Sketch a proof of the claim that $VC(\mathscr{L}_{ray} \cup \mathscr{L}_{nray}) = 2$.

(28) **Exercise.** Show that for any concept class such that $|\mathscr{L}| = n$ (for finite n), $VC(\mathscr{L}) \leq \log_2 n$

(29) It is not an accident that all the classes $\mathscr{L}$ for which we provided PAC learners are also ones with finite VC dimension. It is possible to show:
Thm. $\mathscr{L}$ is PAC learnable iff it has finite VC dimension (Blumer et al., 1989; Pestov, 2011).
This PAC result is based on the more general idea described by Vapnik (1998, §4.9.3). Many approaches to learning have success criteria characterized by VC-dimension (Poggio et al., 2004).

(30) Developing observations in Blumer et al. (1989), Takács (2007) shows that the VC dimension of convex n-gons in the plane is finite, extending the result to similar finite bounds for convex polytopes in higher dimensions (Takács and Pataki, 2007). But it is easy to see that the dimension of the whole collection of convex subsets of the plane is infinite. Consider for example n distinct points on a circle: the convex hull enclosing any subset of those points will exclude the others. See the nice discussion of this result in the text Alon, Spencer, and Erdös (1992, §14.4).

(31) The VC dimension of OT with k constraints, fixed in advance, is $k - 1$ (Riggle, 2009; Heinz, Kobele, and Riggle, 2009). Similarly for the VC dimension of harmonic grammars with finitely many constraints fixed in advance (Bane, Riggle, and Sonderegger, 2010).

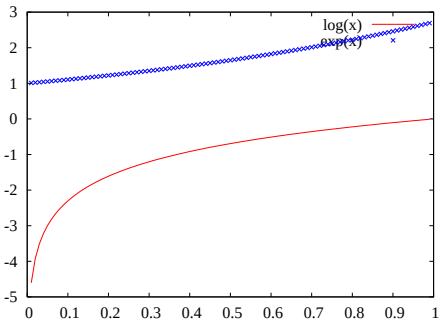
(32) While the set of regular languages has infinite VC dimension, if we consider canonical acceptors with at most n states with a fixed alphabet Σ, since this class is finite, the class of languages accepted by these machines is also finite. For example, if $|\Sigma| = 2$, then the number of different machines $\leq n^{2n} \cdot 2^{n^2}$, and so the VC dimension of this class of languages is $O(n \log n)$ (de la Higuera, 2010, §6.1.2).

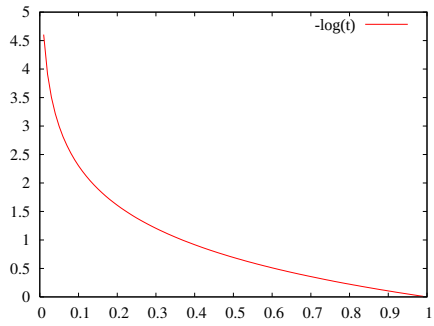
(33) Angluin (1987) shows that an arbitrary regular language can be learned from membership and equivalence queries, but she notes that in many real applications a learner cannot expect answers to equivalence queries. But she shows that if, instead of equivalence queries, the learner samples sentences from Σ^* according to some unknown distribution μ, then the probability of being within ϵ of the target language is at least $1 - \delta$ after $O(n + (1\epsilon)(n \log(1/\delta) + n^2))$ samples.

2.5 *Appendix:* exp and log review

$$k^m \cdot k^n = k^{m+n}$$
$$k^0 = 1$$
$$k^{-n} = \frac{1}{k^n}$$
$$\frac{a^m}{a^n} = a^{m-n}$$
$$\log_k x = y \text{ iff } k^y = x$$
$$\log_k(k^x) = x \text{ since: } k^x = k^x$$
and so: $$\log_k k = 1$$
and: $$\log_k 1 = 0$$
$$\log_k(\frac{M}{N}) = \log_k M - \log_k N$$
$$\log_k(MN) = \log_k M + \log_k N$$
so, in general: $$\log_k(M^p) = p \cdot \log_k M$$
and we will use: $$\log_k \frac{1}{x} = \log_k x^{-1} = -1 \cdot \log_k x = -\log_k x$$

E.g. $512 = 2^9$ and so $\log_2 512 = 9$. And $\log_{10} 3000 = 3.48 = 10^3 \cdot 10^{0.48}$. And $5^{-2} = \frac{1}{25}$, so $\log_5 \frac{1}{25} = -2$



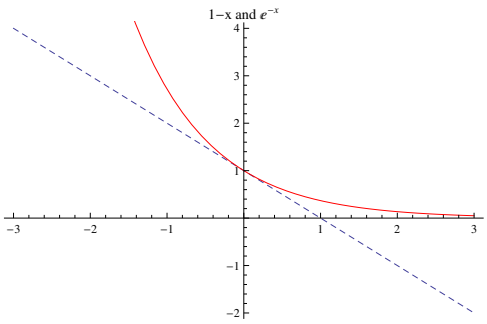


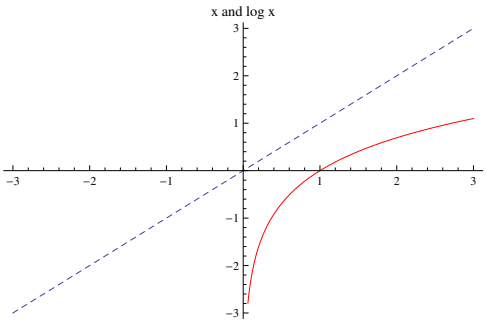
surprisal as a function of $p(A)$: $-\log p(A)$

The function $\log_2$ is often used to caculate"bits" of information, units of "surprise," but another common choice is $\log_e$. I usually write log for $\log_e$, but some people write ln. The precise value of e is defined in various ways which will not really matter to us here

$$e = \lim_{x \to 0}(1+x)^{\frac{1}{x}} = \sum_{n=0}^{\infty} \frac{1}{n!} \approx 2.7182818284590452$$

More commonly, e is defined as the x such that a unit area is found under the curve $\frac{1}{u}$ from $u = 1$ to $u = x$, that is, it is the positive root x of $\int_1^x \frac{1}{u} du = 1$. In general: $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$. Euler also discovered $e^{\pi\sqrt{-1}} + 1 = 0$ Maor (1994); Graham, Knuth, and Patashnik (1989).





We will use the fact that $1 - x \leq e^{-x}$, as displayed in the graph above left, and more generally $(1-x)^m \leq e^{-xm}$. We will also use $x > \log x$, as displayed in the graph above right. Cf. e.g. Graham, Knuth, and Patashnik (1989); Knuth (1973).

2.6 *Appendix:* probability review

The basics of probability can be found in many texts. For gentle introductions, begining with the axioms as I do here, see for example Jacod and Protter (2000) or Resnick (1999). More serious introductory presentations are given in Rosenthal (2006) and Durrett (1996), for example. These are texts I have used, but there are many excellent texts.

(34) A set $I \subset \mathbb{R}$ is an **interval** iff $x \leq y \leq z$ and $x, z \in I$ implies $y \in I$.

If I has an upper bound then $I \subseteq (-\infty, b]$ where $b = \bigvee I$ and $(-\infty, b] = \{x \in \mathbb{R} |\ x \leq b\}$

If I also has a lower bound then $I \subseteq [a, b]$ where $a = \bigwedge I$ and $[a, b]$ is the **closed interval** $\{x \in \mathbb{R} |\ a \leq x \leq b\}$.

(a, b) is the **open interval** $\{x \in \mathbb{R} |\ a < x < b\}$

(35) A **measure space** $(\Omega, \mathcal{F}, p)$, where

1. $\mathcal{F}$ is a collection of subsets of Ω containing Ω, closed under countable unions and complementation (and hence also under intersections).

   Such an $\mathcal{F}$ is a σ**-field**.

   When $(\Omega, \mathcal{F})$ is a σ-field which is also a topology, it is called a **Borel** field.

2a. $p : \mathcal{F} \to [0, \infty]$

2b. (**countable additivity**) for any disjoint sets $A_0, A_1, \ldots \in \mathcal{F}$,

$$p(\bigcup_{i=0}^{\infty} A_i) = \sum_{i=0}^{\infty} p(A_i)$$

(36) A **sample space** Ω is a set of **outcomes**.

Then an **event** A is an element of $\mathcal{F}$, that is $A \subseteq \Omega$.

(37) Given events A, B, the usual notions $A \cap B, A \cup B$ apply.

For the complement of A let's write $\overline{A} = \Omega - A$.

Sometimes set subtraction $A - B = A \cap \overline{B}$ is written $A \backslash B$, but since we are not using the dash for complements, we can use it for subtraction without excessive ambiguity.

(38) When $A \cap B = \emptyset$, A and B are **disjoint.**

$A_0, A_1, \ldots, A_n$ is a sequence of **disjoint** events iff
for all $0 \leq i, j \leq n$ where $i \neq j$, the pair A_i and A_j are disjoint.

(39) **A probability space** is a measure space $(\Omega, \mathcal{F}, p)$, where P satisfies the 3 Kolmogorov axioms:

a. $p : \mathcal{F} \to [0, 1]$

b. $p(\Omega) = 1$

c. (**countable additivity**) for any disjoint sets $A_0, A_1, \ldots \in \mathcal{F}$,

$$p(\bigcup_{i=0}^{\infty} A_i) = \sum_{i=0}^{\infty} p(A_i)$$

$p : \mathcal{F} \to [0, 1]$ is then called a **probability measure** or **distribution**.

(40) Typically the relevant domain $\mathcal{F}$ over Ω is assumed to be $\wp(\Omega)$.

Since this is the case, when Ω is no bigger than $\mathbb{N}$, it is common to identify $p(\{x\})$ with $p(x)$ and just think of P as a function $p : \Omega \to [0, 1]$ where with the probabilities of subsets of Ω determined by the axioms.

We will sometimes do this below.

(41) **Theorem:** $p(\overline{A}) = 1 - p(A)$

Proof: Obviously $\overline{A}$ and A are disjoint, so by axiom c, $p(A \cup \overline{A}) = p(\overline{A}) + p(A)$

Since $\overline{A} \cup A = \Omega$, axiom b tells us that $p(\overline{A}) + p(A) = 1$ □

(42) **Theorem:** $p(\emptyset) = 0$

(43) **Theorem:** $p(A \cup B) = p(A) + p(B) - p(A \cap B)$

Proof: Since A is the union of disjoint events $(A \cap B)$ and $(\overline{B} \cap A)$, $p(A) = p(A \cap B) + (\overline{B} \cap A)$.

Since B is the union of disjoint events $(A \cap B)$ and $(\overline{A} \cap B)$, $p(B) = p(A \cap B) + (\overline{A} \cap B)$.

And finally, since $(A \cup B)$ is the union of disjoint events $(\overline{B} \cap A)$, $(A \cap B)$ and $(\overline{A} \cap B)$, $p(A \cup B) = p(\overline{B} \cap A) + p(A \cap B) + p(\overline{A} \cap B)$.

Now we can calculate $p(A) + p(B) = p(A \cap B) + (\overline{B} \cap A) + p(A \cap B) + (\overline{A} \cap B)$, and so $p(A \cup B) = p(A) + p(B) - p(A \cap B)$. □

(44) **Theorem (Boole's inequality):** $p(\bigcup_0^\infty A_i) \leq \sum_0^\infty p(A_i)$

(45) **Exercises**

a. Prove that if $A \subseteq B$ then $p(A) \leq p(B)$

b. In (43), we see what $p(A \cup B)$ is. What is $p(A \cup B \cup C)$?

c. Prove Boole's inequality.

(46) The **conditional probability of** A **given** B**,** $p(A|B) =_{df} \frac{p(A \cap B)}{p(B)}$

(47) **Bayes' theorem:** $p(A|B) = \frac{p(A)p(B|A)}{p(B)}$

Proof: From the definition of conditional probability just stated in (46), (i) $p(A \cap B) = p(B)p(A|B)$. The definition of conditional probability (46) also tells us $p(B|A) = \frac{p(A \cap B)}{p(A)}$, and so (ii) $p(A \cap B) = p(A)p(B|A)$. Given (i) and (ii), we know $p(A)p(B|A) = p(B)p(A|B)$, from which the theorem follows immediately. □

(48) A and B are **independent** iff $p(A \cap B) = p(A)p(B)$.

(49) A **random (or stochastic) variable (or process)** on Ω is a function $X : \Omega \to \mathbb{R}$.

(50) The range of X is sometimes called the **sample space of the stochastic variable** X**,** Ω_X**.**

(51) The **expectation** $E(X)$ of a discrete random variable X

$$E(X) = \sum_{x=-\infty}^{\infty} xp(X = x).$$

2.6.1 Borel-Cantelli lemma

This theorem is often called a lemma because it sets the stage for a number of important results. It is covered in all four of the texts mentioned at the beginning of this appendix, and is mentioned in some of the learnability literature. We may go over this if we rely on it later.

(52) For any sequence of independent events $\langle A_0, A_1, \ldots \rangle$

let the **limit supremum** be $\limsup_{n\to\infty} A_n = \bigcap_{n=1}^{\infty} \bigcup_{m=n}^{\infty} A_m$.

This event is the collection of the set of points that are members of infinitely many of the A_n, and so so it is sometimes written (A_n *i.o.*) where "i.o." stands for "infinitely often." Kallenberg (1997)

The **limit infimum** of the sequence is $\liminf_{n\to\infty} A_n = \bigcup_{n=1}^{\infty} \bigcap_{m=n}^{\infty} A_m$

Intuitively, the limit infimum of a sequence of sets is the event containing the points that happen in all but finitely many A_i. That is, at some finite point in the sequence, all later events in the sequence have the points in the limit infimum, so it is sometimes written (A_n *ult.*) where "ult." stands for "ultimately."

(53) For example, let's find the limit supremum of sequence of events

$$A_1 = \{0\}, A_2 = \{0, 1\}, A_3 = \{0, 1, 2\}, \ldots$$

When $n = 1$, we have $\bigcap \bigcup_1^\infty A_m$, which is $A_1 = \{0\}$.

When $n = 2$, clearly $\bigcap \bigcup_2^\infty A_m = A_2$, and the limit of this sequence is $\bigcup_{n=1}^{\infty} A_n = \mathbb{N}$.

Calculating the limit infimum, when $n = 1$, $\bigcup_{n=1}^{\infty} \bigcap_{m=1}^{\infty} A_m = A_1$.

When $n = 2$, clearly $\bigcup_{n=1}^{\infty} \bigcap_{m=2}^{\infty} A_m = A_2$, and the limit of this sequence is $\bigcup_{n=1}^{\infty} A_n = \mathbb{N}$.

The limit supremum and infimum coincide like this whenever the sequence is monotonic.

(54) **Theorem** If $\langle A_0, A_1, \ldots \rangle$ is increasing, then $\limsup_{n\to\infty} A_n = \liminf_{n\to\infty} A_n = \bigcup_{n=1}^{\infty} A_n$.

In this case we say the sequence converges to $\lim_{n\to\infty} A_n$.

If $\langle A_0, A_1, \ldots \rangle$ is decreasing, then $\limsup_{n\to\infty} A_n = \liminf_{n\to\infty} A_n = \bigcap_{n=1}^{\infty} A_n = \lim_{n\to\infty} A_n$.

(55) **(Borel lemma)** For any sequence of events $\langle E_0, E_1, \ldots \rangle \in \mathcal{F}$,

$$\sum_{n=1}^{\infty} p(E_n) < \infty \quad \text{implies} \quad p(\bigcap_{m=1}^{\infty}(\bigcup_{n=m}^{\infty} E_n)) = 0.$$

Proof: Since for each m, $A \subseteq \bigcup_{n=m}^{\infty} A_n$, it follows from the monotonicity and additivity of probability measures:

$$p(A) \leq P(\bigcup_{n=m}^{\infty}) \leq \sum_{n=m}^{\infty} p(A_n).$$

Since $\sum_n p(A_n) \leq \infty$, the right side approaches 0 as $m \to \infty$, so $p(A) = 0$. □

(56) **(Borel-Cantelli lemma)** For any sequence of independent events $\langle E_0, E_1, \ldots \rangle \in \mathcal{F}$,

$$\sum_{n=1}^{\infty} p(E_n) < \infty \quad \text{iff} \quad p(\bigcap_{m=1}^{\infty}(\bigcup_{n=m}^{\infty} E_n)) = 0.$$

I think we have now done enough to understand what this lemma says. The proof of the lemma is not trivial though, and so I will omit it. It is provided in (Port, 1994, Prop47.6), (Kallenberg, 1997, Thm2.18), (Brémaud, 1999, Lemma 8.1), and is carefully introduced but then left as an exercise in (Fristedt and Gray, 1997, §6.2).

(57) A sequence $\{Z_i | \, i \geq 1\}$ **converges p-almost surely (p-a.s.)** to random variable Z iff $p(\lim_{n\uparrow\infty} Z_n = Z) = 1$.

(58) **Kolmogorov's Strong Law of Large Numbers (SLLN):** Let $\{X_i | \, i \geq 1\}$ be an i.i.d. (independent and identically distributed) sequence of random variables such that $E(|X_i|) < \infty$. Then, p-a.s.

$$\lim_{n\uparrow\infty} \frac{X_1 + \ldots + X_n}{n} = E(X_i).$$

References

References

Alon, Noga, Joel H. Spencer, and Paul Erdös. 1992. *The Probabilistic Method*. Wiley, NY.

Angluin, Dana. 1987. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106.

Anthony, Martin and Norman Biggs. 1992. *Computational Learning Theory*. Cambridge University Press, NY.

Bane, Max, Jason Riggle, and Morgan Sonderegger. 2010. The VC dimension of constraint-based grammars. *Lingua*, 120(5):1194 – 1208.

Blumer, Anselm, Andrzei Ehrenfeucht, David Haussler, and Manfred K. Warmuth. 1989. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36:929–965.

Brémaud, Pierre. 1999. *Markov Chains: Gibbs Fields, Monte Carlo Simulations, and Queues*. Springer, NY.

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

Durrett, Rick. 1996. *Probability: Theory and Examples*. Duxbury, Boston.

Fristedt, Bert and Lawrence Gray. 1997. *A Modern Approach to Probability Theory*. Birkhäuser, Boston.

Graham, Ronald L., Donald E. Knuth, and Oren Patashnik. 1989. *Concrete Mathematics*. Addison-Wesley, Menlo Park, California.

Heinz, Jeffrey, Gregory M. Kobele, and Jason Riggle. 2009. Evaluating the complexity of optimality theory. *Linguistic Inquiry*, 40(2):277–288.

Jacod, Jean and Philip Protter. 2000. *Probability Essentials*. Springer, NY.

Kallenberg, Olav. 1997. *Foundations of Modern Probability*. Probability and its Applications. Springer, NY.

Kearns, Michael J. and Umesh V. Vazirani. 1994. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, Massachusetts.

Knuth, Donald E. 1973. *The Art of Computer Programming: Volume 1, Fundamental Algorithms*. Addison-Wesley, Menlo Park, California.

Maor, Eli. 1994. *e: The Story of a Number*. Princeton University Press, Princeton.

Pestov, Vladimir. 2011. PAC learnability versus VC dimension: a footnote to a basic result of statistical learning. *ArXiv*. http://arxiv.org/abs/1104.2097.

Poggio, Tomaso, Ryan Rifkin, Partha Niyogi, and Sayan Mukherjee. 2004. General conditions for predictivity in learning theory. *Nature*, 428:419–422.

Port, Sidney C. 1994. *Theoretical Probability for Applications*. John Wiley and Sons, NY.

Resnick, Sidney. 1999. *A Probability Path*. Birkhäuser, Boston.

Riggle, Jason. 2009. The complexity of ranking hypotheses in optimality theory. *Computational Linguistics*, 35(1).

Rosenthal, Jeffrey S. 2006. *A First Look at Rigorous Probability Theory*. World Scientific, Hackensack, New Jersey.

Takács, Gábor. 2007. The Vapnik-Chervonenkis dimension of convex n-gon classifiers. *Hungarian Electronic Journal of Sciences*.

Takács, Gábor. 2010. *Convex polyhedron learning and its applications*. Ph.D. thesis, Budapest University of Technology and Economics, Faculty of Electrical Engineering and Informatics, Budapest, Hungary.

Takács, Gábor and Béla Pataki. 2007. Lower bounds on the vapnik-chervonenkis dimension of convex polytope classifiers. In *Proceedings of the 11th International Conference on Intelligent Engineering Systems, INES 2007*.

Taylor, J. C. 1997. *An Introduction to Measure and Probability Theory*. Springer, NY.

Valiant, Leslie G. 1984. A theory of the learnable. *Communications of the Association for Computing Machinery*, 27(11):1134–1142.

Vapnik, Vladimir N. 1998. *Statistical Learning Theory*. Wiley, NY.

Chapter 3 *k*-reversible languages

For Gold, while it is impossible to learn an arbitrary regular language from examples (where by "learn" we mean "identify from positive text"), there are infinite subsets of the regular languages which are learnable. We already saw that one such subset is $\mathscr{L}_{fin}$, but all those languages are finite – not very interesting. For PAC-learners, even the class $\mathscr{L}_{fin}$ is not identifiable. Angluin (1982) found a much more interesting collection of subsets, subsets in which some of the languages can have a really interesting structure, containing both finite and infinite languages: for every $k \geq 0$, the k-reversible languages are learnable. She presents a Gold learner which, given evidence like `abc, adc`, conclude that `b` and `c` have similar syntactic roles. And given `ab, abc`, the learner will conclude that `c` can be iterated any number of times. After presenting Angluin's learner, we will consider this problem from a PAC perspective.

3.0 Finite state (i.e. regular) languages

Finite systems, systems that can only have finitely many (computationally relevant) states, can recognize infinite languages, but only if, in recognizing any string, only a finite amount of information needs to be remembered at each point. They play an important role in recent computational phonology.

A language can be recognized with finite memory iff it can be defined with a rewrite grammar in which all the rules have one of the following forms:

$C \rightarrow \epsilon$	(where C is any category and ϵ is the empty sequence)
$C \rightarrow a\ D$	(where C, D are any categories and a is any (terminal) vocabulary element)

For example, the following grammar which defines $\{a, b\}^*$ has this form:

$$S \rightarrow \epsilon \qquad S \rightarrow aS \qquad S \rightarrow bS$$

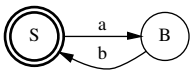
And the following grammar defines $(ab)^*$:

$$S \rightarrow \epsilon \qquad S \rightarrow aB \qquad B \rightarrow bS$$

These grammars branch only to the right. (It turns out that languages defined by these grammars can also be defined with grammars that branch only to the left.)

3.0.1 A simple representation of finite machines

Grammars of the form shown above can be regarded as specifications of finite machines that can recognize (or generate) the language defined by the grammar. We just think of the categories as states, the non-empty productions as rules for going from one state to another, and the empty productions specify the final states. The machine corresponding to the grammar above can be represented by the following graph, where the initial states are indicated by a bold circle and the final states are indicated by the double circles:



This kind of machine is usually formalized with the following 5 parts. (Here we follow the fairly standard presentation of Perrin (1990) fairly closely.)

Definition 5 A *finite automaton* $A = \langle Q, \Sigma, \delta, I, F \rangle$ where

Q is a finite set of states ($\neq \emptyset$);

Σ is a finite set of symbols ($\neq \emptyset$);

δ $\subseteq Q \times \Sigma \times Q$,

I $\subseteq Q$, the initial states;

F $\subseteq Q$, the final states.

○

Definition 6 A *path* is a sequence $c = (q_i, a_i, q_{i+1})_{1 \leq i \leq n}$ of transitions in δ. In any such path, q_1 is its *origin*, q_{n+1} its *end*, the sequence $a_1 a_2 \ldots a_n$ is its *label*, and n is its length.
We add the case of a length 0 path from each state to itself, labeled by the empty string 1.
To indicate that there is a a path from q_1 to q_{n+1} labeled with a sequence $a_1 a_2 \ldots a_n$ we will sometimes write $(q_1, a_1 a_2 \ldots a_n, q_{n+1}) \in \delta$. ○

NB: We have defined finite automata in such a way that every transition is labeled with an alphabet symbol. Since there is a 0-step path labeled ϵ going from every state to itself, to define a language that contains ϵ, we simply let $F \cap I \neq \emptyset$.

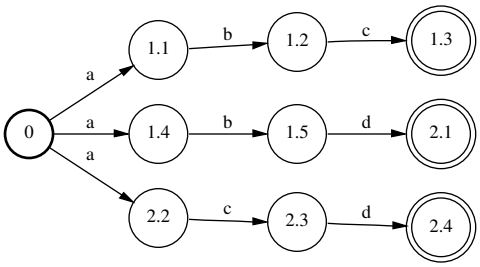
We could allow ϵ to label paths that change state, with only a slight change in our definitions. For any set S, let $S^\epsilon = S \cup \{\epsilon\}$. Then we revise our definition of finite automata just by letting $\delta \subseteq Q \times \Sigma^\epsilon \times Q$. Given such an automaton, the ϵ transitions can be eliminated without changing the language accepted just by equating all states that are related by ϵ transitions.

Definition 7 A path is *successful* if its origin is in I and its end is in F.
The language $L(A)$ accepted by the automaton A is the set of labels of successful paths. ○

3.0.2 Some basic results about finite machines

Definition 8 A language $L \subseteq \Sigma^*$ is *regular* (*finite state, recognizable*) iff for some finite automaton A, $L = L(A)$. ○

Clearly, every finite language is regular. Given a set like $\{abc, abd, acd\}$ we can construct a trivial finite automaton like this:



For any finite language L we can define an acceptor like this. This language L is obviously not "minimal" – that is, it has more states than necessary. One simple step for reducing states involves sharing common prefixes.

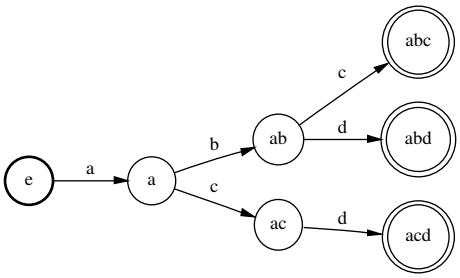
Definition 9 We define the *prefixes of L*, $Pr(L) = \{u |$ for some v, $uv \in L\}$ ○

Definition 10 For finite L, the *prefix tree acceptor for L*, $PT(L) = \langle Q, \Sigma, \delta, I, F \rangle$ where

$Q =$	$Pr(L)$;
Σ	is a finite set of symbols ($\neq \emptyset$);
$(w, a, wa) \in \delta$ iff	$w, wa \in Q$,
$I =$	$\{\epsilon\}$;
$F =$	L.

○

Example 1 $PT(\{abc, abd, acd\})$ is smaller than the acceptor shown above, but accepts exactly the same language: ○

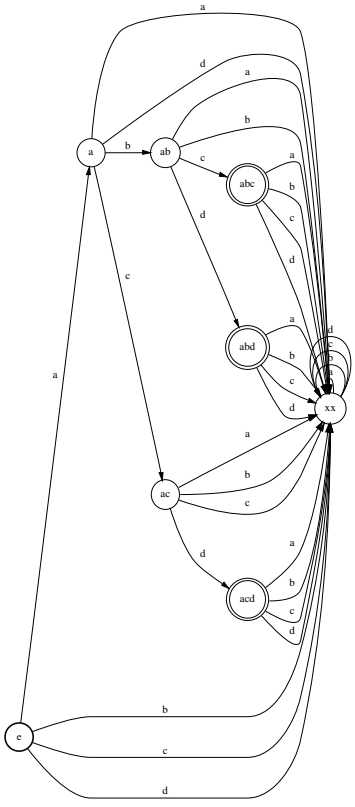


3.0.3 Deterministic finite machines

Definition 11 A finite automaton is *complete* iff for every $q \in Q,\ a \in \Sigma$ there is at least one $q' \in Q$ such that $(q, a, q') \in \delta$. ○

For any automaton there is a complete automaton that accepts the same language. We simply add arcs that go to a "dead" state – a state from which there is no path to a final state.

For example, $PT(\{abc, abd, acd\})$ is not complete, but the following automaton is, and accepts the same language:



Definition 12 A *deterministic finite automaton (DFA)* is a finite automaton where where δ is a function $\delta : (Q \times \Sigma) \to Q$ and I has at most one element.
When a deterministic automaton has a path from p_1 to p_n labeled by $a_1 a_2 \ldots a_n$ we will sometimes write $\delta(p_1, a_1 a_2 \ldots a_n) = p_n$.
○

(A DFA can be represented by a $Q \times \Sigma$ matrix.)

Theorem 14 (Myhill) A language is accepted by a DFA iff it is accepted by a finite automaton.

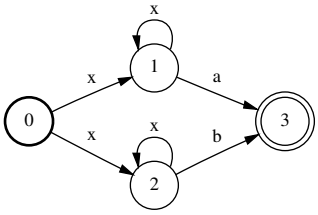
We use $\wp(S)$ to indicate the powerset of S, that is, the set of all subsets of S. The powerset of a set S is sometimes also represented by 2^S, but we will use $\wp(S)$. (Note, for example, that 2^n in the theorem just below refers to a number, not to a set of sets.)

Proof: Given NFA A=$\langle Q, \Sigma, \delta, I, F\rangle$ define
DFA=$\langle \wp(Q), \Sigma, \delta', \{I\}, \{s \in \wp(Q) | \ s \cap F \neq \emptyset\}\rangle$ where

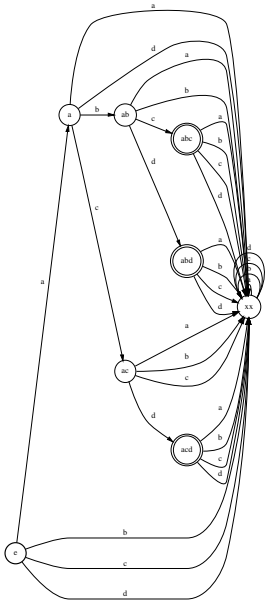
$$(q'_i, a, q'_j) \in \delta' \text{ iff } q'_j = \{q_j | \ (q_i, a, q_j) \in \delta \text{ and } q_i \in q'_i\}$$

The proof that this DFA is equivalent is an easy induction: see for example Hopcroft and Ullman (1979, Thm 2.1) or Lewis and Papadimitriou (1981, Thm 2.3.1) □

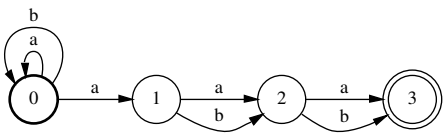
Example 2 The automata shown above are all deterministic. The following automaton is not:



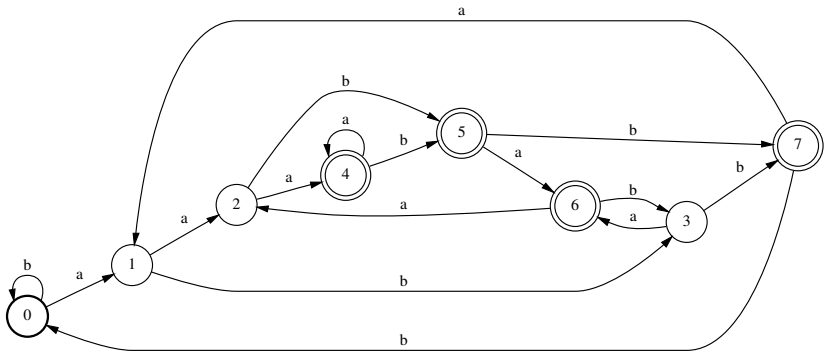
We can use the "subset construction" of the previous theorem to make this machine deterministic:



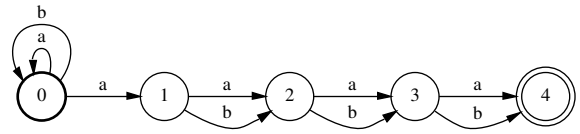
This machine is larger than the original. In fact, for some nondeterministic machines, the smallest deterministic equivalent machines can be much larger. Perrin (1990, p30) considers as an example $\{a, b\}^* a \{a, b\}^n$. When $n = 2$ this language is accepted by the following 4 state nondeterministic automaton:

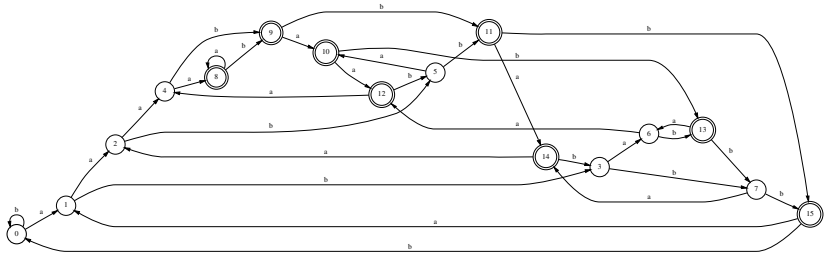


The corresponding deterministic automaton is this one:



Adding one state to the nondeterministic automaton, we find that its minimal deterministic equivalent doubles in size:





Theorem 15 There are n-state automata A such that the smallest DFA accepting $L(A)$ has at least 2^n states.

Since DFAs have nice computational properties, there are studies of how to compute them without unnecessary blowup in size (Leslie, 1995).

3.0.4 The Myhill-Nerode theorem and the canonical acceptor $A_{\equiv_L}$

For finite languages L, PT(L) is not generally the minimal deterministic automaton accepting L. That is, it is not the DFA accepting L with the smallest number of states. However, it is fairly easy to construct a minimal DFA for any regular language using the equivalence classes of the Nerode equivalence relation (sometimes called the right congruence relation induced by L). These equivalence relations also give us a characterization of the finite state languages.

Definition 13 The *Nerode equivalence relation for L*, $x \equiv_L y$ iff for all $z \in \Sigma^*$, $xz \in L$ iff $yz \in L$. ○

Theorem 16 If $w \in L$ and $w \equiv_L w'$ then $w' \in L$.

Proof: By definition, letting $z = \epsilon$. □

Lemma 1 If $a \in \Sigma$ and $w \equiv_L w'$ then $wa \equiv_L w'a$.

Proof: Assume $a \in \Sigma$, $w \in \Sigma^*$ and $w \equiv_L w'$. By definition, for any $x \in \Sigma^*$,

$wx \in L$ iff $w'x \in L$ So let $x = az$:

$w(az) \in L$ iff $w'(az) \in L$ But then

$(wa)z \in L$ iff $(w'a)z \in L$ and so $wa \equiv_L w'a$.

□

Definition 14 Given any equivalence relation $\equiv$, the *equivalence class of* w is $[w]_{\equiv} = \{x |\ w \equiv x\}$.
(Often we use just the brackets, leaving off the subscript when no confusion will result.)
The *index of equivalence relation* $\equiv$, $I(\equiv)$ is the number of different equivalence classes it induces, $I(\equiv) = |\{[x] | x \in \Sigma^*\}|$. ○

Theorem 17 (Myhill-Nerode Theorem) For any language L, $\equiv_L$ has finite index iff L is regular.[1]

Proof: ($\Leftarrow$) Since every regular language is accepted by some DFA $A = \langle Q, \Sigma, \delta, \{q_0\}, F\rangle$, assume $L = L(A)$. For any $x, y \in \Sigma^*$, let $x \equiv_A y$ just in case $\delta(q_0, x), \delta(q_0, y)$ are defined and $\delta(q_0, x) = \delta(q_0, y)$. Obviously, $\equiv_A$ is an equivalence relation, and its index cannot be larger than $|Q|$. But if $x \equiv_A y$ then for all z, $xz \equiv_A yz$, and so $xz \in L$ iff $yz \in L$. Since $\equiv_A$ need only be defined for prefixes of L, while $\equiv_L$ is defined for all of Σ^*, let $\equiv'_L$ be the restriction of $\equiv_L$ to the domain of $\equiv_A$; in $\equiv_L$ the non-prefixes of L are all in 1 block, with the empty set of "good finals." By the definition of the Nerode equivalence relation, if $x \equiv_A y$ then $x \equiv'_L y$. It follows that the index $I(\equiv'_L) \leq I(\equiv_A)$, and hence even if $\equiv_L$ has one more block than $\equiv'_L$ (the non-prefixes, if any), both $I(\equiv'_L)$ and $I(\equiv_L) + 1$ are finite.
($\Rightarrow$) Assume $\equiv_L$ has finite index. We define *the canonical acceptor for* L, $A_{\equiv_L}$. We let equivalence classes themselves be the states of the automaton, $Q = \{[w] |\ w \in Pr(L)\}$. So, by assumption, Q is finite. Let

$$\delta([w], a) = \{[wa]\} \quad \text{whenever } w, wa \in Pr(L),$$

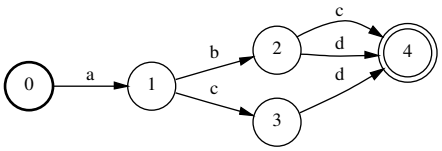
$$F = \{[w] |\ w \in L\}, \text{ and}$$

$$I = \{[\epsilon]\}.$$

Now it is clear that $A_{\equiv_L} = \langle Q, \Sigma, \delta, I, F\rangle$ is a deterministic automaton which accepts L, since by definition $w \in L(A_{\equiv_L})$ iff $[w] \in F$ iff $w \in L$. □

Example 3 The canonical acceptor for $\{abc, abd, acd\}$ is smaller than $PT(\{abc, abd, acd\})$. In fact, it is this:

○



Corollary 8 $L = \{a^n b^n |\ n \in \mathbb{N}\}$ is not regular.

Proof: Obviously, for each choice of n, $[a^n] \neq [a^{n+1}]$, and so $\equiv_L$ does not have finite index. □

Corollary 9 For any regular language L, the canonical acceptor $A_{\equiv_L}$ has $I(\equiv_L) - 1$ states if there is any string $w \notin Pr(L)$, and otherwise has $I(\equiv_L)$ states.

Proof: Every equivalence class of $\equiv_L$ is a state of $A_{\equiv_L}$ except for the class of strings that are not prefixes of any sentences of L, if there are any. □

Corollary 10 No DFA accepting L has fewer states than $A_{\equiv_L}$.

Proof: This is already implicit in the proof of the Myhill-Nerode theorem. Compare the machine $A_{\equiv_L}$ with states Q to any arbitrary deterministic $A' = \langle Q', \Sigma, \delta', \{q'_0\}, F'\rangle$, where $L = L(A')$. We show that there must be at least as many states in Q' as in Q.
Define: $x \equiv_{A'} y$ iff $\delta'(q'_0, x) = \delta'(q'_0, y)$. Since A' is deterministic and the values of δ' are in Q', $|Q'| \geq I(\equiv_{A'}) - 1$ – that is, $\equiv_{A'}$ only distinguish as many classes as there are states of Q', plus one other class if some strings are not in $Pr(L)$.
But notice that we also have, as in the Myhill-Nerode proof, $x \equiv_{A'} y$ implies $x \equiv_L y$.
(This is the key point! No machine accepting L can equate strings x, y that are not equated by $\equiv_L$!)
That is, $I(\equiv_{A'}) \geq I(\equiv_L)$. It follows then that $|Q'| \geq |Q|$. □

Corollary 11 Any minimal DFA $A = \langle Q', \Sigma, \delta', \{q'_0\}, F'\rangle$ accepting L is isomorphic to $A_{\equiv_L}$, that is, there is a bijection $g : Q \to Q'$ such that $g(\delta(q, a)) = \delta'(g(q), a)$.

[1] The Myhill-Nerode theorem is treated in Hopcroft and Ullman (1979, §3.4) at the end of their second chapter on finite automata. It is treated in Moll, Arbib, and Kfoury (1988, §8.2). In Lewis and Papadimitriou (1981), the Myhill-Nerode theorem is an exercise.

Note: There is an efficient algorithm for converting any deterministic machine accepting L into a minimal deterministic machine accepting L.[2]

Also notice that the previous theorem and its proof rely on the determinism of the automaton that is being compared to $A_{\equiv_L}$. In fact, we can get much smaller machines if we allow nondeterminism.

3.0.5 Grammatical representations of regular languages

Definition 15 A *rewrite grammar* $G = \langle V, \Sigma, P, S \rangle$ where

- V is a finite set of symbols ($\neq \emptyset$)
- $\Sigma \subseteq V$, the terminal symbols;
- $P \subseteq V^*(V - \Sigma)V^* \times V^*$;
- $S \in (V - \Sigma)$.

○

An element $\langle u, v \rangle \in P$ is often written $u \to v$.

Definition 16 For $u, w, x, y \in V^*$, $uxw \Rightarrow uyw$ iff $x \to y$ is in P.
$\Rightarrow^*$ is the reflexive, transitive closure of $\Rightarrow$. ○

Definition 17 The *language generated by grammar G,* $L(G) = \{w \in \Sigma^* | \, S \Rightarrow^* w\}$ ○

Definition 18 Given a grammar G, the sequence $w_0, w_1, \ldots, w_n$ is a *derivation of* w_n *from* w_1 iff $w_i \Rightarrow w_{i+1}$ for all $0 \leq i < n$. If $w_0 = S$, this is a *derivation of w from G*. ○

We generalize the grammar form of the introduction just slightly, to allow single terminals as well as the empty string on the right sides of productions:

Definition 19 G is *right linear* iff every production in P has one of the following forms, where $a \in (\Sigma \cup \{\epsilon\})$, $A, B \in (V - \Sigma)$:

$$\begin{aligned} A &\to aB \\ A &\to a \end{aligned}$$

○

Lemma 2 If a language $L \subseteq \Sigma^*$ is accepted by automaton A, then it is generated by a right linear grammar.

Proof: Let $L = L(A)$, where $A = \langle Q, \Sigma, \delta, I, F \rangle$. We assume that $Q \cap \Sigma = \emptyset$ without loss of generality, since this can always be achieved by renaming the states. We want to associate all of the states in I with the start symbol of the grammar, so let

$$h(q) = \begin{cases} S & \text{if } q \in I \\ q & \text{otherwise} \end{cases}$$

Now we define a right linear grammar $G = \langle V, \Sigma, P, S \rangle$ as follows:

(i) $V = \Sigma \cup (Q - I) \cup \{S\}$.

(ii) Define P as the smallest set such that

(a) if $\delta(q_i, a)$ contains q_j then $(h(q_i) \to ah(q_j)) \in P$;

(b) if $\delta(q_i, a)$ contains q_j and $q_j \in F$ then $(h(q_i) \to a) \in P$.

(c) if $I \cap F \neq \emptyset$ then $(S \to \epsilon) \in P$.

[2] Cf. Algorithm 4.5 of Aho, Hopcroft, and Ullman (1974, pp158,162); Watson (1993).

Call this grammar the right linear equivalent of the A. Now we must show that $w \in L(A)$ iff $w \in L(G)$.
($\Rightarrow$) By definition, for each $w \in L(A)$ there is at least one path $(q_0, a_1, q_1), (q_1, a_2, q1) \ldots, (q_{n-1}, a_n, q_n)$ such that $w = a_1 \ldots a_n$. If the path has length 0 and $w = \epsilon$, then $S \rightarrow \epsilon$ is in P by (c), allowing the corresponding derivation from G. In any other case, we show that the following is a derivation of w from G:

$$\langle h(q0),\ a_1 h(q_1),\ a_1 a_2 h(q_2),\ \ldots,\ a_1 a_2 \ldots a_{n-1} h(q_{n-1}),\ a_1 a_2 \ldots a_{n-1} a_n \rangle.$$

Consider the beginning and end of the sequence first. By the definitions of h and of accepting state sequences, $h(q_0) = S$. By the same definitions q_n is a final state and so we have the production $h(q_{n-1}) \rightarrow a_n$ in the grammar defined above. As for the middle of the derivation, the definition of accepting state sequence guarantees that for $0 \leq i < n$, $q_{i+1} \in \delta(q_i, a_{i+1})$, and so our grammar has productions $h(q_i) \rightarrow a_{i+1} h(q_{i+1})$.
($\Leftarrow$) We must show that each $w \in L(G)$ is accepted by A. The correspondence between derivations and accepting sequences can be used again to show this. □

Lemma 3 If L is generated by a right linear grammar, then L is accepted by a A.

Proof: ($\Rightarrow$) Suppose L is generated by the right linear grammar $G = \langle V, \Sigma, P, S \rangle$. Define A as follows:

$$\begin{array}{lll} Q = & (V - \Sigma) \cup \{q_f\}, & \\ I = & S, & \\ F = & \{q_f\}, & \\ \delta(A, a) = & \begin{cases} \{B | (A \rightarrow aB) \in P\} & \text{if } P \text{ has no rule of the form } A \rightarrow a \\ \{q_f\} \cup \{B | (A \rightarrow aB) \in P\} & \text{otherwise.} \end{cases} & \end{array}$$

Call this automaton A the equivalent of right linear G. It is now easy to show a correspondence between derivations and accepting state sequences as was done in the previous proof. □

Theorem 18 L is accepted by a finite automaton A iff L is generated by a right linear grammar.

Proof: Immediate from the previous 2 lemmas. □

3.0.6 The pumping lemma for regular languages

Theorem 19 If $x \in L(A)$ and $|x| \geq |Q|$ then for some $u, v, w \in \Sigma^*,\ x = uvw,\ |v| > 0$ and for all $n \geq 0$, $uv^n w \in L(A)$.

Proof: Assume $x \in L(A)$, $|x| \geq |Q|$. Then there is a successful path

$$(q_0, a_1, q_1), (q_1, a_2, q1) \ldots, (q_{n-1}, a_n, q_n)$$

where $x = a_1 \ldots a_n$. In particular, $q_0 \in I,\ q_n \in F, a_1 \ldots a_n = x$ and $n \geq |x|$. Since $|x| \geq |Q|,\ n \geq |Q|$, and so there are some $q_i, q_j,\ 0 \leq i < j \leq n$ such that $q_i = q_j$ and $|a_{i+1} \ldots a_j| > 0$. Let

$$\begin{array}{ll} u = & a_1 \ldots a_i, \\ v = & a_{i+1} \ldots a_j, \\ w = & a_{j+1} \ldots a_n \end{array}$$

We noted already that $|v| > 0$. The string $uvw \in L(A)$ by assumption, but we now show that for all $n \geq 0,\ uv^n w \in L(A)$.
So there is a successful path

$$(q_0, a_1, q_1), \ldots, (q_{i-1}, a_i, q_i), \ldots, (q_j, a_j, q_{j+1}), \ldots, (q_{n-1}, a_n, q_n),$$

such that $q_i = q_j$. So instead of going from q_{i-1} to q_i we can go from q_{i-1} to q_j. It follows that

$$(q_0, a_1, q_1), \ldots, (q_{i-1}, a_i, q_j), (q_j, a_j, q_{j+1}), \ldots, (q_{n-1}, a_n, q_n)$$

is a successful path. Consequently, $uv^0 w \in L(A)$. (For any string v, $v^0 = \epsilon$.)
Furthermore, instead of going from q_{j-1} to q_j, we can just as well go back into q_i to repeat the sequences $\langle q_i, \ldots, q_{j-1} \rangle$ and $\langle a_{i+1}, \ldots, a_j \rangle$ any number of times. Consequently, $uv^n w \in L(A)$ for all $n \geq 0$. □

3.0.7 Regular languages are closed under union

Given two finite state machines, we can easily construct a finite state machine that accepts the union of the two languages.

Given $A_1 = \langle Q_1, \Sigma_1, \delta_1, I_1, F_1\rangle$ and $A_2 = \langle Q_2, \Sigma_2, \delta_2, I_2, F_2\rangle$, we can assume without loss of generality that $Q_1 \cap Q_2 = \emptyset$. Then define

$$A = \langle Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta_1 \cup \delta_2, I_1 \cup I_2, F_1 \cup F_2\rangle.$$

It is easy to show that this automaton accepts exactly the language $L(A_1) \cup L(A_2)$.

3.0.8 Regular languages are closed under intersection

Given two finite state machines, we can easily construct a finite state machine that accepts the intersection of the two languages.

Given $A_1 = \langle Q_1, \Sigma, \delta_1, I_1, F_1\rangle$ and $A_2 = \langle Q_2, \Sigma, \delta_2, I_2, F_2\rangle$, define $A = \langle Q_1 \times Q_2, \Sigma, \delta, I_1 \times I_2, F_1 \times F_2\rangle$, where for all $a \in \Sigma, q_1,\ r_1 \in Q_1,\ q_2, r_2 \in Q_2$,

$$([q_1, q_2], a, [r_1, r_2]) \in \delta \text{ iff } (q_1, a, r_1) \in \delta_1 \text{ and } (q_2, a, r_2) \in \delta_2.$$

It is easy to show that this automaton accepts exactly the language $L(A_1) \cap L(A_2)$.

3.0.9 Regular languages are closed under concatenation

Given two finite state machines, we can easily construct a finite state machine that accepts the concatenation of the two languages.

Given $A_1 = \langle Q_1, \Sigma, \delta_1, I_1, F_1\rangle$ and $A_2 = \langle Q_2, \Sigma, \delta_2, I_2, F_2\rangle$, intuitively, we merge all the elements of F_1 with all the elements of I_2, so that δ maps an input a_1 and an element q_1 of F_1 to everything that δ maps it to, together with each q_2 that δ_2 maps an initial state to.

3.0.10 Regular languages are closed under complements

Given a finite state machine A that accepts $L(A) \subseteq \Sigma^*$, we can easily construct a finite state machine that accepts $\Sigma^* - L(A)$. Intuitively, we determinize A and then enrich it so that every element of Σ can be accepted from every state, if only to map the state to a "dead" state from which no final state can be reached. Then, we construct a new machine which is like the first except that it has as final states all the states that are non-final in the previous machine.

3.0.11 Regular languages are closed under reversal

For any string $s = a_1 a_2 \ldots a_n$, the reversal $s^r = a_n \ldots a_2 a_1$. As expected, $\epsilon^r = \epsilon$.

For any language $L \subseteq \Sigma^*$, the reversal $L^r = \{s^r |\ s \in L\}$.

Given a finite state machine $A = \langle Q, \Sigma, \delta, I, F\rangle$ and that accepts $L(A) \subseteq \Sigma^*$, we can easily construct a finite state machine that accepts $L(A)^r$:

$$A^r = \langle Q, \Sigma, \delta^r, F, I\rangle \quad \text{where} \quad (q_0, a, q) \in \delta^r \text{ iff } (q, a, q_0) \in \delta.$$

Notice that the initial states I of A are the final states of A^r and the final states F of A are the initial states of A^r.

3.0.12 Exercises

1. Draw the minimal deterministic automaton that accepts

$$\{C\mathsf{V}, C\mathsf{V}C, \mathsf{V}C, \mathsf{V}\}(.\{C\mathsf{V}, C\mathsf{V}C, \mathsf{V}C, \mathsf{V}\})^*$$

2. Draw the minimal deterministic transducer which maps a sequence $w \in \{C, \mathsf{V}, .\}^*$ to x^n iff w contains n occurrences of $.C$
3. Intersect the domain of the previous transducer with the language defined in the first exercise, and draw the result.
4. Use Nerode's theorem to show that $\{xx |\ x \in \{a, b\}^*\}$ is not regular.

3.1 Gold learners for k-reversible languages

A deterministic finite automaton is one in which, from any state, given any next input symbol, at most one transition can be taken. That is, the transition relation is a function of its first two arguments, $\delta : (Q \times \Sigma) \to Q$, and the set of initial states I has at most one element.

In a deterministic automaton, if $s \in \Sigma^*$ labels any path from a given state, it labels exactly one path from that state. We generalize this notion to consider machines that are deterministic with more than just one symbol of "lookahead."

Definition 20 For any $k \geq 0$, a string $s \in \Sigma^*$ of length $k = |s|$ is a **k-follower** of state q iff $\delta(q, s) \neq \emptyset$.

An automaton A is **deterministic with lookahead** k iff for any distinct pair of states $q_1, q_2 \in I$ or $q_1, q_2 \in \delta(q, a)$ for some $q \in Q, a \in \Sigma$, no string is a k-follower of both q_1 and q_2.

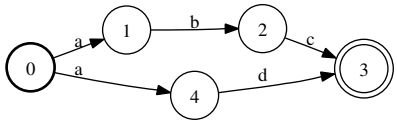
An automaton A is k-**reversible** iff A is deterministic and A^r is deterministic with lookahead k.

A language L is k-**reversible** iff it is the language accepted by some k-reversible automaton.

$\mathscr{L}_{k-rev}$ is the set of all k-reversible languages. ○

In other words, in a machine that is deterministic with lookahead k, initial states and states that can be reached from any given state by accepting a given symbol, have no k-followers in common.

Example 4 Consider this acceptor A:



In this machine, ϵ is a 0-follower of every state, and a is a 1-follower of state 0.

In this machine, there are 2 distinct states, 1 and 4, in $\delta(0, a)$, that both have the same 0-followers. So the machine is not deterministic with lookahead 0.

A^r, on the other hand, is deterministic with lookahead 0.

Acceptor A has no 2 distinct states $q_1, q_2 \in I$ or $q_1, q_2 \in \delta(q, a)$ for some $q \in Q, a \in \Sigma$, such that both q_1 and q_2 have the same 1-followers, so the machine is deterministic with lookahead 1. And A^r is also deterministic with lookahead 1, so the machine is 1-reversible. (And hence k-reversible for all $k \geq 1$.) ○

Proposition 1

a. Every state has exactly one 0-follower, namely the empty string ϵ.

b. A is 0-reversible iff A and A^r are both deterministic.

c. If A is 0-reversible, then the result of removing states that are not reached in accepting any string of $L(A)$ is the canonical acceptor for $L(A)$.

d. If $A = \langle Q, \Sigma, \delta, \{q_0\}, \{q_f\}\rangle$ is 0-reversible and $uv \in L(A)$, then $\delta(q_0, u) = \delta^r(q_f, v^r)$.
So if $u_1 v, u_2 v \in L(A)$, then $\delta(q_0, u_1) = \delta(q_0, u_2)$.

e. For all $k \geq 0$, a deterministic automaton is deterministic with lookahead k.

f. If A is k-reversible and $u_1 vw, u_2 vw \in L(A)$, where $|v| = k$,
then there is a unique state q such that $q = \delta(q_0, u_1 v) = \delta(q_0, u_2 v)$.

g. In fact, let L be any regular language. Then L is k-reversible iff whenever $u_1 vw, u_2 vw \in L$ and $|v| = k$, $u_1 v \equiv_L u_2 v$.

Theorem 20 For every language $L \in \mathscr{L}_{rev}$, there is a finite distinguished subset $D_L \subseteq L$ such that for all $L' \in \mathscr{L}_{rev}$, if $D_L \subseteq L'$ then $L' \not\subset L$.

Proof: If $L = \emptyset$, $D_L = \emptyset$ is a distinguished subset for L.
So now suppose $L \neq \emptyset$, and let $A = \langle Q, \Sigma, \delta, \{q_0\}, \{q_f\}\rangle$ be the canonical acceptor for L. For each $q \in Q$, let $u(q)$ and $v(q)$ be strings of minimum length such that $\delta(q_0, u(q)) = q$ and $\delta(q, v(q)) = q_f$. Let

$$D_L = \{u(q)v(q) |\ q \in Q\} \cup \{u(q)bv(q') |\ q \in Q, b \in \Sigma, \delta(q, b) = q'\}.$$

We show that this set D_L is a distinguished subset for L.
Suppose $L' \in \mathscr{L}_{rev}$ and $D_L \subseteq L'$. We show that $L \subseteq L'$.
We first we show by induction on the length of w that for all $w \in Pr(L)$, (i) $w \equiv_{L'} u(q)$ where $q = \delta(q_0, w)$ (and where $u(q), \delta, q_0$ are as in the definition of D_L just above).
Suppose $|w| = 0$, that is $w = \epsilon$. Since $\delta(q_0, \epsilon) = q_0$ the minimum length string $u(q_0)$ can only be ϵ.
Suppose (IH) the result holds for $|w| \leq n$, and consider any $b \in \Sigma$ such that $wb \in Pr(L)$. By IH, $w \equiv_{L'} u(q)$. But then by the definition of $\equiv_{L'}$ we have (ii) $wb \equiv_{L'} u(q)b$.
Letting $q' = \delta(q, b) = \delta(q_0, wb)$, it follows by the definition of D_L that $u(q')v(q'), u(q)bv(q') \in D_L$.
Since L' is reversible, by Proposition 1d, $\delta(q_0, u(q')) = \delta(q_0, u(q)b)$ and so $u(q') \equiv_{L'} u(q)b$. By (ii) then, $wb \equiv_{L'} u(q')$, completing the inductive proof of (i).
It follows from (i) that for every $w \in L$, $w \equiv_{L'} u(q_f)$, and since $u(q_f) \in D_L$ and $D_L \subseteq L'$, it follows that $w \in L'$. This completes the proof. □

Corollary 12 $\mathscr{L}_{rev}$ is learnable.

learning algorithm ZR, for $\mathscr{L}_{rev}$: (informally!) Given any finite sequence of strings $T[i]$,

1. Construct $PT(content(T[i]))$.
2. Merge final states so that there is a single final state q_f; then merge any distinct states that can lead to q_f on transitions labeled a for any $a \in \Sigma$. Continue in this way to previous states until the resulting machine A is such that A^r is deterministic.
3. Output $ZR(T[i]) = A$. This accepts the smallest reversible language that includes $content(T[i])$.

Full details, proof of correctness, and complexity analysis are given in Angluin's paper.

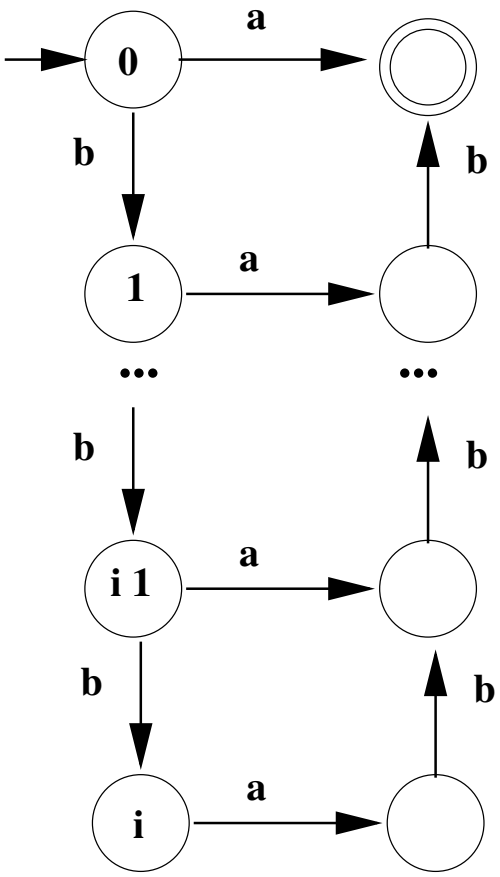
3.2 VC dimension of $\mathscr{L}_{k\text{–reversible}}$

It is easy to show that for every $k \in \mathbb{N}$, the set of k-reversible languages has infinite VC-dimension (Stabler, 2009, pp217-8). That is, there is no finite bound on the size of the sets that are shattered by $\mathscr{L}_{k\text{–rev}}$.

We can establish the infinite VC dimension of $\mathscr{L}_{0\text{–rev}}$ as follows. For any finite $k \geq 0$, let the language

$$L_k = \{b^i a b^i | \ 0 \leq i \leq k\}.$$

It is clear that every such L_k is reversible:



Furthermore, it's clear that every subset of this language is reversible, since it will be defined by the result of deleting any number of the a-arcs (and then we can remove any states and arcs that are not on a path from the start state to the final state to obtain the canonical automaton). Clearly the size of L_k grows with k, and every such set can be shattered by $\mathscr{L}_{rev}$ since every subset of L_k is also reversible.

From this we have immediately the infinite VC dimension for all the k-reversible classes, $k \geq 0$, since they all include $\mathscr{L}_{0\text{–rev}}$.

3.3 Exercises

1. Consider this sequence of 8 strings:

$$T[8] = \langle \begin{array}{l} \text{John sings,} \\ \text{John will sing,} \\ \text{John will have sung,} \\ \text{John will have been singing,} \\ \text{John has sung,} \\ \text{John has been singing,} \\ \text{John is singing,} \\ \text{John sings happily} \rangle \end{array}$$

   a. Draw the prefix tree automaton for $content(T[8])$.

   b. Draw the automaton $ZR(T[8])$.

   c. Is $L(ZR(T[8]))$ finite? If not, which of its elements surprise you the most (or rather, which might surprise someone who did not know how the ZR learner works)?

2. State in the simplest, most intuitive terms possible, the rational basis for the step that generalizes the machine to accept an infinite language in the previous problem. That is, explain why, given the input and innate knowledge about the target class, this generalization step was justified. (Make sure that my mother, a non-linguist, could understand your explanation.)

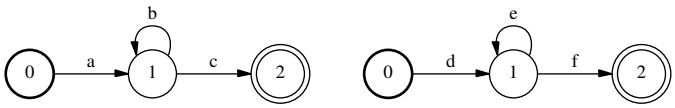
3. Now, abstracting a little bit, consider this set of strings:

$$\text{Aux} = \{\text{Iwhbv}, \text{Iwhv}, \text{Iwv}, \text{Iv}, \text{Ihbv}, \text{Ihv}, \text{Iwbv}, \text{Ibv}\}$$

   Let's count the size of any subset of Aux by counting the symbols in each string. So then size(Aux) = 31. Using this size measure, what is the smallest subset S ⊆ Aux such that:

$$\text{if content(T[i]) = S then L(ZR(T[i])) is infinite.}$$

4. Consider the two zero reversible languages defined by these machines:



   The difference between them is uninteresting: one is just a "relettering" of the other.

   Suppose we abstract away from "lexical differences" like this, so that these two languages count as the same example (mere reletterings of each other). With this abstraction, do we obtain a finite class of "core" zero-reversible acceptors? Explain why I say: no.

References

References

Aho, A.V., J.E. Hopcroft, and J.D. Ullman. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Massachusetts.

Angluin, Dana. 1982. Inference of reversible languages. *Journal of the Association for Computing Machinery*, 29:741–765.

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

Leslie, Ted. 1995. *Efficient Approaches to Subset Construction*. Ph.D. thesis, University of Waterloo, Ontario.

Lewis, H. R. and C. H. Papadimitriou. 1981. *Elements of the Theory of Computation*. Prentice-Hall, Englewood Cliffs, New Jersey.

Moll, R.N., M.A. Arbib, and A.J. Kfoury. 1988. *An Introduction to Formal Language Theory*. Springer-Verlag, NY.

Perrin, Dominique. 1990. Finite automata. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier, NY, pages 1–57.

Stabler, Edward P. 2009. Computational models of language universals: Expressiveness, learnability and consequences. In Morten H. Christiansen, Chris Collins, and Shimon Edelman, editors, *Language Universals*. Oxford University Press, NY, pages 200–223.

Watson, Bruce W. 1993. A taxonomy of deterministic finite automata minimization algorithms. Computing science report 93/44, Eindhoven University of Technology.

Chapter 4 Learning subsets of the PDFAs

Extending the basic strategy in Angluin (1982), this section considers methods for learning not just subsets of deterministic finite automata (DFAs), but probabilistic DFAs (PDFAs), which define probability distributions over regular languages.

4.1 Distances between languages (or parts of a language)

(0) A language learner could begin by assuming each utterance, and each part of each utterance is different from any other. But then the learner should notice that the environments in which one part occurs are relevantly similar to environments in which the other occurs – maybe the grammar should treat them as equivalent, as the same category. This description is intentionally vague – different grammars can differ about what counts as a part, what counts as a relevant environment for a part, etc.

In a context free rewrite grammar and certain other kinds of grammars, there is a natural definition of the 'yield' of each category: what pronounced sequences can be derived from that category.[1] Then each category can be regarded as defining a language: its yield. Then a learner could ask of any two categories: "Are their yields similar enough that I should regard them as the same category?"

So here is the problem. Suppose we have two grammars G, G' (or two automata, or two categories). What is an appropriate measure of the distance $d_p(G, G')$ from one to the other? (Some of our measures will not be symmetric, and will fail the triangle inequality $d(x_1, x_2) < d(x_1, x_3] + d[x_3, x_2)$.)

[1] This is not generally well-defined for certain context sensitive grammars, where what gets rewritten may be more than one symbol, or may vary with context.

4.1.1 Distances between sets of strings

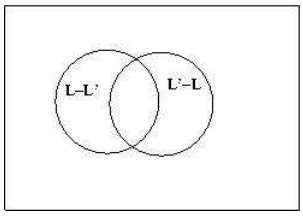
(1) When $L, L' \subseteq \Sigma^*$, we can compare them by checking whether $L = L'$, $L \subset L'$, $L \cap L'$.
When the languages are finite, we can compare the sizes $|L|$ and $|L \cap L'|$.

4.1.2 Distances between probabilistically sampled languages

(2) **p(false positives)+p(false negatives).** If the grammars G, G' define subsets L, L' of Σ^*, and we have a probability distribution p over Σ^*, then we could define the distance as the

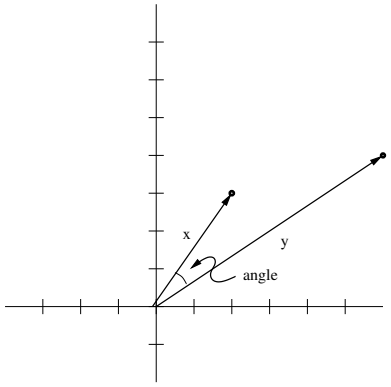
$$d_p(G, G') = p(L' - L) + p(L - L').$$

This idea is common in some of the early work on PAC learning, e.g. Kearns and Vazirani (1994).



(3) Another common measure is to look at word frequencies in the samples. Sampling L and L', we could compute the frequencies of each word in the respective samples $x : \Sigma \to \mathbb{N}$, $y : \Sigma \to \mathbb{N}$. Then, regarding these counts as Σ-dimensional vectors, it is common to measure their distance with the cosine of the angle θ between them

$d_{cos}(L, L')] = \cos\theta$



It is easy to calculate θ given any two points using the 'cosines law':

$$\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \, \|\mathbf{y}\| \cos\theta,$$

where the **norm** or 'length' of a vector

$$\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{n} x_i^2}.$$

and where θ is the angle between $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. Rearranging, and recalling that (for $-1 \leq \theta \leq 1$) $\arccos\theta = \alpha$ iff $\cos\alpha = \theta$,

$$\theta = \arccos\left(\frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \, \|\mathbf{y}\|}\right),$$

where the **dot product of the vectors**

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} x_i y_i.$$

This measure has the advantage that the absolute counts do not matter – only their ratios matter. The points of interest will only be in the upper right, positive quadrant, of course, but the the cosine approaches 0 as the two points become more nearly orthogonal.
This measure has the disadvantage that word order is neglected! We could count trigrams (or even larger n-grams) instead of words, to introduce some order information, but that seems like a crude strategy, not well-motivated.

4.1.3 Distances between probability distributions on Σ^*

If the grammars G, G' define probablities p, q over Σ^*, then there are various ways to compare them:

(4) **maximum difference (L_∞).**

$$d_\infty(p,q) = \max_{x\in\Sigma^*} |p(x) - q(x)|$$

(5) **variation distance (L_1).**

$$d_{L1}[p,q] = \sum_{x\in\Sigma^*} |p(x) - q(x)|$$

This idea appears in some recent work (Palmer and Goldberg, 2007) and is discussed below.

This has the problem that every string x has equal weight.

(6) **mean square distance, Euclidean distance (L_2).**

$$d_{L2}(p,q) = \sqrt{\sum_{x\in\Sigma^*} (p(x) - q(x))^2}$$

Again, this has the problem that every string x has equal weight.

(7) **cross-entropy.** Charniak (1993, pp26,33) says "..., there are reasons for wanting a good probabilistic model... there is a good figure of merit that can be used to compare such models... In dealing with real English, we do not know the correct model... So we can ask, how much good does our approximate model do? One measure of this is given by the cross entropy."

$$d_{CE}(p,q) = -\sum_{x\in\Sigma^*} p(x) \log q(x).$$

What does d_{CE} signify, intuitively? Well, comparing this to the entropy of p,

$$H(p) = -\sum_{x\in\Sigma^*} p(x) \log p(x)$$

(discussed below) we see that d_{CE} will be greater when p, q differ.

This measure has the advantage that it is scaled by the probability of each event x.

(8) **relative entropy (KL divergence).** When what we care about is the divergence between grammars and not the actual entropy of either one, it is natural to subtract the entropy from the cross entropy. This is, in effect, what relative entropy is.

$$\begin{aligned} d_{KL}(p,q) &= d_{CE}(p,q) - H(p) \\ &= -\textstyle\sum_{x\in\Sigma^*} p(x) \log q(x) + \sum_{x\in\Sigma^*} p(x) \log p(x) \end{aligned}$$

Equivalently, as pointed out by Nederhof and Satta (2004, (37)), Cortes et al. (2008, (7))

$$d_{KL}(p,q) = \sum_{x\in\Sigma^*} p(x) \log \frac{p(x)}{q(x)} = E_p[\log \frac{p(x)}{q(x)}].$$

$D(p\|q)$ is a common notation for $d_{KL}(p,q)$

Notice that both this measure and the previous one are scaled by taking logs, so (Cover and Thomas, 1991, p9) say "relative entropy arises as the exponent in the probability of error in a hypothesis test between distributions p and q."

(9) For any p, q, d_{KL} is larger than any of the other values (Reid and Williamson, 2009)

$$L_\infty \leq L_2 \leq L_1 \leq KL$$

Cf. also Hellinger distance, Jensen-Shannon distance, χ^2-distance, Triangle distance,... (Topsøe, 2000; Csiszar and Korner, 1997; Reid and Williamson, 2009).

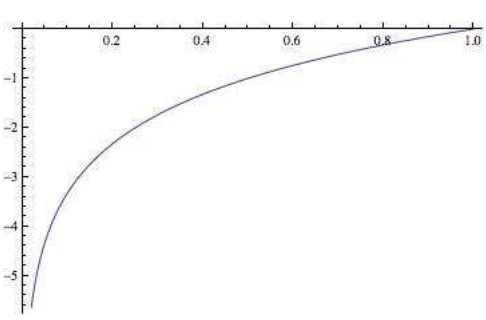
We begin with (8) relative entropy.

4.2 Relative entropy

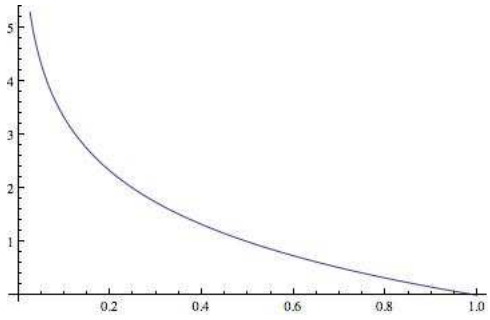
(10) The surprisal of an event A in a space with probability p

$$i(A) = \log \frac{1}{p(A)} = -\log p(A),$$

normally using $\log_2$, for units in "bits".



p vs $\log p$



p vs $\log \frac{1}{p}$

as p increases, surprisal decreases

Example. So if we have 10 possible outcomes $A_0, \ldots, A_9$ with equal probabilities, so $p(A_0) = 0.1$, then

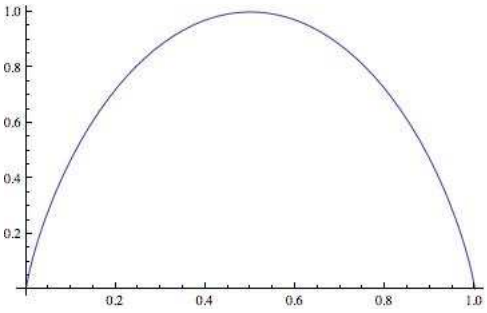
$$i(A_0) = \log \frac{1}{0.1} = -\log 0.1 \approx 3.32 \text{ bits}$$

If the outcomes $B_0, \ldots, B_9$ are independent, then $p(A_0, B_0) = 0.01$, and $i(A_0, B_0) \approx 6.64$. "Surprisals add where probabilities multiply," as the saying goes.

(11) Intuitively, the entropy is average surprisal of events from some source X. The entropy of probability mass function p over discrete set X

$$H(p) = -\sum_{x \in X} p(x) \log p(x)$$

(12) If there are just 2 possible events (e.g. flipping a coin that never lands on an edge), we can plot of the entropy of these events as a function of the probability of one of them. The entropy averages over all the events, so sometimes we say that this is the entropy of the 'source' of the events (e.g. of the coing flipping, as a function of the fairness of the coin):



That's a nice shape. It has a unique maximum value at 0.5, and from any other point, if you can figure out your slope, moving uphill always takes you toward that unique maximum.

That is, entropy is a concave function of p.[2]

(13) A set $C \subseteq \mathbb{R}^n$ is **convex** iff for any points in C, the line segment connecting them is entirely in C too. That is, for all $\mathbf{x}, \mathbf{y} \in C$, and all $0 \leq c \leq 1$,

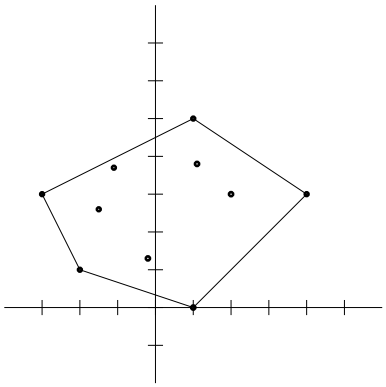
$$c\mathbf{x} + (1 - c)\mathbf{y} \in C.$$

- Given any set $S \subseteq \mathbb{R}^n$, the **convex hull** of S

$$Co\ S = \{c_1\mathbf{x}_1 + \ldots + c_n\mathbf{x}_n | \ \mathbf{x}_i \in S,\ 0 \leq c_i \leq 1,\ c_1 + \ldots + c_n = 1\}.$$

This is the smallest convex set containing the elements of S.

[2] Cf. general proofs in Boyd and Vandenberghe (2004, p67), Cover and Thomas (1991, p23).



The convex hull $Co\ S$ of a set $S \subset \mathbb{R}^2$

- A function defined on a convex set is convex iff the graph of f lies (on or) below the line segment between $(x, f(x))$ and $(y, f(y))$ So a function f on points in an interval (a, b) is **convex** iff $\forall x, y \in (a, b)$ and $\forall 0 \leq c \leq 1$

$$f(cx + (1 - c)y) \leq c(f(x)) + (1 - c)f(y).$$

A function f is **concave** iff $-f$ is convex.

(14) The relative entropy (Kullback-Leibler divergence) between distributions p, q

$$D(p||q) = \sum_{x \in X} \log \frac{p(x)}{q(x)} = E_p[\log \frac{p(x)}{q(x)}],$$

with conventions (Cover and Thomas, 1991, p19)

$$0 \log \frac{0}{q} = 0 \qquad p \log \frac{p}{0} = \infty. \qquad 0 \log \frac{0}{0} = 0$$

(15) The relative entropy $D(p||q)$ is a convex function of p, q. That is, for any pairs p_1, q_1 and p_2, q_2 of probability functions,

$$D(cp_1 + (1 - c)p_2||cq_1 + (1 - c)q_2) \leq c(D(p_1||q_1)) + (1 - c)D(p_2||q_2).$$

$\forall 0 \leq c \leq 1$ (Cover and Thomas, 1991, p30)

4.3 Weighted finite state automata

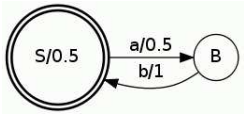
Now following Cortes et al. (2008), to set the stage for a more efficient computation of relative entropy in §4.4.4, we introduce this slight variation on standard notation:

(16) Automaton $A = (\Sigma, Q, I, F, E, \lambda, \rho)$ over $\mathbb{K}$ where

- Σ finite nonempty set, the alphabet
- Q is a finite set, the states
- initial states $I \subseteq Q$, final states $F \subseteq Q$
- $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \mathbb{K} \times Q$ the edges or transitions
- $\lambda : I \to \mathbb{K}$ defines initial weights,
- $\rho : F \to \mathbb{K}$ defines the final weights

(17) We could consider various kinds of weights $\mathbb{K}$. For example, we could let the weights be non-negative real numbers $\mathbb{R}_+ = \{n \in \mathbb{R} | \ n \geq 0\}$ which are summed and multiplied in the usual way.

(18) **Example.** Let's call this automaton (18):



This automaton is has the following parts

$\Sigma = \{a, b\}$
$Q = \{S, B\}$
$I = \{S\}$ (indicated by bold circle)
$F = \{S\}$ (indicated by double circle)
$E = \{ (S, a, 0.5, B), (B, b, 1.0, S) \}$
$\lambda = \{(S, 1), (B, 0)\}$
$\rho = \{(S, 0.5), (B, 0)\}$

(19) **Functions on edges.** Given any edge $e = (q0, a, p, q1) \in E$,

$i[e] = a$	input label
$p[e] = q0$	'previous' state
$n[e] = q1$	'next' state
$w[e] = p$	weight

(20) **Paths.** A path π is a sequence of adjacent edges

$$\pi = ((q0, a_0, p_0, q1), (q1, a_1, p_1, q2), \ldots, (qn-1, a_n, p_n, qn)).$$

- Such a π is a member of $P(q0, qn)$, the set of paths from state $q0$ to state qn
- Such a π is a member of $P(q0, a_0 \ldots a_n, qn)$, the set of paths from state $q0$ to state qn labeled $a_0 \ldots a_n$
- Let $SP(q0, qn)$ be the set of 'simple paths' from state $q0$ to state qn, paths in which no two edges have the same previous state. That is, no cycles properly between the first and last state. NB
- The previous functions on edges are extended to paths in the obvious way:

$i[\pi] = a_0 \ldots a_n$	input label
$p[\pi] = q0$	'previous' state
$n[\pi] = qn$	'next' state
$w[\pi] = \otimes_{e_i \in \pi} w[e_i]$	weight

- A cycle is a path π with the same initial and final states, that is, $p[\pi] = n[\pi]$.
- A path π is accepting iff $p[\pi] \in I$ and $n[\pi] \in F$. That is, we can say, $\pi \in P(I, F)$.
- $L(A) = \{i[\pi] | \pi \in P(I, F)\}$.
- Following Cortes et al. (2008, p.6), and identically Cortes et al. (2006, p.326), define

$$s[A] = \sum_{\pi \in P(I,F)} w[\pi],$$

when it is defined and in $\mathbb{K}$. This is the weight of all the paths. It may be undefined if a cycle, which can be taken any number of times, has weight greater than 1.

(21) For any such A and any input $x \in \Sigma^*$ define

$$[\![A]\!](x) = \oplus_{\pi \in P(I,x,F)} \lambda[p[\pi]] \otimes w[\pi] \otimes \rho[n[\pi]]$$

That is, $[\![A]\!]$ maps each string x to the sum of the weights of its derivations.

(22) Note that if there is a unique start state q_0 and $\lambda(q_0) = 1$, and a unique final state q_f, $\rho(q_f) = 1$, and if the automaton is unambiguous, then for any $x \in L(A)$, there is a unique path in $P(I, x, F) = \{\pi\}$ and $[\![A]\!](x) = \otimes_{e \in \pi} w[e]$.

(23) **Example.** Considering automaton (18) again,

$[\![(18)]\!](\epsilon) = 0.5$
$[\![(18)]\!](a) = 0.0$
$[\![(18)]\!](b) = 0.0$
$[\![(18)]\!](aa) = 0.0$
$[\![(18)]\!](ab) = 0.25$
...

Claim: $\oplus_{x \in \Sigma^*} [\![(18)]\!](x) = \oplus_{x \in (ab)^*} [\![(18)]\!](x) = 1$.

(24) A is unambiguous iff for all $x \in \Sigma^*$ at most one path accepts x

(25) **Example.** (18) is unambiguous.

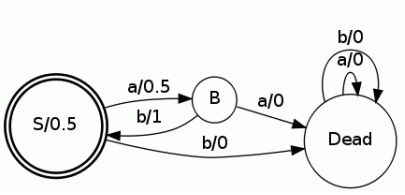
(26) A is deterministic iff (i) it has a unique initial state and (ii) no 2 distinct edges e, e' have the same source $p[e] = p[e']$ and input label $i[e] = i[e']$

(27) **Example.** (18) is deterministic.

(28) As noted by Cortes et al. (2008, p.6), this definition of determinism allows ϵ transitions, since the algorithm for unambiguous automata presented below can handle them. But also, ϵ transitions can be removed from unambiguous weighted automata (Mohri, 2002a).

(29) A is complete iff for all $a \in \Sigma^*$, and all states $q \in Q$, there is some edge $(q, a, w, q') \in E$.

(30) **Example.** (18) is not complete, but we can define a complete automaton (30) such that $[\![(18)]\!] = [\![(30)]\!]$:



(31) Let the distance from state $p \in Q$ to $q \in Q$

$$d[p, q] = \oplus_{\pi \in P(p,q)} w[\pi]$$

where $P(p, q)$ is the set of paths π from p to q

(32) A is trim iff every state can be reached from an initial state, and from every state a final state can be reached.

(33) **Example.** (18) is trim, but (30) is not.

(34) Cortes et al. (2008, Def 3, p.6) say automaton A with weights in $\mathbb{R}_+$ is **probabilistic** iff

- for all $q \in Q$, $\oplus_{\pi \in P(q,q)} w[\pi]$ is defined and in $\mathbb{R}_+$, and
- $\sum_{x \in \Sigma^*} [\![\mathrm{A}]\!](x) = 1$.

Probabilistic automaton A is **stochastic** if, at every state, the final weight of that state plus the sum of the weights of outgoing transitions is 1.

(35) **Example.**

- In automaton (18), since $I = F = \{\mathrm{S}\}$, $P(\mathrm{S}, \mathrm{S})$ is the set of accepting paths, an infinite set. So $0 < \oplus_{\pi \in P(\mathrm{S},\mathrm{S})} w[\pi]$ is defined and an element of $\mathbb{R}_+$, but this should be proven.[3]
- In automaton (18), $P(\mathrm{B}, \mathrm{B})$ is an infinite set, and $0 < \oplus_{\pi \in P(\mathrm{B},\mathrm{B})} w[\pi]$ is defined and an element of $\mathbb{R}_+$, but this should be proven.[3]
- In automaton (30), $P(\mathrm{Dead}, \mathrm{Dead})$ is an infinite set, and $\oplus_{\pi \in P(\mathrm{Dead},\mathrm{Dead})} w[\pi] = 0$ is defined and an element of $\mathbb{R}_+$.
- (18) and (30) are both probabilistic, but a proof should be provided.[3]

(36) Given 2 automata A_1, A_1 define

$$A_1 \cap A_1 = (\Sigma, Q_1 \times Q_2, I_1 \times I_2, F_1 \times F_2, E, (\lambda_1, \lambda_2), (\rho_1, \rho_2))$$

where E is the smallest set such that

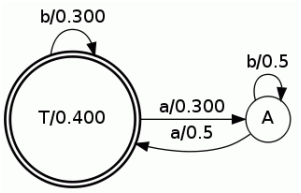
$$(q_1, a, w_1, q_1) \in E_1 \text{ and } (q_1', a, w_1', q_2') \in E_2 \Rightarrow ((q_1, q_1'), a, (w_1 \otimes w_1'), (q_2, q_2')) \in E$$

It follows that:

- The machine $A_1 \cap A_2$ can be computed from A_1, A_2 in time $O(|A_1||A_2|)$
- $[\![\mathrm{A}_1 \cap \mathrm{A}_2]\!](x) = [\![\mathrm{A}_1]\!](x) \otimes [\![\mathrm{A}_2]\!](x)$

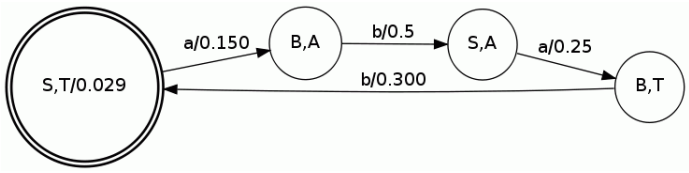
(37) **Example:** Consider the following automaton, which we will call (37):

[3] Cf. the key lemma, Cortes et al. (2008, Lemma7, p.11) discussed in (51) below.



This automaton accepts every sequence of with an even number (possibly zero) of a's.

(38) **Example:** When we compute (18)∩(37), we expect the resulting machine to accept strings whose lengths are multiples of 4. Let's call this automaton (38):



Claim: (38) = (18) ∩ (37)

Claim: For $x \in \Sigma^*$, $[\![(38)]\!](x) = [\![(18)]\!](x) \otimes [\![(37)]\!](x)$

4.4 Computing exact relative entropy of two automata

The calculation of relative entropy is similar to the calculation of the probability of a string in fundamental respects, if instead of real numbers we use pairs of numbers, and modify the product and sum operations. The more general perspective that this invites has been useful for many things.

4.4.1 Carrasco'97

(39) Carrasco (1997, p7) shows that for deterministic machines A, A'

$$D([\![\mathrm{A}]\!]||[\![\mathrm{A}']\!]) = \sum_{q_i \in Q} \sum_{q'_j \in Q'} \sum_{a \in \Sigma} c_{ij} [\![\mathrm{A}]\!](q_i, a) \log \frac{[\![\mathrm{A}]\!](q_i, a)}{[\![\mathrm{A}']\!](q'_j, a)}.$$

where $[\![\mathrm{A}]\!](q_i, a) = p$ iff $(q_i, a, w, q_j) \in E$ for some (unique) $q_j \in Q$, and similarly for $[\![\mathrm{A}']\!]$, and where

$$c_{ij} = \sum_{x \in L_{ij}} [\![\mathrm{A}]\!](x\Sigma^*|L),$$

$$L_{ij} = \{x \in \Sigma^* |\ \delta(q_i, x) = q_i \wedge \delta'(q'_i, x) = q_j\}$$

A method for evaluating these coefficients is provided (Carrasco, 1997, §4).

(40) Carrasco (1997, p7) also shows compares values of $D([\![\mathrm{A}]\!]||[\![\mathrm{A}']\!])$ estimated from samples with the actual value, showing that convergence can be slow.

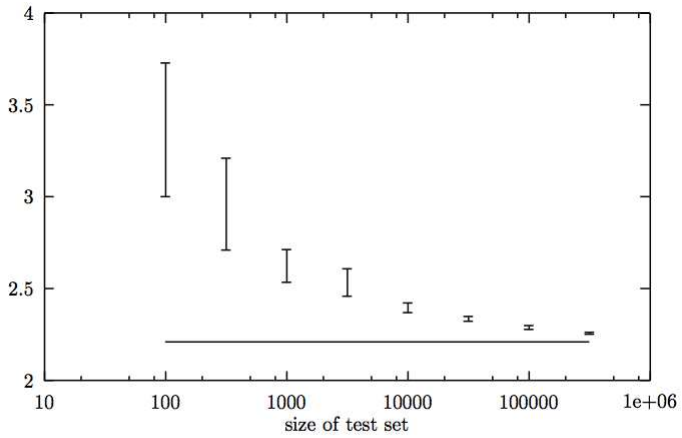


Figure 1: Relative entropy (in bits) between two randomly generated grammars of size 10. Solid line: exact computation. Dots: estimation using samples.

Carrasco's method for computing relative entropy of two automata is less efficient than the one in Cortes&al and in Mohri'09, so we return to this alternative approach.

4.4.2 Semirings

(41) A semiring is a set $\mathbb{K}$ with some basic elements and operations. $(\mathbb{K}, \oplus, \otimes, 0, 1)$ is a **semiring** iff

$(\mathbb{K}, \oplus, 0)$ is a commutative monoid, that is, for all $x, y, z \in \mathbb{K}$
$$\begin{cases} (x \oplus y) \oplus z = x \oplus (y \oplus z) & (\oplus \text{ is associative}) \\ (x \oplus y) = (y \oplus x) & (\oplus \text{ is commutative}) \\ (x \oplus 0) = (0 \oplus x) = x & (0 \text{ is an identity for } \oplus) \end{cases}$$

$(\mathbb{K}, \otimes, 1)$ is a monoid, that is, for all $x, y, z \in \mathbb{K}$
$$\begin{cases} (x \otimes y) \otimes z = x \otimes (y \otimes z) & (\otimes \text{ is associative}) \\ (x \otimes 1) = (1 \otimes x) = x & (1 \text{ is an identity for } \otimes) \end{cases}$$

$\otimes$ distributes over $\oplus$ that is, for all $x, y, z \in \mathbb{K}$
$$\begin{cases} (x \oplus y) \otimes z = (x \otimes z) \oplus (x \otimes y) \\ x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z) \end{cases}$$

0 is an annihilator for $\otimes, \oplus$, that is, for all $x \in \mathbb{K}$ $\quad (x \otimes 0) = (x \oplus 0) = 0.$

(42) A semiring is **commutative** iff $\otimes$ is (i.e. $a \otimes b = b \otimes a$),

(43) A semiring is **idempotent** iff $\oplus$ is (i.e. $a \oplus a = a$).

(44) A **star** semiring is a semiring in which, for any $a \in \mathbb{K}$ the infinite sum

$$a^* = a \oplus a \oplus \cdots = \oplus_0^\infty a^n$$

is defined and in $\mathbb{K}$. In this case it is common to treat * as one of the basic operations: $(\mathbb{K}, \oplus, \otimes, *, 0, 1)$. (And of course, the semiring axioms hold for the infinite sums, as for all elements of $\mathbb{K}$.)

(45) A semiring is **complete** iff for any countable sequence $a_0, a_1, \ldots,$ $\oplus_{0<i} a_i$ is defined and in $\mathbb{K}$.

(46) Define $\mathbb{R}_\infty = \mathbb{R} \cup \{+\infty, -\infty\}$ and let $\mathbb{K} = \mathbb{R}_\infty \times \mathbb{R}_\infty$ where

$$\begin{aligned} (x_1, y_1) \oplus (x_2, y_2) &= (x_1 + x_2, y_1 + y_2) \\ (x_1, y_1) \otimes (x_2, y_2) &= (x_1 x_2, x_1 y_2 + x_2 y_1) \\ (x, y)^* &= \begin{cases} (\frac{1}{1-x}, \frac{y}{(1-x)^2}) & \text{if } 0 \le x < 1 \\ (\infty, ??) & \text{if } 1 \le x \end{cases} \end{aligned}$$

(47) What to do with the infinite cases in this definition? For $x, y \in \mathbb{K}$ where at least one of $x, y \notin \mathbb{R}$, let:

$$\begin{aligned} x + y &= \begin{cases} +\infty & \text{if either } x \text{ or } y \text{ is } +\infty \\ -\infty & \text{otherwise} \end{cases} \\ xy &= \begin{cases} 0 & \text{if either } x = 0 \text{ or } y = 0 \\ +\infty & \text{if both } x, y > 0 \text{ or both } x, y < 0 \\ -\infty & \text{otherwise} \end{cases} \end{aligned}$$

(48) Lemma: With these definitions, $(\mathbb{K}, \oplus, \otimes, *, (0,0), (1,0))$ is a complete star semiring.

It is easy to check that the proof sketch offered by Cortes et al. (2008, Lemma 4) goes through for the infinite cases as it should:

- The infinite sums a^* are present in $\mathbb{R}_\infty$. In fact, when $0 \leq a < 1$, the corresponding $a^* \in \mathbb{R}$ is finite.
- $(\mathbb{K}, \oplus, (0,0))$ is a commutative monoid with $(0,0)$ as the identity for $\oplus$
- $(\mathbb{K}, \oplus, (1,0))$ is a monoid with $(1,0)$ as the identity for $\otimes$ and $(0,0)$ as the annihilator for $\otimes$
- $\otimes$ distributes over $\oplus$.

(49) Eisner (2001) calls this the **expectation semiring** but Cortes et al. (2006, 2008) call it the **entropy semiring**. Nothing said so far in these notes motivates either name! ... but see §4.4.3 immediately below.

4.4.3 The entropy semiring

(50) Given automaton $A = (\Sigma, Q, I, F, E, \lambda, \rho)$ with weights over $\mathbb{R}_+$

let $\Phi_1(A) = (\Sigma, Q, I, F, E', \lambda', \rho')$ with weights over $\mathbb{K} = \mathbb{R}_\infty \times \mathbb{R}_\infty$ be defined as follows:

- $(q, a, p, q') \in E$ iff $(q, a, (p, 0), q') \in E'$
- $\lambda(q) = p$ iff $\lambda'(q) = (p, 0)$
- $\rho(q) = p$ iff $\rho'(q) = (p, 0)$

let $\Phi_2(A) = (\Sigma, Q, I, F, E', \lambda', \rho')$ with weights over $\mathbb{K} = \mathbb{R}_\infty \times \mathbb{R}_\infty$ be defined as follows:

- $(q, a, p, q') \in E$ iff $(q, a, (1, p), q') \in E'$
- $\lambda(q) = p$ iff $\lambda'(q) = (1, p)$
- $\rho(q) = p$ iff $\rho'(q) = (1, p)$ NB: Φ_2 applied when p is a log weight

let $\log A = (\Sigma, Q, I, F, E', \lambda', \rho')$ with weights over $\mathbb{R}_\infty$ be defined as follows:

- $(q, a, p, q') \in E$ iff $(q, a, \log p, q') \in E'$
- $\lambda(q) = p$ iff $\lambda'(q) = \log p$
- $\rho(q) = p$ iff $\rho'(q) = \log p$ NB: If A is stochastic, $\log A$ has log weights, which are neg

(51) **Key Lemma.** Given automata A, B over $\mathbb{R}_+$ where all edges in A have weight $0 \leq p < 1$. Then letting

$$m1(A) = \Phi_1(A) \cap \Phi_2(\log A)$$
$$m2(A, B) = \Phi_1(A) \cap \Phi_2(\log B)$$

the following are defined and in $\mathbb{K} = \mathbb{R}_\infty \times \mathbb{R}_\infty$:

$$s[m1(A)]$$
$$s[m2(A, B)].$$

(52) An immediate corollary: If automata A, B are stochastic in the sense of (34), then $s[m1(A)]$, $s[m2(A, B)]$ are defined and in $\mathbb{K} = \mathbb{R}_\infty \times \mathbb{R}_\infty$. See Cortes et al. (2006, p.330), Cortes et al. (2008, p.12).

(53) If A, B are stochastic, complete, unambiguous, with unique start and final states, then

$$s[m1(A)] = (1, -H(A))$$
$$s[m1(A)] = s[m2(A, A)]$$
$$s[m2(A, B)] = (1, x) \text{ for some } x \leq -H(A)$$

So $s[m1(A)] - s[m2(A, B)] = (0, x)$ for some $x \geq 0$.

(54) **Thm** (Cortes et al., 2006, p.328), (Cortes et al., 2008, p.10):

$$\begin{aligned} s[m1(A)] - s[m2(A, B)] &= s[\Phi_1(A) \cap \Phi_2(\log A)] - s[\Phi_1(A) \cap \Phi_2(\log B)] \\ &= (0, D(A||B)) \end{aligned}$$

The proof uses the equivalences mentioned in (8).

4.4.4 Cortes&al'08

Cortes et al. (2006, 2008) notice that the computation of relative entropy of two automata uses the structure of the automata in a way analogous to the computation of probabilities of strings. Both are instances of shortest distance problems which can be solved with a generalized all-pairs shortest-distance Floyd-Warshall algorithm (Mohri, 2002b).[4]

(55) Conventions used for (51) and (54) repeated here: Cortes et al. (2008, p.9), Cortes et al. (2006, p.328)

- Automata A, B are unambiguous, complete stochastic automata, with weights $0 \leq w < 1$
- In both A, B, λ and ρ assign 1 to a unique state and 0 to all other states – see (22)

Then to compute the relative entropy of automata A, B we construct the entropy-semiring automata $\Phi_1(A) \cap \Phi_2(\log A)$ and $\Phi_1(A) \cap \Phi_2(\log B)$. With these, we can use the algorithm in (56), twice, to compute the values

$$\begin{aligned}(p_A, e_A) &= s[\Phi_1(A) \cap \Phi_2(\log A)] \\ (p_B, ce_{AB}) &= s[\Phi_1(A) \cap \Phi_2(\log B)]\end{aligned}$$

With these, we have the relative entropy $ce_{AB} - e_A$.

(56) The Floyd-Warshall algorithm given by Mohri (2009, Figure 2):[5]

```
for i ← 0 to |Q|                                          (initialize)
    for j ← 1 to |Q|
        d[i, j] ← ⊕_{e∈E∩P(i,j)} w[e]
for k ← 0 to |Q|
    for i ← 1 to |Q|, i ≠ k                               (main loop)
        for j ← 1 to |Q|, j ≠ k
            d[i, j] ← d[i, j] ⊕ (d[i, k] ⊗ d[k, k]* ⊗ d[k, j])
    for i ← 1 to |Q|, i ≠ k                               (final loop)
        d[k, i] ← d[k, k]* ⊗ d[k, i]
        d[i, k] ← d[i, k] ⊕ d[k, k]*
    d[k, k] ← d[k, k]*                                    (diagonal fix)
```

[4] The generalization is the recognition from Lehmann (1977) that the algorithm applies when we have a complete star semiring, even when $\oplus$ is not idempotent.

[5] This version corrects the one in Cortes et al. (2008, p.12), identically Cortes et al. (2006, p.331).

4.5 'Smoothing' probabilities

(57) Conventions:

- We will consider only DFA with unique final states. If necessary, we can add a new symbol zeta ζ to mark the end of a string, as done in Clark and Thollard (2004).
- We will let $I = \{0\}$ and $F = \{\max(Q)\}$, with $\rho(0) = \lambda(\max(Q)) = 1$
- We will only consider DFA in which

$$\forall q \in Q,\ (\rho(q) + \sum_{e \in E, p[e]=q} w[e]) = 1$$

With this convention, if we write weights on the arcs, we don't need to also provide ρ.

(58) (**Sample probabilities**) Given any such DFA A over alphabet Σ, and any $s \in \Sigma^*$, define

$$\begin{array}{rcl} cnt_{q,a}(s) & = & \text{the number of edge occurrences } e \text{ in } P(q_0, s, q_f) \text{ such that } p[e] = q, i[e] = a \\ cnt_q(s) & = & \text{the number of edge occurrences } e \text{ in } P(q_0, s, q_f) \text{ such that } p[e] = q \end{array}$$

We extend these functions to sequences of strings $S \subset (\Sigma^*)^*$:

$$\begin{aligned} cnt_{q,a}(S) &= \sum_{s \in S} cnt_{q,a}(s) \\ cnt_q(S) &= \sum_{s \in S} cnt_q(s) \end{aligned}$$

Now, for any finite sample $T[i]$ we define the sample-determined weights for any state q and symbol σ,

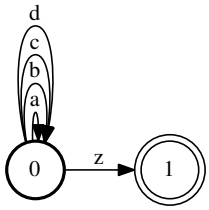
$$\gamma_{T[i]}(q, \sigma) = \frac{cnt_{q,a}(T[i])}{cnt_q(T[i])}.$$

Cortes&al write E for the set of edges or transitions (q, a, w, q'), so that means, in the Cortes&al notation, that for each $(q, a, w, q') \in E$ we set $w = \gamma_{T[i]}(q, a)$.

(59) **Example.** Suppose we have the sample

$$T[5] = \langle ac, bd, bc, ac, ac \rangle \qquad \text{(calculated in (69), below).}$$

Appending z's to each string, a learner might hypothesize that this sample is generated by this DFA:



To get the best fit with the data, we calculate $\gamma_{T[5]}$ with the following steps. First, we calculate:

$cnt_{q_0}(T[5]) = 15$
$cnt_{q_0,a}(T[5]) = 3$, since q_0 precedes every symbol that occurs in $T[5]$
$cnt_{q_0,b}(T[5]) = 2$
$cnt_{q_0,c}(T[5]) = 4$
$cnt_{q_0,d}(T[5]) = 1$
$cnt_{q_0,z}(T[5]) = 5$

Then, following (58), we calculate weights that fit the data most tightly, as follows:

$$\gamma_{T[i]}(q_0, a) = \tfrac{3}{15}, \quad \gamma_{T[i]}(q_0, b) = \tfrac{2}{15}, \quad \gamma_{T[i]}(q_0, c) = \tfrac{4}{15}, \quad \gamma_{T[i]}(q_0, d) = \tfrac{1}{15}, \quad \gamma_{T[i]}(q_0, z) = \tfrac{5}{15}.$$

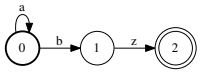
How well does this weighting fit the data? We can calculate

$$\begin{aligned}
[\![\mathrm{A}]\!](T[i]) &= [\![\mathrm{A}]\!](acz) * [\![\mathrm{A}]\!](bdz) * [\![\mathrm{A}]\!](bcz) * [\![\mathrm{A}]\!](acz) * [\![\mathrm{A}]\!](acz) \\
&= (\tfrac{3}{15}\tfrac{4}{15}\tfrac{5}{15})^3 * \tfrac{2}{15}\tfrac{1}{15}\tfrac{5}{15} * \tfrac{2}{15}\tfrac{4}{15}\tfrac{5}{15} \\
&= (0.017778)^3 * 0.0029630 * 0.011852 \\
&= 1.9731 \times 10^{-10}
\end{aligned}$$

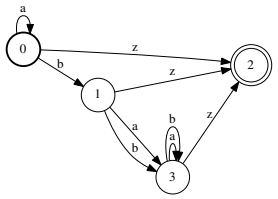
No setting of the probabilities on the automaton in (59) can make the data set $T[5]$ more likely than this.

(60) As we see in this example, probabilities of sentences can be very small. In practical applications, with larger grammars and longer strings, the numbers get really tiny, so it is common to use scaled weights instead. Commonly, instead of p, the negative log of p is used, or something similar. In this case, for example, $-\log_2(1.9731 \times 10^{-10}) = 6.1248 \times 10^4$.

(61) (**Fuzzy completion with smooth probabilities.**) Suppose we hypothesize the following DFA with $\Sigma = \{a, b\}$ for sample text $\langle ab \rangle$:



We complete the machine **fuzzily** by (a) adding a ground (≠dead!) state, (b) adding all possible arcs labeled with Σ going to the ground state (except none from final state), and then (c) adding final transitions labeled z from each state to the final state:



This is like standard completion up to step (c). Now there are $|\Sigma| + 1$ arcs leaving every state except the final state. Given any sample, like $\langle abz \rangle$, Given smoothing constant $0 < \gamma_{min} < \frac{1}{|\Sigma|+1}$, finite sample $T[i]$, and a 'fuzzily complete' DFA A over Σ (as defined in the example below), define

$$\gamma_{smooth}(q, \sigma) = \begin{cases} \gamma_{T[i]}\nu + \gamma_{min}, & \text{where } \nu = (1 - (|\Sigma| + 1)\gamma_{min}) \quad \text{if } q \neq \text{ground} \\ \frac{1}{|\Sigma|+1} & \text{if } q = \text{ground} \end{cases}$$

The idea is: *some probability is removed from non-zero weights $\gamma_{T[i]}$ for distribution over unattested transitions.* Note that all outgoing arcs from the ground state are equally likely (each is traversed 0 times), so we divide the unit uniformly among them. In our example, setting $\gamma_{min} = 0.001$, we calculate:

$$\nu = (1 - (|\Sigma| + 1)\gamma_{min}) = (1 - \tfrac{2+1}{1000}) = \tfrac{997}{1000}$$

$\gamma_{\langle abz \rangle}(0, a)$	=	$\frac{1}{2}$	$\gamma_{smooth}(0, a)$	=	$\frac{\nu}{2} + \gamma_{min} = \frac{997}{2000} + \frac{2}{2000} = \frac{999}{2000}$
$\gamma_{\langle abz \rangle}(0, b)$	=	$\frac{1}{2}$	$\gamma_{smooth}(0, b)$	=	$\frac{\nu}{2} + \gamma_{min} = \frac{997}{2000} + \frac{2}{2000} = \frac{999}{2000}$
$\gamma_{\langle abz \rangle}(0, z)$	=	0	$\gamma_{smooth}(0, z)$	=	$0\nu + \gamma_{min} = \frac{1}{1000}$
$\gamma_{\langle abz \rangle}(1, a)$	=	0	$\gamma_{smooth}(1, a)$	=	$0\nu + \gamma_{min} = \frac{1}{1000}$
$\gamma_{\langle abz \rangle}(1, b)$	=	0	$\gamma_{smooth}(1, b)$	=	$0\nu + \gamma_{min} = \frac{1}{1000}$
$\gamma_{\langle abz \rangle}(1, z)$	=	1	$\gamma_{smooth}(1, z)$	=	$\nu + \gamma_{min} = \frac{998}{1000}$
$\gamma_{\langle abz \rangle}(3, a)$	=	$\frac{1}{3}$	$\gamma_{smooth}(3, a)$	=	$\frac{\nu}{3} + \gamma_{min} = \frac{997}{3000} + \frac{3}{3000} = \frac{1}{3}$
$\gamma_{\langle abz \rangle}(3, b)$	=	$\frac{1}{3}$	$\gamma_{smooth}(3, a)$	=	$\frac{\nu}{3} + \gamma_{min} = \frac{997}{3000} + \frac{3}{3000} = \frac{1}{3}$
$\gamma_{\langle abz \rangle}(3, z)$	=	$\frac{1}{3}$	$\gamma_{smooth}(3, a)$	=	$\frac{\nu}{3} + \gamma_{min} = \frac{997}{3000} + \frac{3}{3000} = \frac{1}{3}$

We see that the outgoing arcs of every state except the final state now sum to 1, every arc has positive weight, and Σ^* is accepted. For example, while

$$p(aabz) = \frac{999}{2000} \cdot \frac{999}{2000} \cdot \frac{999}{2000} \cdot \frac{998}{1000} \approx 0.12438,$$

we have

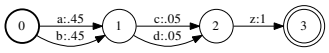
$$p(bbbz) = \frac{999}{2000} \cdot \frac{1}{1000} \cdot \frac{1}{3} \cdot \frac{1}{3} \approx 5.5500 \times 10^{-5}.$$

(62) This is a simple 'backoff' smoothing method: we back off from the actual relative frequencies of attested events so that we can distribute some probability over unattested events. See for example Chen and Goodman (1998) for a tutorial survey of various smoothing methods.

(63) From this perspective the learner makes two fundamental choices: (i) over what space is the probability defined (i.e. 'what is the model?'), and (ii) how is probability distributed over the space defined by the language model? Sometimes this latter question is broken again into two sub-questions: (ii.a) how does the evidence (e.g. relative frequencies) indicate probability to be distributed, and (ii.b) how much probability mass should be reserved, at each point, for things that are unevidenced, and how should that probability mass be allocated? Answers to (ii) (or sometimes just ii.b) are often called 'smoothing strategies'.[6]

4.6 Example: comparing 4 earlier learners

(64) Consider this example automaton A:



(65) Obviously, $L(A) = \{acz, bcz, adz, bdz\}$.

(66) Note that this language is in 4 learnable classes we have already studied, plus the one more from Clark&Thollard that is presented in §4.7 below:

$$L(A) \in \mathscr{L}_{fin} \quad L(A) \in \mathscr{L}_{SL2} \quad L(A) \in \mathscr{L}_{prec} \quad L(A) \in \mathscr{L}_{0-rev} \quad L(A) \in \mathscr{L}_{|Q|=3,Es=2,\mu=0.45}$$

where the parameters for this last class are set for a particular example (64), where $|Q|$ is an upper bound on the number of states of the target, $Es = 2$ is a bound on the expected length of tail from any state, and $\mu = 0.45$ is a distinguishability threshold, defined in (74) below.

(67) The distribution $p : \Sigma^* \to [0, 1]$ defined by the DFA in (64), for any $s \in \Sigma^*$,

$$p(s) = \begin{cases} 0.45 & \text{if } s \in \{acz, bcz\} \\ 0.05 & \text{if } s \in \{adz, bdz\} \\ 0 & \text{otherwise.} \end{cases}$$

(68) The entropy

$$\begin{aligned} H(p) &= -\sum_{x \in \Sigma^*} p(x) \log p(x) \\ &= -2(0.45 * \log 0.45 + 0.05 * \log 0.05) \\ &= 1.469 \text{ bits} \end{aligned}$$

[6] Remember that the smoothing aims to provide an estimate of how likely utterances are, how frequent. We must be realistic about what can be achieved here in real settings.

- We cannot expect to find the right way to smooth actual linguistic hypotheses, because there can be no correct model of the actual frequencies of utterances. Many factors, and especially many non-linguistic factors obviously influence what will be said on any occasion. It is unreasonable to expect any scientific account of that. But we can find approximations that work more or less in certain normal circumstances, where we do not expect to ever be able to define 'more or less' or 'normal'. (compare weather forecasting)

  These general considerations are not mentioned in the tutorial Chen and Goodman (1998), where various smoothing methods are compared for how well they do on n-gram probabilities for various corpora. The worry is that those corpora are not really representative, and more seriously, that there is not really any fixed, real thing for them to be representative of. (For engineering tasks like data retrieval, this does not really matter – as long as many relevant things are returned in many cases, it can be useful.)

- ES thinks (as argued by Fodor, Chomsky, et al) the prospects for finding the human, individual, (possibly incorrect) way of estimating probabilities of utterances do not look good either: the factors taken into consideration are unbounded. We have no good models of unbounded reasoning over 'what to say/how to interpret' (compare Fodor (2008)). Note that Bayesian models provide a criterion for comparison based on priors and evidence, but no conception of unbounded comparisons, of comparisons over unbounded spaces involving unknown concepts (compare Gelman and Shalizi (2010); Earman (1992)).

Learner's judgements about syntactic surprisal are clearly influenced by frequencies, even though there are many other factors. And ES has the optimistic belief that the human choice of model – question (i) – may be within our reach in a more substantial way than (ii) is. Although (i) too depends on percieved frequencies of utterances, the sensitivity to frequencies may be a rather small factor, as evidenced by the relative insensitivity of acquisition to details of the communicative environment. Clearly learners do notice details, but the details can vary quite widely and still support a normal course of language acquisition.

Compare rolling fair tetrahedral dice

$$\begin{aligned} H(\text{tetra-dice}) &= -4(0.25 * \log 0.25) \\ &= -4(0.25 * -2) \\ &= 4(0.5) \\ &= 2 \text{ bits} \end{aligned}$$

(69) **Data.** We use a random number generator to generate data from this machine. Let's use the first sequence of length 5, but I generate some others to see what they look like. (Code in appendix on page 79.)

```
# randSeq 5 [("ac",0.45);("bc",0.45);("ad",0.05);("bd",0.05);];;
  <ac, bd, bc, ac, ac>
- : unit = ()
# randSeq 5 [("ac",0.45);("bc",0.45);("ad",0.05);("bd",0.05);];;
  <ac, bc, ac, bc, ac>
- : unit = ()
# randSeq 5 [("ac",0.45);("bc",0.45);("ad",0.05);("bd",0.05);];;
  <ac, ad, ac, ac, ac>
- : unit = ()
```

(70) **Practice.** To get a first impression of what the cross entropy and relative entropy are like, let's compare p from (67) to the distribution q defined by the automaton in (59). For any $s \in \Sigma^*$,

target distribution p

$$p(s) = \begin{cases} \frac{9}{20} & \text{if } s \in \{acz, bcz\} \\ \frac{1}{20} & \text{if } s \in \{adz, bdz\} \\ 0 & \text{otherwise.} \end{cases}$$

hypothesis q

$$q(s) = \prod_{x \in \Sigma, x \in s} \gamma_{T[5]}(q_0, x)$$

$$\text{where } \gamma_{T[5]}(q_0, x) = \begin{cases} \frac{3}{15} & \text{if } x = a \\ \frac{2}{15} & \text{if } x = b \\ \frac{4}{15} & \text{if } x = c \\ \frac{1}{15} & \text{if } x = d \\ \frac{5}{15} & \text{if } x = z \end{cases}$$

Note that while p is 0 on all but 4 strings, so these calculations by hand are easy (cf conventions in (14)):

CE. Recalling the definition of d_{CE} from (7) on page 61, we consider first the strings that p assigns positive probability – acz,bcz,adz,bdz – and then the rest:

$$\begin{aligned} d_{CE}(p,q) &= -\sum_{x \in \Sigma^*} p(x) \log q(x) \\ &= -(\tfrac{9}{20} * \log(\tfrac{3}{15}\tfrac{4}{15}\tfrac{5}{15}) + \tfrac{9}{20} * \log(\tfrac{2}{15}\tfrac{4}{15}\tfrac{5}{15}) + \tfrac{1}{20} * \log(\tfrac{3}{15}\tfrac{1}{15}\tfrac{5}{15}) + \tfrac{1}{20} * \log(\tfrac{2}{15}\tfrac{1}{15}\tfrac{5}{15}) + 0 * \ldots) \\ &= -(-2.6162 + -2.8794 + -0.39069 + -0.41994 + 0) \\ &= 6.3062 \text{ bits} \end{aligned}$$

Compare $H(p)$ which was calculated in (68) just above.

KL. Now recalling the definition of d_{KL} from the second equation in (8) on page 61, consider the strings that p assigns positive probability first – acz,bcz,adz,bdz, in that order – and then the rest:

$$\begin{aligned} d_{KL}(p,q) &= \textstyle\sum_{x \in \Sigma^*} p(x) \log \frac{p(x)}{q(x)} \\ &= (\tfrac{9}{20} * \log \tfrac{9}{20} / (\tfrac{3}{15}\tfrac{4}{15}\tfrac{1}{15}) + \tfrac{9}{20} * \log \tfrac{9}{20} / (\tfrac{2}{15}\tfrac{4}{15}\tfrac{1}{15}) + \tfrac{1}{20} * \log \tfrac{1}{20} / (\tfrac{3}{15}\tfrac{1}{15}\tfrac{1}{15}) + \tfrac{1}{20} * \log \tfrac{1}{20} / (\tfrac{2}{15}\tfrac{1}{15}\tfrac{1}{15}) + 0 * \ldots) \\ &= (2.0978 + 2.3610 + 0.17459 + 0.20384 + 0) \\ &= 4.8372 \text{ bits} \end{aligned}$$

Since just above we calculated d_{CE} and $H(p)$, we have another way to calculate this same value, using the first equation in (8), which says:

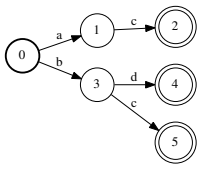
$$\begin{aligned} d_{KL}(p,q) &= d_{CE}(p,q) - H(p) \\ &= 6.3062 - 1.469 \\ &= 4.8372 \text{ bits} \end{aligned}$$

4.6.1 ϕ_e using prefix tree automata, with γ_{smooth}.

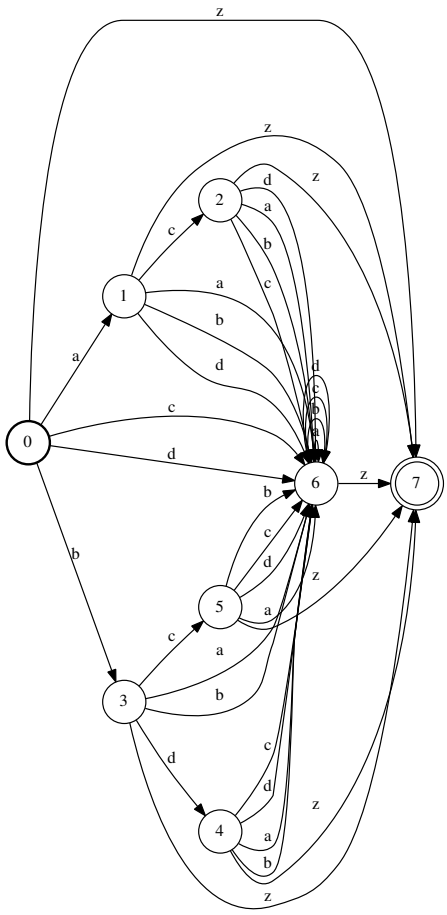
(71) Recall that ϕ_e learns $\mathscr{L}_{fin}$ by, in effect, remembering what has been heard. One grammar/automaton that encodes what has been heard is the prefix tree automaton. Let's consider the sample already used in (59):

$$T[5] = \langle ac, bd, bc, ac, ac \rangle \qquad \text{(calculated in (69), above).}$$

The prefix automaton is:



Completing the automaton by adding ground state 6, and then adding z's so that everything is accepted, we get:



With 5 arcs leaving every non-final state, there are 35 arcs in this machine. What should the smoothing constant be? For the moment, this choice seems arbitrary, so let $\gamma_{min} = 0.001$.

$$\Rightarrow \text{more coming!} \Leftarrow$$

4.6.2 $\mathscr{L}_{SL2}$ bigram learner, with γ_{smooth}

4.6.3 $\mathscr{L}_{prec}$ learner, with γ_{smooth}

4.6.4 $\mathscr{L}_{0-rev}$ learner ZR with γ_{smooth}

4.7 Clark&Thollard'04: $\phi_{PDFA,|Q|,Es,\mu}$

Notation:

- I write Es for the max expected length of tail from any state. (C&T use L.)
- For any DFA and sample $T[i]$, for any state $q \in Q$ any $\sigma \in \Sigma$, define the 'suffix distributions'

  $S_q =$ the list of xz such that for some $uxz \in T[i]$, $\delta(q_0, u) = q$ and $\delta(q, xz) = q_f$.

  $S_{q,\sigma} =$ the list of elements xz such that $\sigma xz \in S_q$
- For any $s \in \Sigma$, $S_q(s)$ = the number of occurrences of s in S_q. Similarly for $S_{q,\sigma}(s)$.
- $S_{u,\sigma}, S_v$ are **similar, not μ-distinct** iff $\forall s \in S_{u,\sigma} \cup S_v, \quad |\frac{S_{u,\sigma}(s)}{|S_{u,\sigma}|} - \frac{S_v(s)}{|S_v|}| \leq \frac{\mu}{2}$ cf L_∞ def in (4), p.61

Algorithm $\phi_{PDFA,|Q|,Es,\mu}(T, \epsilon, \delta)$ where T is the sample source:

1. **Initialize.**
   - Calculate m_0, N and get sample $T[N]$ where N is defined as follows, with parameter $|Q| = n$:

$$\gamma_{min} = \frac{\epsilon}{4(Es+1)(|\Sigma|+1)} \tag{4.1}$$

$$\delta' = \frac{\delta}{2(n|\Sigma|+2)} \tag{4.2}$$

$$\epsilon_1 = \frac{\epsilon^2}{16(|\Sigma|+1)(Es+1)^2} \tag{4.3}$$

$$m_0 = \max\left(\frac{8}{\mu^2}\log\left(\frac{96n|\Sigma|}{\delta'\mu}\right), \frac{1}{2\epsilon_1^2}\log\left(\frac{12n|\Sigma||\Sigma+1|}{\delta'}\right)\right) \tag{4.4}$$

$$\epsilon_3 = \frac{\epsilon}{2(n+1)\log(4(Es+1)(|\Sigma|+1))/\epsilon} \tag{4.5}$$

$$N = \frac{4n|\Sigma|Es^2(Es+1)^3}{\epsilon_3^2}\max\left(2n|\Sigma|m_0, 4\log\frac{1}{\delta'}\right) \tag{4.6}$$

   - Begin a graph with just one node q_0, and let $S_{q_0} = T[N]$, the suffix distribution of q_0.
2. **While for any** $q \in Q,\ \sigma \in \Sigma,\ |S_{q,\sigma}| > m_0$:

   if $S_{q,\sigma}$ is similar to any $S_{q'}$ in the graph,

   then add arc $\delta(q, \sigma) = q'$;

   else add new node q'' to graph and add arc $\delta(q, \sigma) = q''$.
3. **Fuzzy complete and smooth.** Fuzzily complete the graph and smooth, as defined in (61) on page 71 above.

4.7.1 Analysis: PAC wrt d_{KL}

(72) **(Thm 5, p480)** For every DFA A with $|Q|$ states, with distinguishability $\mu > 0$ such that the expected length of the string generated from every state is $\leq Es$, for every $1 > \delta > 0$ and $1 > \epsilon > 0$, with probability greater than $1 - \delta$

$$d_{KL}(A, \phi_{PDFA,|Q|,Es,\mu}(T[N])) < \epsilon.$$

(73) **NB:** we see in the proof of this result that, with high probability, the machine built by step 2 is a subgraph of the target.

4.7.2 Example run

(74) Recall that a PDFA is μ-distinguishable iff every pair of states is. States $q_1, q_2 \in Q$ are μ distinguishable iff

$$\exists s \in \Sigma^*, \quad |\gamma(q_1, s\zeta) - \gamma(q_2, s\zeta)| \geq \mu.$$

In the example started in (64) above, we have

$$\begin{array}{l}\gamma(0, ac\zeta) = 0.45 \text{ but } 0 \text{ from every other state}\\ \gamma(1, c\zeta) = 0.9 \text{ but } 0 \text{ from every other state}\\ \gamma(2, \zeta) = 1 \text{ but } 0 \text{ from every other state}\end{array}$$

so we can choose $\mu = 0.45$. Choosing μ smaller than this could lead us to merge states which should really be distinct. With $\mu = 0.45$, with suffix sets of N strings (where constant N is defined in the algorithm) we are likely to find a string different enough to distinguish all pairs of distinct states.

(75) Run $\phi_{PDFA,|Q|,Es,\mu}(T, \epsilon, \delta)$ with $|Q| = n = 4, Es = 3, \mu = 0.45, \epsilon = \delta = 0.01, \Sigma = \{a, b, c, d\}$:

1. • We calculate:

$$\begin{array}{rcl}\gamma_{min} &=& \frac{\epsilon}{4(Es+1)(|\Sigma|+1)} = \frac{0.01}{4*4*5} = 1.2500e-04\\ \delta' &=& \frac{\delta}{2(n|\Sigma|+2)} = \frac{0.01}{2(4*4+2)} = 2.7778e-04\\ \epsilon_1 &=& \frac{\epsilon^2}{16(|\Sigma|+1)(Es+1)^2} = \frac{0.01^2}{16*5*16} = 7.8125e-08\\ m_0 &=& \max\left(\frac{8}{\mu^2}\log\left(\frac{96n|\Sigma|}{\delta'\mu}\right), \frac{1}{2\epsilon_1^2}\log\left(\frac{12n|\Sigma||\Sigma+1|}{\delta'}\right)\right)\\ &=& \max\left(\frac{8}{0.01^2}\log\left(\frac{96*4*2}{2.7778e-04*0.01}\right), \frac{1}{2*(7.8125e-08)^2}\log\left(\frac{12*4*4*5}{2.7778e-04}\right)\right)\\ &=& \max(2.2434e+06, 3.5587e+15)\\ &=& 3.5587e+15\\ \epsilon_3 &=& \frac{\epsilon}{2(n+1)\log(4(Es+1)(|\Sigma|+1))/\epsilon} = \frac{0.01}{2(4+1)\log(4(3+1)(4+1))/0.01} = 7.7126e-05\\ N &=& \frac{4n|\Sigma|Es^2(Es+1)^3}{\epsilon_3^2}\max\left(2n|\Sigma|m_0, 4\log\frac{1}{\delta'}\right)\\ &=& \frac{4*4*4*3^2*4^3}{7.7126e-05^2}\max\left(2*4*4*3.5587e+15, 4\log\frac{1}{2.7778e-04}\right)\\ &=& 2.9285e+14\end{array}$$

   • We let $G = \{q_0\}$. Since $T[N] = T[2.9285e+14]$ is too big to collect, let's just assume that we have a sample S_{q_0} in which each string has a relative frequency equal to its probability in the target.

2. Loop (abbreviating)

   a. $S_{q_0} = \langle acz, bcz, adz, bdz\rangle$
   $S_{(q_0,a)} = \langle cz, dz\rangle$
   candidate $S_{(q_0,a)}$ is not similar to $S_{q,0}$, so we let $q_1 = (q_0, a)$, $q_0 \to aq_1$
   $S_{(q_0,b)} = \langle cz, dz\rangle$ is similar to S_{q_1} so we merge them: $q_0 \to bq_1$

   b. $S_{(q_1,a)} = S_{(q_1,b)} = \emptyset$
   $S_{(q_1,c)} = S_{(q_1,d)} = \langle z\rangle$
   so $q_1 \to cq_2, \quad q_1 \to dq_2$

   c. $S_{(q_2,\_)} = \emptyset$

3. Fuzzy complete and smooth

4.8 Castro&Galvadà'08 $\phi_{PDFA,|Q|,Es}$

> The C-T algorithm receives a number of parameters of the target as input, computes the worst-case numbr of examples required for those parameters, and asks for the full sample up front. Rather, we place ourselves in the more common situation where we have a given, fixed sample and we have to extract as much information as possible from it. Our main theorem then says that the algorithm PAC-learns provided the sample is large enough with respect to the targets parameters (some of which, such as μ, are unknown...) ...In contrast with the C-T algorithm, it does not need as input parameter the distinguishability μ of the target. (pp.4,5)

Notation:

- $\text{prefL}_\infty(p,q) = \max_{x\in\Sigma^*}(p(x\Sigma^*) - q(x\Sigma^*))$
- PDFA state q_i defines a distribution $p_{q_i}(x) = \gamma(q_i, x\zeta)$
- PDFA states q_i, q_j are **μ-distinguishable** iff

$$\mu \le \max\{\text{L}_\infty(p_{q_i}, p_{q_j}), \text{prefL}_\infty(p_{q_i}, p_{q_j})\}.$$

  Note: $\text{prefL}_\infty(p_{q_i}, p_{q_j}) > \mu$ iff there is some $x \in \Sigma^*$ $|\gamma(q_i,x) - \gamma(q_j,x)| \ge \mu$
- Candidate node (q,σ) is **important** iff $|S_{q,\sigma}| > \epsilon_0 \frac{N}{2}$
- Candidate node (q,σ) and node q' are **similar** iff $d > t_{q,\sigma,q'}$ where

$$\begin{aligned} m_{q,\sigma} &= |S_{q,\sigma}| \\ s_{q,\sigma} &= |\text{prefixes}(S_{q,\sigma})| \\ m_{q'} &= |S_{q'}| \\ s_{q'} &= |\text{prefixes}(S_{q'})| \\ t_{q,\sigma,q'} &= \left(\frac{2}{\min(m_{q,\sigma}, m_{q'})} \ln \frac{4m_{q,\sigma}^2 (s_{q,\sigma}+s_{q'})\pi^2}{3\delta_0}\right)^{\frac{1}{2}} \\ d &= \max\left(\text{L}_\infty(S_{q,\sigma}, S_{q'}), \text{prefL}_\infty(S_{q,\sigma}, S_{q'})\right) \end{aligned}$$

Algorithm $\phi_{PDFA,|Q|,Es}(T,\epsilon,\delta)$ where T is the sample source: (NB: no μ parameter, except for N_0 guarantees)

1. **Initialize.**
   - Calculate these constants (and for PAC ϵ,δ guarantees, get sample $T[N]$ where $N > N_0$):

$$\epsilon_1 = \frac{\epsilon^2}{16(|\Sigma|+1)(Es+1)^2} \tag{4.7}$$

$$\epsilon_2 = \frac{\epsilon}{4n(n+1)Es(Es+1)\log(4(Es+1)(|\Sigma|+1)/\epsilon)} \tag{4.8}$$

$$\epsilon_5 = \frac{\epsilon}{4|\Sigma|(n+1)Es(Es+1)\log(4(Es+1)(|\Sigma|+1)/\epsilon)} \tag{4.9}$$

$$\epsilon_0 = \frac{\epsilon_2\epsilon_5}{n|\Sigma|(Es+1)} \tag{4.10}$$

$$\delta_0 = \frac{\delta}{n^2|\Sigma| + 3n|\Sigma| + n} \tag{4.11}$$

$$N_0 = \max\left(\frac{16}{\epsilon_0\mu^2}(3e\ln\frac{48}{\epsilon_0\mu^2} + \ln\frac{16Es\pi^2}{3\delta_0^2}), \frac{8}{\epsilon_0\epsilon_1^2}\ln\frac{2(|\Sigma|+1)}{\delta_0}\right) \tag{4.12}$$

   - Begin a graph with just one 'safe' node q_0 and let $S_{q_0} = T[N]$, the suffix distribution of q_0.
2. **While for any** $q \in Q$, $\sigma \in \Sigma$, $|S_{q,\sigma}| > \epsilon_0\frac{N}{2}$: For each of these 'important candidate nodes' (q,σ)

   if (q,σ) is similar to any node q' in the graph,

   then add arc $\delta(q,\sigma) = q'$;

   else add new node q'' to graph and add arc $\delta(q,\sigma) = q''$.
3. **Merge remaining.** If any important candidate nodes (q_i,σ) have not been promoted to safe nodes in the graph by the previous step, for each, add an edge from (q_i,σ) to v where v is the safe node with the most similar suffix distribution.

4.9 Observations, questions

- State-merging perspective is useful – What criteria for category similarity?
- Both Clark&Thollard and Castro&Galvadà use variants of L_∞ to distinguish categories. With bounds on how improbable distinguishing strings are (given by $\mu, Es, |Q|$), PAC results are available, but maybe L_∞ is not the best choice? cf Balle, Castro, and Gavaldà (2012)
- Now we are in a position to assess this quote from Clark and Lappin (2011):

> The . . . approach to the inference of reversible automata has yielded positive results for the entire class of regular languages. The basic idea is to replace. . . the test for equality of residual languages, with a more refined test. . . Extending the work of Ron, Singer, and Tishby (1995), Clark and Thollard (2004) established that the class of Probabilistic Deterministic Finite State Automata can be PAC learned, and this result has been extended in a number of directions (Palmer and Goldberg, 2007; Castro and Galvadà, 2008). Clark and Thollard (2004) prove that if the class of distributions is restricted to those produced by PDFSAs which generate the same language then the class of languages that these PDFSAs generate can be PAC learned from positive data only. In this sequence of results the class of learnable languages has been expanded from a small nonsuprafinite set that can be identified in the Gold paradigm, to a nonarbitrary infinite class of infinite languages that can be efficiently learned when the set of possible distributions is plausibly constrained to those that facilitate acquisition. (p156)

I think we can formulate what they want to say in a clearer, more accurate way. Consider first: what claims along these lines really seem justified?

- ◦ The class of distributions $\mathscr{L}_{PDFA,|Q|,Es,\mu}$, for particular settings of $|Q|, Es, \mu$, plausibly includes all human phonotactics, and can perhaps be feasibly learned.

Might this idea lead us to notice universal properties of human languages (cf. e.g. Heinz), or generalization strategies that are evidenced in human language acquisition? Maybe. . . note:

- ◦ The learner $\phi_{PDFA,|Q|,Es,\mu}$ is insensitive to order, and typically rather insensitive to the particular contents of each S_q – only the μ distinctness matters.

Furthermore, note that

- ◦ With enough data, and liberal enough settings for $|Q|, Es, \mu$, perhaps linguists (and scientists in other domains) could use these methods to discover the patterns (finite or otherwise) in domains that we do not understand.

Is that plausible? Yes, I think so – remember (73). What this kind of learner will not do, of course, is to

- $\times_1$ explain substantial universals of the sort linguists are expecting, like the lack of intervention effects in long distance assimilation (Heinz, 2010),[7] Greenberg's universal 20 and other constituent order preferences (Cinque, 2005), patterns of intervention in syntax (Friedmann, Belletti, and Rizzi, 2009; Grillo, 2008), etc.
- $\times_2$ find nice simple automata $F_1, \ldots, F_n$ such that the possibly much more complex $\bigcap F_i$ is the target automaton.

Gavaldà et al. (2006, p150) say:

> The standard approach for learning Markov Models with Hidden State uses the Expectation-Maximization framework. While this approach had a significant impact on several practical applications (e.g. speech recognition, biological sequence alignment) it has two major limitations: it requires a known model topology, and learning is only locally optimal. We propose a new PAC framework for learning both the topology and the parameters in partially observable Markov models. Our algorithm learns a Probabilistic Deterministic Finite Automata (PDFA) which approximates a Hidden Markov Model (HMM) up to some desired degree of accuracy.

In this perspective, interesting versions of the innateness question concern are the criteria of linguistic similarity, the ones that provide the best model of human acquisition, specific to language.

- Q When (if) we find human-like settings of $|Q|, Es, \mu$, will they be language-specific? Does something like $\phi_{|Q|,Es,\mu}$ provide a good model for human generalization in regular domains?

Since they are not impressed by worries $\times_1, \times_2$, I think Clark&Lappin mean to answer these questions with 'No reason to think so', and 'Possibly', respectively. – This is a reasonable position. But many open questions here!

[7] Meaghan Fowlie points out that ϕ_{prec} learns dissimilation effects that are not subject to intervention, so maybe that learner does not capture the right generalization. Cf. the discussion of this issue in Heinz (2010, §4.6).

Appendix: Generating random sequences in OCaml

Most programming languages provide a random number generator library which makes it easy to generate random sequences according to a defined probability distribution.

To generate my test sequences from finite sets of outcomes, I used `randSeq`. To generate using a (possibly cyclic) DFA, I use the function `selectByRange` to make the choice among arcs at each step.

```
(* initialize, choosing seed in system-dependent way *)
Random.self_init ();;

(* examples: generate random numbers in [0,1]
Random.float 1. ;;
Random.float 1. ;;
 *)

let rec selectByRange choice outcomes = match outcomes with
    (outcome,_)::[] -> outcome
  | (outcome,probability)::more ->
    if choice < probability
    then outcome
    else selectByRange (choice -. probability) more
  | [] -> failwith "selectByRange error";;

let choose outcomes =
  let choice = Random.float 1. in selectByRange choice outcomes;;

(* example:
choose [("heads",0.3);("tails",0.6);("edge",0.1)];;
*)

let randSeq n outcomes =
  Printf.fprintf stdout "<";
  for k=1 to n
  do
    print_string (choose outcomes);
    if k=n
    then Printf.fprintf stdout ">\n"
    else Printf.fprintf stdout ", ";
  done;;

(* example
randSeq 5 [("ac",0.45);("bc",0.45);("ad",0.05);("bd",0.05);];;
*)
```

References

References

Angluin, Dana. 1982. Inference of reversible languages. *Journal of the Association for Computing Machinery*, 29:741–765.

Balle, Borja, Jorge Castro, and Ricard Gavaldà. 2012. Learning probabilistic automata: A study in state distinguishability. *Theoretical Computer Science*, forthcoming.

Boyd, Stephen and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press, NY.

Carrasco, R.C. 1997. Accurate computation of the relative entropy between stochastic regular grammars. *Theoretical Informatics and Applications (RAIRO)*, 31(5):437–444.

Castro, Jorge and Ricard Galvadà. 2008. Towards feasible PAC-learning of probabilistic deterministic finite automata. In *Proceedings of the 9th international colloquium on Grammatical Inference, ICGI'08*, pages 163–174, Springer-Verlag, Berlin.

Charniak, Eugene. 1993. *Statistical Language Learning*. MIT Press, Cambridge, Massachusetts.

Chen, Stanley and Joshua Goodman. 1998. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Harvard University, Cambridge, Massachusetts.

Cinque, Guglielmo. 2005. Deriving Greenberg's Universal 20 and its exceptions. *Linguistic Inquiry*, 36(3):315–332.

Clark, Alexander and Shalom Lappin. 2011. *Linguistic Nativism and the Poverty of the Stimulus*. Wiley-Blackwell, NY.

Clark, Alexander and Franck Thollard. 2004. PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5:473–497.

Cortes, Corinna, Mehryar Mohri, Ashish Rastogi, and Michael Riley. 2006. On the computation of the relative entropy of probabilistic automata. In *Latin American Theoretical Informatics, LATIN2006*, pages 323–336.

Cortes, Corinna, Mehryar Mohri, Ashish Rastogi, and Michael Riley. 2008. On the computation of the relative entropy of probabilistic automata. *International Journal of Foundations of Computer Science*, 19:219–241.

Cover, Thomas M. and Joy A. Thomas. 1991. *Elements of Information Theory*. Wiley, NY.

Csiszar, Imre and Janos Korner. 1997. *Information Theory: Coding Theorems for Discrete Memoryless Systems*. Akadémiai Kiadó, Budapest.

Earman, John. 1992. *Bayes or Bust? A Critical Examination of Bayesian Confirmation Theory*. MIT Press, Cambridge, Massachusetts.

Eisner, Jason. 2001. Expectation semirings: Flexible EM for finite-state transducers. In *Proceedings of the ESSLLI Workshop on Finite-State Methods in Natural Language Processing (FSMNLP)*.

Fodor, Jerry A. 2008. *LOT2: The Language of Thought Revisited*. Clarendon Press, NY.

Friedmann, Naama, Adriana Belletti, and Luigi Rizzi. 2009. Relativized relatives: Types of intervention in the acquisition of A-bar dependencies. *Lingua*, 119:67–88.

Gavaldà, Ricard, Philipp W. Keller, Joelle Pineau, and Doina Precup. 2006. PAC-learning of Markov models with hidden state. In *Proceedings, 11th European Conference on Machine Learning, ECML-2006*, Lecture Notes in Artificial Intelligence 4212, pages 150–161, Springer.

Gelman, Andrew and Cosma Rohilla Shalizi. 2010. Philosophy and the practice of Bayesian statistics. *ArXiv*. http://arxiv.org/abs/1006.3868.

Grillo, Antonino. 2008. *Generalized Minimality: Syntactic Underspecification in Broca's Aphasia*. Ph.D. thesis, Universiteit Utrecht, The Netherlands.

Heinz, Jeffrey. 2010. Learning long-distance phonotactics. *Linguistic Inquiry*, 41(4):623–661.

Kearns, Michael J. and Umesh V. Vazirani. 1994. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, Massachusetts.

Lehmann, Daniel J. 1977. Algebraic structures for transitive closures. *Theoretical Computer Science*, 4:59–76.

Mohri, Mehryar. 2002a. Generic epsilon-removal and input epsilon-normalization algorithms for weighted transducers. *International Journal of Foundations of Computer Science*, 13(1):129–143.

Mohri, Mehryar. 2002b. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350.

Mohri, Mehryar. 2009. Weighted automata algorithms. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science. Springer, pages 213–254.

Nederhof, Mark-Jan and Giorgio Satta. 2004. Kullback-Leibler distance between probabilistic context-free grammars and probabilistic finite automata. In *Proceedings of the 20th International Conference on Computational Linguistics*, COLING'04.

Palmer, Nick and Paul W. Goldberg. 2007. PAC-learnability of probabilistic deterministic finite state automata in terms of variation distance. *Theoretical Computer Science*, 387(1):18 – 31.

Reid, Mark D. and Robert C. Williamson. 2009. Generalised Pinsker inequalities. *ArXiv*. http://arxiv.org/abs/0906.1244.

Ron, Dana, Yoram Singer, and Naftali Tishby. 1995. On the learnability and usage of acyclic probabilistic automata. In *Proceedings of the 8th Annual Conference on Computational Learning Theory, COLT'95*, pages 31–40.

Topsøe, Flemming. 2000. Some inequalities for information divergence and related measures of discrimination. *IEEE Transactions on Information Theory*, 46:1602–1609.

Chapter 5 Learning subsets of $\mathscr{L}_{CF}$

In this section we will consider methods for learning certain subsets of context free languages. Is English context free? Recalling our earlier introductory discussion in §0, we want to distinguish the questions:

Q1 Is English (as mentally represented, with some variation, by each typical speaker) defined by a CFG?

Q2 Is the grammar G of a typical English speaker such that L(G)$\in \mathscr{L}_{CF}$?

We noticed these obvious methodological points:

M1 The answer to the second question depends on assumptions about the first, even though some linguists seem, at least implicitly, to think they can make generalizations about the infinite extent of languages without any assumptions about the grammar.

M2 The evidence bearing on Q1 goes beyond the finite set of sequences of gestures that we see. First, only some of those sequences of gestures ought to be regarded as relevant, and we make some empirical assumptions there. Second, we use considerations about semantic compositionality, perceptual effects of boundaries, and other general scientific strategies (theoretical simplicity, parsimony) in assessing the plausibility of claims about grammar.

One class of constructions which would evidence non-CF grammars are productive copying constructions. Manaster-Ramer (1986) observes that English has a number of these (but perhaps not as many as most other languages). He points out the following examples – fixed, idiomatic expressions but with productive copying:

(X-shmX) Linguistics shminguistics.

(X-is-a-X-is-a-X) A dog is a dog is a dog.

(X-or-no-X) Linguistics or no linguistics, I am going home.

(X-while-X-is-good) Philosophize while the philosophizing is good!

(X-is-as-X-does) Moral is as moral does.

(X or X?) Is she beautiful or is she beautiful?

(focus reduplication) I'll make the tuna salad, and you make the SALAD-salad. (Ghomeshi et al., 2004)

Oh, we're not LIVING-TOGETHER-living-together.

Constructions like X-or-X and X-or-no-X, and possibly contrastive reduplication, are apparently not subject to any fixed bound on the size of the copied elements. Consider the set of X,Y such that the following is acceptable:

X or no Y, I am going home

If this set were one in which X=Y (or something similar), where X,Y are not subject to any fixed, linguistic bound on size, then the set of English sentences $\notin \mathscr{L}_{CF}$. Arguments about this question are (appropriately) influenced by assumptions you might have about human grammars.[1] The question is somewhat controversial, and (Pullum and Rawlins, 2007, p.285) say "no reason remains for regarding a non-CFness argument based on X or no X as persuasive. Hence those computational linguists who essentially always take the problem of parsing English to be a problem in CF parsing (which is to say, virtually all of them) do so with considerable justification"

[1] For discussion of these questions, see for example Manaster-Ramer (1986), Stabler (2004), Pullum and Rawlins (2007), Kobele (2007).

Clark links the online video presentation of the Clark, Eyraud, and Habrard (2008) conference paper and he begins with a number of methodological and historical claims – very useful for understanding what his goals and hunches are. Let's begin by briefly considering some of those.

5.0.1 What is learnability?

(0) In the video (Clark, Eyraud, and Habrard, 2008, 3:25):

> In first language acquisition, we don't know what the representations are, but we do know that they're learnable. So linguists sometimes talk as they know what the representations are, but they don't. But we do know that they're leanable because the infant child can learn.

- Is it true that languages are learnable? One problem is that we don't know what that question means. What is a 'language'? And what does 'learnable' mean?
- You might think: Yes, we want to define those terms in a way that will let us really understand as true the claim that languages are learnable.

  But ES doesn't think that is right. And certainly, it's not obviously right. If by 'languages' we mean all possible human languages $\mathscr{L}_h$, all languages which are structurally similar to attested human languages in relevant senses, and if by 'learnable' we mean acquirable by the kinds of methods children use, it could well be that languages are not learnable. One way this could happen is that there could be factors X which are relevant for learning but which do not enter into the determination of structure, and hence do not constrain the definition of $\mathscr{L}_h$. In such a situation, it could happen that $\mathscr{L}_h$ is not learnable, but $\mathscr{L}_h \cap X$ is. ES actually thinks this is what happens (as hinted several times already, and as will become crystal clear as we proceed towards more reasonable learning theories for human syntax).

(1) Is it true that linguists don't know what our representations of language are? That is, do we know anything about those representations? (ES thinks we do know quite a few substantial things. We want a learner that fits with them!)

5.0.2 The nature of 'primary linguistic data'

(2) Clark and Lappin (2011, p.163):

> . . . shallow representations with easily observable representational primitives are better candidates for efficient learning. . . [We should] seek representations whose structural properties are transparently recoverable from the PLD.

Obviously, by 'shallowness' they cannot mean depth of derivation tree or anything like that (any cyclic canonical DFSA provides derivation trees of arbitrary depth, for example), but 'depth' of learner inference, in some sense.

(3) In the video (Clark, Eyraud, and Habrard, 2008, 3:25), Clark is even clearer:

> We need to use representations without hidden structure. The representational primitives must be observable. In fact, we must be able to define any repesentational primitive of our representation – any symbol, nonterminal state – has to be definable in purely language theoretic terms.

(4) Is that right? Compare Chomsky (1981, p.11):

> . . . in the case of UG . . . we want the primitives to be concepts that can plausibly be assumed to provide a preliminary, prelinguistic analysis of a reasonable selection of presented data, that is, to provide the data that are mapped by the language faculty to a grammar. . . It would, for example, be reasonable to suppose that such concepts as 'precedes' or 'is voiced' enter into the primitive basis. . . But it would be unreasonable to incorporate, for example, such notions as 'subject of a sentence' notions"

(5) Picking up on (1) again, ES thinks that among the things we know for sure are

- Two expressions with the same pronunciation can have different categories.
- Our representations of language have hidden structure, in the standard sense of 'hidden'. E.g., to take the standard example, because DFAs and HMMs have hidden structure (namely, their states), first order Markov chains of observable events are unable to define hidden syllable structure the way a DFA or HMM or CFG can.

5.0.3 Substitution tests

(6) Harris (1946, pp.163-5) wrote,

> The procedure to be indicated below consists essentially of repeated substitution: e.g. *child* for *young boy* in *Where did the – go*?. To generalize this, we take a form A in an environment C – D and then substitute another form B in the place of A. If, after such substitution, we still have an expression which occurs in the language concerned, i.e. if not only CAD but also CBD occurs, we say that A and B are members of the same substitution-class, or that both A and B fill the position C – D, or the like.
>
> The first step in our procedure is to form substitution classes of single morphemes. We list, for the language concerned, all single morphemes which replace each other in the substitution test, i.e. which occur in the same environments (have the same selection).
>
> We now ask not only if A and B each occur in the environment C – D but also if AE together, or FGH, also occur in that environment. If they do, then A, B, AE, FGH are all substitutable for each other. We may say that they are all members of one substitution class, which is now not merely a class of morphemes but a class of morpheme sequences.
>
> However, there may be sequences of morphemes which occur in environments where single morphemes do not occur, i.e. where they cannot be replaced by any single morpheme... We may consider this sequence to constitute an element in the utterance structure...

(7) In contemporary linguistics texts too, we get warnings against using semantic tests for grammatical category, with advice about using substitution tests instead:

> That is, there is no known method of defining morpho-syntactic categories in semantic terms. At present, the best that can be done is to define morpho-syntactic categories in the terms that Zellig Harris gave us: substitution classes. (Johnson, 2004, §2.1).

Some other texts are much more cautious, emphasizing semantic contraints on substitution tests, and regarding observations of phrases in context and judgments of grammaticality as experimental results:

> As in any other experiment, there is a danger that the experimental result is sensitive to several different variables. In general we want to minimize as much as possible interference by factors not relevant for establishing constituency ... It is important to remember that the way we interpret the results of such psychological experiments is itself part of the theory, a hypothesis to be verified or corroborated (Koopman, Sportiche, and Stabler, 2012, §3.3).

Is this caution necessary? That is, are there really cases where two expressions of exactly the same category cannot occur in the same environments? Yes.

(8) (Keenan and Stabler, 2003, 135ff) mention these examples (these bear directly on ϕ_{SCF} as noted in (62), below)

	substrings	contexts
(a)	Sue laughed	$\langle\epsilon,\epsilon\rangle$, ⟨the boy who kissed,$\epsilon$⟩
(b)	it rains in Spain	$\langle\epsilon,\epsilon\rangle$, *⟨the boy who kissed,$\epsilon$⟩
(c)	and	⟨I saw John,Bill), ⟨I saw both John,Bill⟩
(d)	or	⟨I saw John,Bill), *⟨I saw both John,Bill⟩

In (a) and (b), we have two sentences, presumably the same category, but they do not occur in the same environments. Intuitively, this substitution fails because we have failed to respect constituency. The latter pair of cases (c,d) adds another possible issue, with coordinators failing to respect the dependency between *both* and *and*. We have other evidence from agreement, etc., that *both* and *and* must really be different categories, but examples (c,d) raise the possibility that there could be dependencies that substitution will fail to respect. Let's watch to see whether and how these problems are avoided by the learning methods in this chapter and the next.

(9) **Exercise.** (i) Think of some more examples of the first type (a,b), but using categories other than the start category (S or CP or whatever). (ii) Think of some examples of the latter type (c,d), more persuasive than the one given here.

(10) When Chomsky discusses assumptions of earlier structural linguistics that he wants to discard, he mentions his doubts about general discovery procedures which could take us from corpora to grammars, and he mentions distinctions which structural methods might not reveal: at least not easily, like these:

They are easy to please	They are flying planes	*I know [the men$_i$ want to see them$_i$]
They are eager to please	The chicken is ready to eat	I know who [the men$_i$ want to see them$_i$]

It is not obvious how these things bear on the adequacy of a grammar based on substitution evidence, but let's keep an eye on them as we consider proposals in this and following weeks.

5.0.4 CFGs

(11) A **context free grammar** G has 4 parts, $G = \langle \Sigma, \text{Cat}, (\rightarrowtail), \text{S} \rangle$, where

- Σ is a finite nonempty vocabulary, the "pronounced" or "terminal" elements
- N is a set of categories (disjoint from Σ), the "nonterminals", and let the whole vocabulary of the grammar $\mathsf{V} = (\text{Cat} \cup \Sigma)$.
- $\rightarrowtail$ is a rewrite relation that is a subset of $\text{Cat} \times \mathsf{V}^*$. So each pair (or "rule" or "production") in this relation has the form (C, α), often written $C \rightarrowtail \alpha$, where $C \in \text{Cat}$ is a category and α is a (possibly empty) sequence of vocabulary elements and categories,
- S $\in$ Cat is the category chosen as the start symbol.

(12) To define derivations, we first define $\Rightarrow_G$ as follows, for any $x, y, z \in \mathsf{V}^*$ and any $A \in N$,

$$xAy \Rightarrow_G xzy \text{ iff } A \rightarrowtail z.$$

Then we let $\Rightarrow_G^*$ be the reflexive, transitive closure of $\Rightarrow_G$. Often we drop the subscripts when no confusion will result. Finally, a **derivation** of $s \in \Sigma^*$ from $A \in N$ is a finite sequence of steps:

$$A \Rightarrow \cdots \Rightarrow s.$$

If there is such a derivation, then of course $A \Rightarrow_G^* s$. Such a derivation is leftmost(rightmost) if every step rewrites the leftmost(rightmost) category. We sometimes write $x \Rightarrow_{lm} y$ to indicate a leftmost rewrite step, and similarly $x \Rightarrow_{rm} y$ for rightmost.

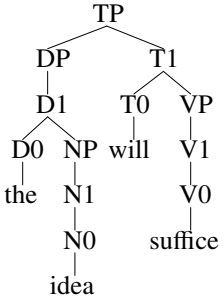
(13) The **language** defined by the grammar $L_G = \{s \in \Sigma^* | \ S \Rightarrow_G^* s\}$. And in general, we can consider the "language" or "yield" of any category A: $yield_G(A) = \{s \in \Sigma^* | \ A \Rightarrow_G^* s\}$, so $L_G = yield_G(\text{S})$.

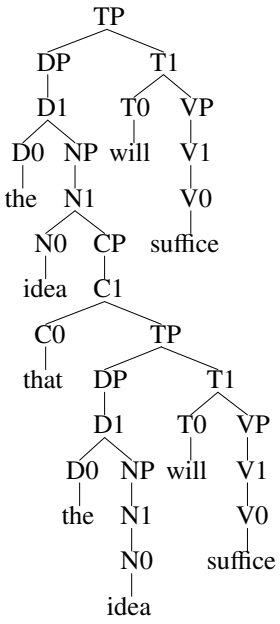
(14) **Example grammar G with start symbol TP**:

TP $\rightarrowtail$ DP T1	T1 $\rightarrowtail$ T0 VP	T0 $\rightarrowtail$ will
DP $\rightarrowtail$ D1	D1 $\rightarrowtail$ D0 NP	D0 $\rightarrowtail$ the
NP $\rightarrowtail$ N1	N1 $\rightarrowtail$ N0 CP	N0 $\rightarrowtail$ idea
	N1 $\rightarrowtail$ N0	
VP $\rightarrowtail$ V1	V1 $\rightarrowtail$ V0	V0 $\rightarrowtail$ suffice
CP $\rightarrowtail$ C1	C1 $\rightarrowtail$ C0 TP	C0 $\rightarrowtail$ that

Nerode's theorem easily shows that the language defined by this grammar is not regular:

$$\text{L(G)=\{(the idea that)}^n \text{ the idea will suffice (will suffice)}^n | \ n \geq 0\}.$$





5.0.5 PCFGs

Basic properties of PCFGs are surveyed in Chi (1999).

(15) Given a context free grammar $G = \langle \Sigma, N, (\rightarrowtail), \mathrm{S}, P \rangle$, a probabilistic context free (PCF) grammar $G = \langle \Sigma, N, (\rightarrowtail), \mathrm{S}, P \rangle$, where the function $P : (\rightarrowtail) \rightarrow [0, 1]$ maps productions to real numbers in the closed interval between 0 and 1 in such a way that

$$\sum_{\langle \mathsf{c}, \beta \rangle \in (\rightarrowtail)} P(\mathsf{c} \rightarrowtail \beta) = 1$$

We will often write the probabilities assigned to each production on the arrow: $c \xrightarrow{p} \beta$

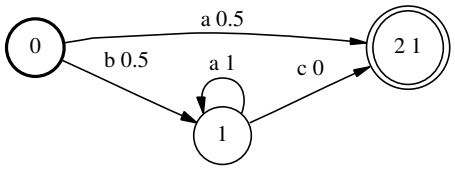
(16) The **probability of a derivation** is the product of the probabilities of the productions in the derivation. And the **probability of a string** is the sum of the probabilities of its derivations.

(17) A PCFG is said to be **consistent** iff $p(L(G)) = 1$.
That is, where G defines the whole space of linguistic events, given that a linguistic event is going to occur, the probability that some linguistic event occurs should be 1. Not more, not less!

(18) **Thm: Regular grammar consistency.** A regular grammar is consistent iff from every state there is a non-zero probability of reaching some terminal string.

(19) Example inconsistent probabilistic regular grammar with start category 0, with the corresponding inconsistent **probabilistic finite state automaton** diagram:

$0 \xrightarrow{0.5}$ a 2
$0 \xrightarrow{0.5}$ b 1
$1 \xrightarrow{1}$ a 1
$1 \xrightarrow{0}$ c 2
$2 \xrightarrow{1} \epsilon$



Note that, in the finite state diagram, the probability p of the empty production from some state q (= stopping) is commonly written q/p.

(20) The following grammar is not regular, though the language it defines is:

$$S \xrightarrow{0.4} a$$
$$S \xrightarrow{0.6} S\,S$$

This grammar is inconsistent. If we sum the probabilities in decreasing order, the series does not sum to 1 (Chi, 1999, p.132):

$$0.4 + 0.6 * 0.6 * 0.4 + \ldots$$

(21) **Thm: PCFG consistency.** (Grenander, 1967; Booth and Thompson, 1973) Consider the matrix $T(G)$ with rows indexed by N and columns indexed by $(\rightarrowtail)$, where for any $A \in N, r \in (\rightarrowtail)$,

$$T(A, r) = \begin{cases} 0 & \text{if } R \neq A \rightarrowtail \gamma \qquad \text{(for some } \gamma\text{)} \\ p(r) & \text{otherwise.} \end{cases}$$

This is sometimes called the first-moment matrix for G. Let ρ be the largest eigenvalue of the first-moment matrix for PCFG G. Then G is consistent if $|\rho| < 1$ and inconsistent if $|\rho| > 1$. When $|\rho| = 1$, G is consistent iff there is no 'final class' of nonterminals, where a final class is a set of nonterminals which, when rewritten, always produce at least one other member of the class.[2] See Wetherell (1980); Harris (1963, for example), for algorithms to check for $|\rho| < 1$.

(22) **Thm.** A PCFG is consistent if the probabilities of its rules are the relative frequencies of those rules in a finite sample of derivations. (Chi and Geman, 1998)

[2]This result is sometimes attributed to the unpublished technical report Grenander (1967), but it is independently presented in Booth and Thompson (1973). It is also covered in the text Gonzalez and Thomason (1978).

5.1 Learning $\mathscr{L}_{SCF}$ from examples

Let's study the learner in Clark and Eyraud (2005) and Clark and Eyraud (2007).

(23) **Def.** In regular, right-branching grammars, categories define suffixes of strings. But in context free grammars, categories define substrings that could be on the right, the left, or in the middle. So instead of using Nerode's right congruence, let's say, for any $L \subseteq \Sigma^*$ and any $x, y \in \Sigma^*$

$$x \equiv_L y \quad \text{iff} \quad (\forall l, r \in \Sigma^*,\ lxr \in L \text{ iff } lyr \in L)$$

(24) **Thm.** $\equiv_L$ is an equivalence, and immediately from this definition we see that it is a congruence with respect to concatenation:

$$\text{if } x \equiv_L y, \text{ then } \forall a \in \Sigma,\ xa \equiv_L ya.$$

$$\text{In fact:} \quad \text{if } x \equiv_L y, \text{ then } \forall l, r \in \Sigma^*,\ lxr \equiv_L lyr$$

This is, in effect, a substitution principle.

As usual, let $[u]_{\equiv_L} = \{v |\ v \equiv_L u\}$, leaving off subscripts when they are clear from context.

(25) **Bkgd considerations relevant to congruences and categories.** Consider zero-reversible DFA, or the DFA hypotheses of Clark&Thollard. In these grammars, are the sets of good finals

$$\{\{u \in \Sigma^* |\ A \Rightarrow^* u\} |\ A \text{ a state/category of } G\}$$

partitions of Σ^* (or of the set of strings that have some category)? No. Similarly for the actual grammars of human languages $\mathcal{G}_h$.

But as noted by Keenan and Stabler (2003), categorizing strings $\langle u, C\rangle$ trivially induces a partition by category. That is, for any categorized expression e

$$\{\{e |\ e \text{ has category } C\} |\ C \text{ a state/category of } G\}$$

is obviously a partition of the set of all categorized strings. So we can write $e \equiv_{Cat} e'$ iff e, e' have the same category.

Furthermore, in the zero-reversible DFA, or the DFA hypotheses of Clark&Thollard, the equivalence $\equiv_{Cat}$ is a congruence wrt the rules of the grammar. That is, for every rule f of the grammar/automaton and any $u \in \text{Dom}(f)$, if $u \equiv_{Cat} v$ then $v \in \text{Dom}(f)$ and $f(u) \equiv_{Cat} f(v)$.

(In the terminology of K&S: in such grammars, a permutation of strings of the same category is a symmetry, an 'automorphism', in the sense that it leaves the rules of the grammar unchanged. K&S do not claim that the grammars of $\mathcal{G}_h$ have this property, but this is argued by Stabler (2012).)

(26) **Def.** In any L and $u \in \Sigma^+$, let the **contexts of** u, $C_L(u) = \{\langle l, r\rangle |\ lur \in L\}$

(27) **Def.** Let's say $u, v \in \Sigma^+$ are **weakly substitutable in** L,

$$u \doteq_L v \text{ iff } \exists l, r \in \Sigma^*,\ lur \in L \text{ and } lvr \in L.$$

(28) **Thm.** Weak substitutability, $\doteq$, is not necessarily transitive.

Proof: For example, when $S = \{ab, bb, bc\}$ we have

- $a \doteq_S b$ since both share $\langle \epsilon, b\rangle$
- $b \doteq_S c$ since both share $\langle b, \epsilon\rangle$
- $a \not\doteq_S c$ since they share no context in S

(29) **Def.** A language L is **substitutable** iff $\forall u, v \in \Sigma^*$, $u \doteq v$ implies $u \equiv_L v$. We write $\mathscr{L}_S$ for the substitutable languages.

(30) Clark and Eyraud (2005, p294): "Thus reversibility is the exact analogue of substitutability for regular languages."

(31) **Def.** L is a substitutable context free language (SCFL) iff it is context free and subsitutable. $\mathscr{L}_{SCF}$ for the substitutable CF languages.

(32) **Thm.** If L is substitutable, then $\doteq_L$ is transitive.

Proof: Assume L is substitutable, and for some $u, v, w \in \Sigma^*$, $u \doteq v$, and $v \doteq w$. By the def of 'substitutable', then, $u \equiv v$, and $v \equiv w$, and so $u \equiv w$. That means that for all $l, r \in \Sigma^*$, $lur \in L$ iff $lwr \in L$. By our assumptions we know that there is some context in which u occurs (one that it shares with v), and so now we know that w shares that context too. That is, $u \doteq w$.

(33) **Thm.** If L is substitutable, then $\doteq_L$ is an equivalence relation.

(34) **Thm.** If L is substitutable and $S \subseteq L$, it is possible that S is not substitutable.
Proof: Consider the set $S = \{ab, bb, bc\}$ mentioned in (28). We showed there that this set is not substitutable. (Notice also that $S \notin \mathscr{L}_{0-rev}$.) But $S \subseteq \Sigma^*$, and obviously Σ^* is substitutable. For every $u, v \in \Sigma^*$, $u \doteq_{\Sigma^*} v$ and $u \equiv_{\Sigma^*} v$.

(35) **Thm.** $\{a, aa\} \notin \mathscr{L}_{SCF}$. (Clark and Eyraud, 2005, p293)
Proof.

- $a \doteq_S aa$ since both share $\langle \epsilon, \epsilon \rangle$
- $a \not\equiv_S aa$ since a but not aa has the context $\langle a, \epsilon \rangle$

(36) **Thm.** If L is substitutable and $S \subseteq L$, for any $u, v \in \Sigma*$, $u \doteq_S v$ implies $u \doteq_L v$, and so $u \doteq_S v$ implies $u \equiv_L v$.

(37) **Def.** For any $S \subseteq L$ let $\cong$ be the transitive closure of $\doteq_S$. (Clark and Eyraud, 2005, p287)
Since $\cong_S$ is obviously an equivalence, let $[u]_{\cong_S} = \{v |\ v \cong_S u\}$, leaving off subscripts when they are clear from context.

(38) **Thm.** If L is substitutable and $S \subseteq L$, $u \cong_S v$ implies $u \equiv_L v$,

(39) **Thm.** $\mathscr{L}_{fin} \nsubseteq \mathscr{L}_{SCF}$.
Proof: We saw this in (28) and in (35).

(40) **Thm.** $|\mathscr{L}_{fin} \cap \mathscr{L}_{SCF}| = \infty$
Proof. For example, as mentioned by Clark and Eyraud (2005, p293), it is easy to see that every singleton language $L = \{x\} \in \mathscr{L}_{SCF}$, since no two distinct substrings of x can share any context.

(41) **Thm.** $(\mathscr{L}_{CF} - \mathscr{L}_{reg}) \cap \mathscr{L}_{SCF} \neq \emptyset$.
Proof. For example, as mentioned by C&E, $\{a^n c b^n\} \in \mathscr{L}_{SCF}$.

(42) **Thm.** $\mathscr{L}_{CF} \nsubseteq \mathscr{L}_{SCF}$.
Proof. For example, as mentioned by C&E, $\{a^n b^n\} \notin \mathscr{L}_{SCF}$, as we see by the fact that $a \doteq aab$ since they share the context $\langle \epsilon, b \rangle$, but obviously, $a \not\equiv aab$.

(43) **Exercise.** Is $|(\mathscr{L}_{reg} - \mathscr{L}_{fin}) \cap \mathscr{L}_{SCF}| = \infty$? Establish your answer with a proof.

(44) **Exercise.** Is $|(\mathscr{L}_{CF} - \mathscr{L}_{reg}) \cap \mathscr{L}_{SCF}| = \infty$? Establish your answer with a proof.

5.1.1 A learner ϕ_{SCF}

(45) **Def.** For any $s \in \Sigma^*$, let

$$\text{nesubstring}(s) = \{u \in \Sigma^+ |\ \exists l, r,\ lur = s\}.$$

For any $S \subseteq \Sigma^*$, let

$$\text{nesubstring}(S) = \{u \in \Sigma^+ |\ \exists l, r,\ lur \in S\}.$$

(46) **Def.** We define a function that maps each finite $T[i] \subset \Sigma^*$ to a **substitution graph** SG which pairs substrings with their contexts in the sample:

$$\begin{aligned} \text{SG}(T[i]) = (V, E) \text{ where: } \quad & V = \text{nesubstring}(T[i]) \\ & E = \{(u, v) \in V \times V |\ u \doteq_{T[i]} v\} \end{aligned}$$

(47) **Def.** For any graph $G = (V, E)$, a **component** is a connected subgraph that is not a proper part of any other connected subgraph of G.

(48) **Example.** From (28) consider again $T[3] = \{ab, bb, bc\}$. Then $\text{SG}(T[3]) = (V, E)$ where

$$\begin{aligned} V &= \{a, b, c, ab, bb, bc\} \\ E &= \text{the reflexive,symmetric closure of } \{\langle a, b\rangle, \langle b, c\rangle, \langle ab, bb\rangle, \langle ab, bc\rangle, \langle bb, bc\rangle\} \end{aligned}$$

This graph has 2 components.

(49) **Def.** Given any substitution graph $\text{SG}(T[i])$, let $\text{SG}(T[i])/\ \cong_S$ be the partition of $\text{nesubstring}(T[i])$ induced by $\cong_S$. Each block of this partition is the set of elements in a single component of $\text{SG}(T[i])$.

(50) **Def.** We define a function that maps any substitution graph $\text{SG}(T[i]) = (V, E)$ for any sample from $L \subseteq \Sigma^*$ to a CFG $\langle \Sigma, \text{Cat}, \rightarrowtail, S \rangle$, as follows.[3]

[3] Clark and Eyraud (2007, p.1730) write $\lceil u \rceil_S$ for the component of SG/ $\cong_S$ that contains u, but here I refer to this component with the standard notation $[u]_{\cong_S}$. Clearly, for some finite samples S, it can happen that $[u]_{\cong_S} \neq [u_{\equiv_L}]$.

Cat $= \mathrm{SG}(T[i])/\cong_S$

$\rightarrowtail$ is given by the following set of rules

$$\begin{array}{ll} [a]\rightarrowtail a & \text{for every } a \in \Sigma \\ [vw]\rightarrowtail [v]\,[w] & \text{for all } v, w, vw \in V \end{array}$$

$S = T[i]$

(51) **Continuing example (48).** Considering the graph $\mathrm{SG}(T[3]) = (V, E)$ we see that

$$\begin{array}{ll} [ab] = [bb] = [bc] & \text{for this block, the start symbol, we will use the symbol } [aa] \\ [a] = [b] = [c] & \text{for this block we will use the symbol } [a] \end{array}$$

The rules of the grammar G determined by the graph are the following:

$$[aa]\rightarrowtail [a][a] \qquad \begin{array}{l} [a]\rightarrowtail a \\ [a]\rightarrowtail b \\ [a]\rightarrowtail c \end{array}$$

Clearly, $L(G) = T[i] \cup \{aa, ba, cb, cc\} = \Sigma^2$. It is not hard to see that this language, unlike $T[3]$, is an SCFL.

(52) In our notation, for any distinct nonterminal $a, b \in \Sigma$, if $a \doteq b$, then $[a] = [b]$, and so, for nonterminal a, the following set of rules

$$\{\ [a]\rightarrowtail a,\ [b]\rightarrowtail a\ \}$$

has only 1 element – we've written the same rule twice. So I cannot understand Clark&Eyraud's remark. Of course, we will have both of the following rules:

$$\{\ [a]\rightarrowtail a,\ [a]\rightarrowtail b\ \}.$$

That set contains two elements, since $a \neq b$.

5.1.2 More compact hypotheses: Reduction systems with less redundancy

(53) Clark and Eyraud (2005, p291) say "the algorithm is not practical, since the number of nonterminals will often become very large"

(54) **Exercise.** Is that true? C&E do not quantify this claim. The claim might be puzzling at first, given the demonstration that the algorithm is polynomial. But remember that it is polynomial with respect to the size of $T[i]$. Obviously, as i increases, we would like our grammar to ultimately be much smaller than $T[i]$.

(55) Clark&Eyraud'05 say

> Clearly one can recursively remove all nonterminals that have only one production by replacing the nonterminal on the left hand side of the production with the right had side, wherever it occurs. Secondly, one can remove nonterminals, one by one, and test whether the grammar continues to accept all of the sample, and thus arrive at a minimal CFG. (Clark and Eyraud, 2005, p291)

(56) **Exercise.** Is that a right? That is, applying the second idea, checking against the finite sample, will we end up with a minimal CFG that is weakly equivalent to the one defined in (50)?

(57) (Clark and Eyraud, 2005, p291) say

> In some cases, [a reduction system] is a more natural way of defining the structure of a language than systems from the traditional Chomsky hierarchy.

(58) **Exercise.** Is that true? Couldn't we map the reduction system into a similarly natural grammar that uses nonterminals in the standard way for nonterminal classes? C&E do not mention any results establishing that this kind of reduction system can be significantly (e.g. exponentially?) more succinct than a grammar – are there any such results?

(59) **Def.** A reduction system $T \subseteq \Sigma^* \times \Sigma^*$. $(u, v) \in T$ is often written $u \vdash_T v$.
We extend $\vdash_T$ so that

$$lur \vdash_T lvr \text{ if } u \vdash_T v.$$

And we let $\vdash_T^*$ be the reflexive, transitive closure, as usual.

$$\Rightarrow \text{more coming!} \Leftarrow$$

5.1.3 Assessment

(60) (Clark and Eyraud, 2005, p294): ***

> Importantly, our algorithm does not rely on identifying constituents: that is to say on identifying which substrings have been generated by the non terminals of the target *grammar*. This has up to now been considered the central problem in context-free grammatical inference, though it is in some sense an ill-posed problem since there may be many different grammars with different constituent structures that are nonetheless weakly equivalent, that is to say, define the same languages.

(61) (Clark and Eyraud, 2005, p294): ***

> Ron uses a measure of similarity of residual languages rather than of contexts as we use here. Considered in this way, our measure is very crude, and brittle – contexts are equal if they have non empty intersection. Nevertheless the techniques of Ron et al., suggest a way that this technique could be extended to a PAC-learning result, using a bound on a statistical property of the distribution.

(62) As we saw in (8), English$\notin \mathscr{L}_{\text{SCF}}$.

(63) (Clark and Eyraud, 2007, p.1738) say "The most important subclass of context-free languages that is polynomially identifiable is that of very simple grammars (Yokomori, 2003)", but they do not explain why they think that.

(64) **Exercise.** (Clark and Eyraud, 2007, §7) presents an experiment in support of the view that there is no reason to think that learning English auxiliaries requires any nativist assumptions. Is the argument here persuasive? (In a page or less: Give me your assessment of this argument as persuasively as you can. Don't miss their main points! If any of their claims about formal properties of the situation are not obvious, try to prove them. And if you are unsure about the veracity of any of their claims about previous literature, check some of those too.)

5.2 Learning $\mathscr{L}_{SCF_{k,l}}$ from examples

Yoshinaka (2008) describes a straightforward generalization of the previous idea, analogous to Angluin's generalization of zero-reversible to k-reversible.

(65) **Def.** Recall that $L \in \mathscr{L}_S$ iff

$$x_1y_1z_1, x_1y_2z_1, x_2y_1z_2 \in L \text{ implies } x_2y_2z_2 \in L.$$

Now lets say $L \in \mathscr{L}_{S_{k,l}}$ iff

$$x_1vy_1uyz_1, x_1vy_2uz_1, x_2vy_1uz_2 \in L \text{ implies } x_2vy_2uz_2 \in L,$$

where $|v| = k$, $|u| = l$, and $vy_1u, vy_2u \neq \epsilon$. (Yoshinaka, 2008, p269)

(66) **Def.** $\mathscr{L}_{SCF_{k,l}} = \mathscr{L}_{S_{k,l}} \cap \mathscr{L}_{CF}$

(67) **Thm.** $\mathscr{L}_S = \mathscr{L}_{S_{0,0}}$.

(68) **Thm.** For all $k < k', l < l'$ (Yoshinaka, 2008, p269)

$$\begin{array}{ccc} \mathscr{L}_{S_{k,l}} & \subsetneq & \mathscr{L}_{S_{k',l}} \\ \mathscr{L}_{S_{k,l}} & \subsetneq & \mathscr{L}_{S_{k,l'}} \end{array}$$

(69) **Thm.** $\mathscr{L}_{S_{k,l}}$ is not closed wrt any of: union, intersection with regular sets, concatenation, Kleene closure, ϵ-free homomorphism, inverse homomorphism. (Yoshinaka, 2008, p270)

(70) **Thm.** $\mathscr{L}_{S_{k,l}}$ is closed wrt intersection, ϵ-free inverse homomorphism. (Yoshinaka, 2008, p270)

(71) **Thm.** $\{a^nb^n\} \in \mathscr{L}_{SCF_{1,1}}$. Recall from (42) that $\{a^nb^n\} \notin \mathscr{L}_S$. (Yoshinaka, 2008, p271)

(72) **Thm.** Although $\mathscr{L}_{S_{k,l}}$ is closed wrt intersection, $\mathscr{L}_{SCF_{k,l}}$ is not. (Yoshinaka, 2008, p271)
All the other properties of $\mathscr{L}_{S_{k,l}}$ mentioned in (69) and (70) extend to $\mathscr{L}_{SCF_{k,l}}$.

5.2.1 A learner

(73) **Def.** For any finite $T[i]$, define

$$G_{T[i]} = \langle \Sigma, \text{Cat}, \rightarrowtail, S \rangle, \text{ where}$$

$$\begin{array}{rcl} \text{Cat} & = & \{[y] |\ xyz \in T[i], y \neq \epsilon\} \cup \{S\} \\ \rightarrowtail & = & \{[vyu] \rightarrowtail [vy'u] |\ xvyuz, zvy'uz \in T[i], |v| = k, |u| = l, vyu, vy'u \neq \epsilon\} \\ & & \cup \{S \rightarrowtail [w] |\ w \in T[i]\} \\ & & \cup \{[xy] \rightarrowtail [x][y] |\ [xy], [x], [y] \in \text{Cat}\} \\ & & \cup \{[a] \rightarrowtail a |\ a \in \Sigma\} \end{array}$$

(74) **Def.** $\phi_{SCF_{k,l}}(T[i]) = G_{T[i]}$

(75) **Thm.** $\phi_{SCF_{k,l}}$ identifies $\mathscr{L}_{SCF_{k,l}}$ in the limit from examples.

(76) **Thm.** Every $L \in \mathscr{L}_{SCF_{k,l}}$ has a characteristic set, a locking sequence for $\phi_{SCF_{k,l}}$ with size polynomial wrt target grammar G.

5.3 Learning $\mathscr{L}_{FCP(k)}$ from a membership oracle

A different kind of CFL learner is considered in Clark, Eyraud, and Habrard (2008), which can learn CFLs that have the finite context property and the finite kernel property.

5.3.1 Binary feature grammars

(77) **Thm.** Given any language L, $\forall u, v, u', v'$

$$C_L(u) \subseteq C_L(u') \wedge C_L(v) \subseteq C_L(v') \Rightarrow C_L(uv) \subseteq C_L(u'v') \tag{5.1}$$

This has the corollary, for any $K \subseteq \Sigma^*$ (Clark, Eyraud, and Habrard, 2008, p3)

$$\bigcup_{u'v'=w} \quad \bigcup_{u \in K, C(u) \subseteq C(u')} \quad \bigcup_{v \in K, C(v) \subseteq C(v')} C(uv) \quad \subseteq \quad C(w) \tag{5.2}$$

"This is the basis of our representation: a word w is characterized by its contexts."

(78) **Def.** A **binary feature grammar** (BFG) = $\langle F, f_s, P, P_L, \Sigma \rangle$ where (Clark, Eyraud, and Habrard, 2008, p3)

Σ		a finite nonempty alphabet
F		is a set of features, where $C = 2^F$ the set of all sets of features
f_s	$\in F$	is the sentence feature
P	$\subseteq$	$C \times C \times C$ is a finite set of productions $x \rightarrowtail yz$
P_L	$\subseteq$	$C \times \Sigma$ written $x \rightarrowtail a$

Given any BFG G define the features f_G associated with any string by the grammar:

$$f_G(\epsilon) = \emptyset \tag{5.3}$$

$$f_G(w) = \bigcup_{(c \rightarrowtail w) \in P_L} c \qquad \text{iff } |w| = 1 \tag{5.4}$$

$$f_G(w) = \bigcup_{u,v:uv=w} \quad \bigcup_{x \rightarrowtail yz \in P \ni y \subseteq f_G(u) \wedge z \subseteq f_g(v)} x \qquad \text{iff } |w| > 1 \tag{5.5}$$

(Clark, Eyraud, and Habrard, 2008, p4) say we should note the relation between (5.5) and (5.2).

Equally important, though, is the fact that nothing like (5.1) is enforced in BFGs.

Now define

$$L(G) = \{u | \, f_s \in f_G(u)\}.$$

(79) A Contextual BFG (CBFG) is a BFG in which features F is a finite set of contexts,

$$F \subseteq \Sigma^* \times \Sigma^*.$$

(80) **Def.** Recalling the definition of C_L in (26) on page 89, given a finite set of contexts $F \subseteq \Sigma^* \times \Sigma^*$, (Clark, Eyraud, and Habrard, 2008, p5)

$$F_L(u) = C_L \cap F.$$

(81) **Def.** CBFG G is **exact** iff $\forall u \in \Sigma^*$ (Clark, Eyraud, and Habrard, 2008, p6)

$$f_G(u) = F_{L(G)}(u).$$

That is, a CBFG grammar G is exact iff the features the grammar associates with each string u are exactly the set of contexts the string has in the language when restricted to the set F.

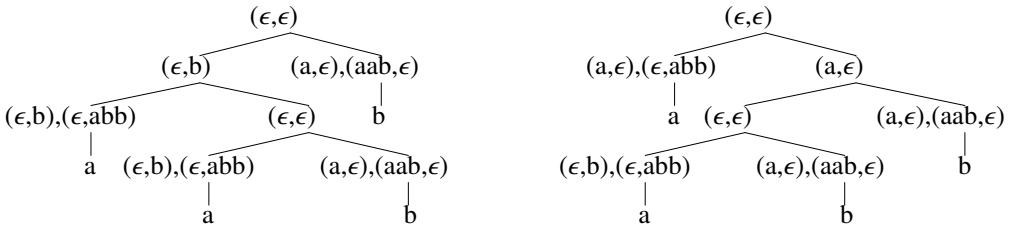
(82) CEH conjecture that $\mathscr{L}_{CBF} \subsetneq \mathscr{L}_{BF}$. (Clark, Eyraud, and Habrard, 2008, p6)

(83) **Example.** Let CBFG $G = \langle \Sigma, F, (\epsilon, \epsilon), P, P_L \rangle$ where

$$\begin{array}{rcl} F & = & \{\ (\epsilon,\epsilon) \\ & & \ \ (a,\epsilon), \\ & & \ \ (aab,\epsilon), \\ & & \ \ (\epsilon,b), \\ & & \ \ (\epsilon,abb)\ \} \\ P_L & = & \{\ \{(\epsilon,b),(\epsilon,abb)\}\rightarrowtail a \\ & = & \ \ \{(a,\epsilon),(aab,\epsilon)\}\rightarrowtail b\ \} \\ P & = & \{\ \{(\epsilon,\epsilon)\}\rightarrowtail\{(\epsilon,b)\}\ \{(aab,\epsilon)\} \\ & & \ \ \{(\epsilon,\epsilon)\}\rightarrowtail\{(\epsilon,abb)\}\ \{(a,\epsilon)\} \\ & & \ \ \{(\epsilon,b)\}\rightarrowtail\{(\epsilon,abb)\}\ \{(\epsilon,\epsilon)\} \\ & & \ \ \{(a,\epsilon)\}\rightarrowtail\{(\epsilon,\epsilon)\}\ \{(aab,\epsilon)\}\ \} \end{array}$$

G is an exact CBFG for $L(G) = \{a^n b^n |\ n > 0\}$.

We can show that $aabb \in L(G)$ with either of the following derivations:



(84) Every BFG has a weakly equivalent non-erasing positive range concatenation grammar (RCG) of arity 1 in 2-var form (Boullier, 1999). (Clark, Eyraud, and Habrard, 2008, p4) For every rule $x\rightarrowtail yz \in P$ and $f \in x$, where $y = \{g_1, \ldots, g_i\}, z = \{h_1, \ldots, h_j\}$ we have the RCG equivalent rules:

$$f(UV)\rightarrowtail g_1(U), \ldots, g_i(U), h_1(V), \ldots, h_j(V).$$

And for every rule $\{f_1, \ldots, f_k\}\rightarrowtail a \in P_L$ we have the RCG equivalent rules (for $1 \leq i \leq k$):

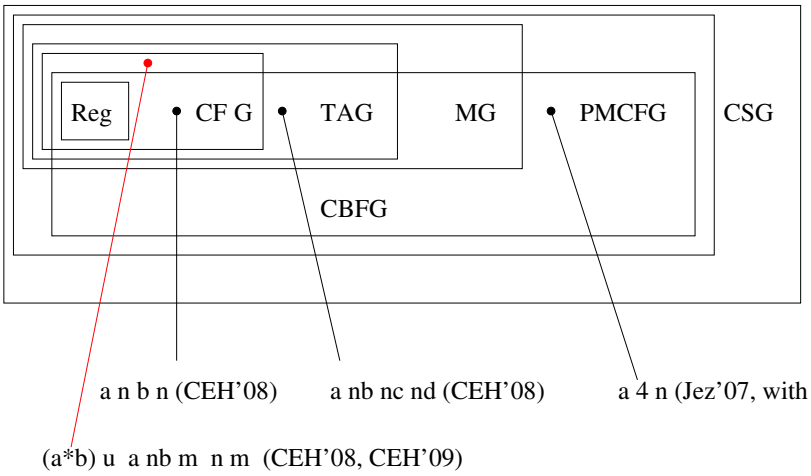
$$f_i(a)\rightarrowtail\epsilon.$$

(85) Clark, Eyraud, and Habrard (2009) define a mapping from an arbitrary conjunctive grammar (ConG) (Okhotin, 2003) defining language L to a CBFL that generates $L\ltimes$.

If their claim about this mapping is correct, then CBFLs include the language $L(G) = \{a^{4^n}\ltimes |\ n \geq 0\}$ since a^{4^n} is defined by the following ConG from Jeż (2007) with start symbol A_1:

$$\begin{array}{ll} A_1\rightarrowtail A_1A_3\&A_2A_2 & A_1\rightarrowtail a \\ A_2\rightarrowtail A_1A_1\&A_2A_6 & A_2\rightarrowtail aa \\ A_3\rightarrowtail A_1A_2\&A_6A_6 & A_3\rightarrowtail aaa \\ A_6\rightarrowtail A_1A_2\&A_3A_3 & \end{array}$$

(86) One cannot help noticing, as Clark does in the video, that the class of CBFLs is approximately what we seem to need for human language. See (91) and footnote 5 on page 98, below.



a n b n (CEH'08)

a nb nc nd (CEH'08)

a 4 n (Jez'07, with transformation from CEH'09)

(a*b) u a nb m n m (CEH'08, CEH'09)

5.3.2 A learner $\phi_{FCP(k)}(T[i])$

Algorithm 1: MakeGrammar

Data: A finite set of strings K, a finite set of contexts F, a finite set of strings D, a non-empty finite set Σ, a bound f
Result: A context-free grammar G
$S = \mathcal{C}(\{(\lambda, \lambda)\})$;
for $A \subseteq F$, $|A| \leq f$ **do**
  $V \leftarrow V \cup \{\mathcal{C}(A)\}$;
end
$P \leftarrow \emptyset$;
for *each* $a \in \Sigma \cup \{\lambda\}$ **do**
  for *each* $N \in V$, $N = \langle S_N, C_N \rangle$ **do**
    if $a \in S_N$ **then**
      $P \leftarrow P \cup \{N \rightarrow a\}$;
    end
  end
end
for *each* $A, B \in V$ **do**
  $J = S_A S_B$;
  $S_X \leftarrow$ `GetK(GetF(`J`))` ;
  for *each* $N \in V$ **do**
    if $S_X \subseteq S_N$ **then**
      $P \leftarrow P \cup \{N \rightarrow AB\}$;
    end
  end
end
return $G = \langle \Sigma, V, P, S \rangle$;

Algorithm 2: CFG learning algorithm

Data: Input alphabet Σ, bounds k, f
Result: A sequence of CFGs $G_1, G_2, \ldots$
$K \leftarrow \Sigma \cup \{\lambda\}$, $K_2 = K$;
$F \leftarrow \{(\lambda, \lambda)\}$, $E = \{\}$;
$D = (F \odot KK) \cap L$;
$G =$ `Make(` K, D, F,f`)` ;
repeat
  $w =$ `GetPositiveExample`; **if** *there is some* $w \in E$ *that is not in* $L(G)$ **then**
    $F \leftarrow Con(E)$;
    $K \leftarrow K_2$;
    $D = (F \odot KK) \cap L$;
    $G =$ `Make(`K, D, F,f`)` ;
  end
  else
    $D_2 \leftarrow (F \odot K_2K_2) \cap L$;
    if $\langle K_2, D_2, F \rangle$ *not isomorphic to* $\langle K, D, F \rangle$ **then**
      $K \leftarrow K_2$;
      $D = (F \odot KK) \cap L$;
      $G =$ `Make(`K,D,F,f`)`;
    end
  end
  Output G;
until ;

from Clark (2010b), extending the simpler proposal in Clark, Eyraud, and Habrard (2008)

(87) **Def.** The simple algorithm IIL($T[i]$): (implementation in Appendix)

1. Let $T = Con(T[i]) \odot Sub(T[i])$ – combine contexts and substrings in all possible ways to get a test set
2. Beginning with empty sets of substrings K and contexts F, for every $w \in T$
   a. If $w \in L$ but not in current hypothesis, add strings to K and add contexts to F
   b. If $w \notin L$ but is in current hypothesis, add contexts to F

(88) **Def.** $u \in L$ has the finite context property iff there is a finite set of contexts $F_u \subseteq C(u)$ such that for any v, if $F_u \subseteq C(v) \subseteq C(v)$, so we will not overgeneralize

(89) **Def.** $K \subseteq \Sigma^*$ is a kernel for L iff there is a set of feature $L(\mathfrak{B}(K, F, L)) \supseteq L$, so we will not undergeneralize.[4]

(90) $\mathscr{L}_{FCP(k)}$ includes all regular languages, all substitutable languages, but not all context free languages.

(91) Clark&al conjecture that human languages are (weakly) included in the class of CBFLs with the FB,FK properties.[5]

5.3.3 Assessment: The larger picture

To understand the learner of §5.3.2, it is useful to adopt a broader perspective, hinted in Clark, Eyraud, and Habrard (2008), and developed further in Clark (2009, 2010b).

(92) For every language $L \subseteq \Sigma^*$ we can define a lattice $\mathfrak{B}(L)$ of syntactic concepts, defined as follows. For any set of strings $S \subseteq \Sigma^*$ and any set of contexts $C \subseteq \Sigma^* \times \Sigma^*$

$$
\begin{array}{rcll}
S' & = & \{(l,r) | \ \forall w \in S, \ lwr \in L\} & \text{'substrings' of } L \\
C' & = & \{w | \ \forall w \in S, \ lwr \in L\} & \text{'contexts' of } L \\
\langle S, C \rangle & \text{where} & C' = S, S' = C & \text{'concepts' of } L \\
\langle S_1, C_1 \rangle \leq \langle S_2, C_2 \rangle & \text{iff} & S_1 \subseteq S_2 & \\
\langle S_x, C_x \rangle \circ \langle S_y, C_y \rangle & = & \langle (S_x S_y)'', (S_x S_y)' \rangle & \\
A \rightarrowtail BC & \text{iff} & A \geq B \circ C &
\end{array}
$$

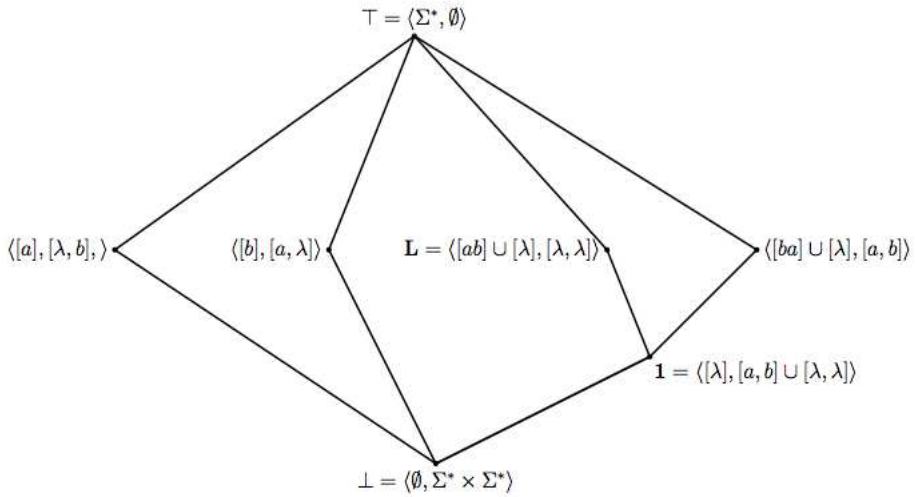


Fig. 1. The Hasse diagram for the syntactic concept lattice for the regular language $L = \{(ab)^*\}$. Each concept (node in the diagram) is an ordered pair of a set of strings, and a set of contexts. We write $[u]$ for the equivalence class of the string u, $[l, r]$ for the equivalence class of the context (l, r).

(93) This lattice is finite iff L is regular, so we restrict ourselves to a finite set of contexts $F \subseteq \Sigma^* \times \Sigma^*$ and consider the finite lattice $\mathfrak{B}(L, F)$ restricted to 2^F.

[4] (Clark, Eyraud, and Habrard, 2008, Def 11, p8) say "for any", where I say "there is a." I think my wording expresses their intention more clearly.

[5] This is suggested in Clark, Eyraud, and Habrard (2008, video 26:50). And Clark, Eyraud, and Habrard (2009, p.39) says of the languages definable by CBFGs "we conjecture that the class of natural language stringsets lies in this class."

For example, consider everyone's favourite example of a context free language: $L = \{a^n b^n | n \geq 0\}$. The concept lattice for this language is infinite: among other concepts we will have an infinite number of concepts corresponding to a^i for each integral value of i. If we take the finite set of the six contexts $F = \{(\lambda, \lambda), (\lambda, b), (a, \lambda), (aab, \lambda), (\lambda, abb), (\lambda, ab)\}$, then we will end up with the finite lattice shown on the left hand side of Figure 2.

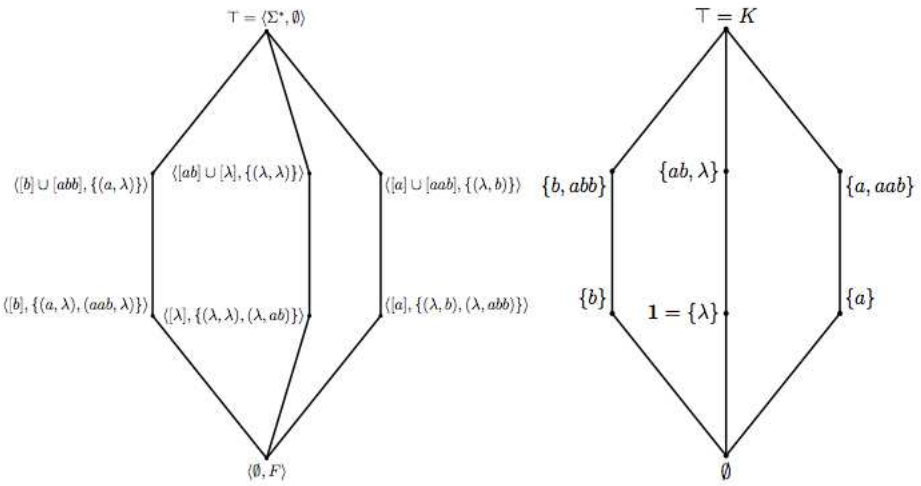


Fig. 2. The Hasse diagram for the partial lattice for $L = \{a^n b^n | n \geq 0\}$, with 6 contexts. The top element $\top$ contains all the strings, and the bottom element contains no strings. The empty string λ is below the language concept, as it has the context (λ, ab). On the left we have the true lattice; on the right we have the lattice as inferred from a small set of strings.

(94) For any such lattice $\mathfrak{B}(L, F)$ and finite set of features F, we can define a mapping $\phi : \Sigma^* \to \mathfrak{B}(L, F)$ from strings to their contexts in F. (This can be computed with standard parsing methods.) Then obviously

$$L(\mathfrak{B}(L, F)) = \{w | \ \phi(w) \leq C((\epsilon, \epsilon))\}$$

(95) The learner gets a finite set of strings K, so consider the lattice $\mathfrak{B}(K, L, F)$ of pairs $\langle S, C\rangle$ such that $S \subseteq K, C \subseteq F$ where $S' \cap F = C$ and $C' \cap K = S$.

(96) **Thm.** As F increases, $L(\mathfrak{B}(K, L, F))$ increases. As K increases, $L(\mathfrak{B}(K, L, F))$ decreases. (Clark, Eyraud, and Habrard, 2008, Lemmas 6,7)

(97) These results motivate the strategy of the learner $\phi_{FCP(k)}(T[i])$ sketched in (87). For example, with

$$\begin{aligned} L &= \{a^n b^n\} \\ F &= \{(\epsilon, \epsilon), (\epsilon, b), (a, \epsilon), (\epsilon, abb), (aab, \epsilon)\} \\ K &= \{\epsilon, a, b, ab, aab, abb\} \end{aligned}$$

the lattice $L(\mathfrak{B}(K, L, F))$ is isomorphic to $L(\mathfrak{B}(L))$ as shown on the right side of Figure 2 above.

(98) **Internal/distributional, primal/dual**

a. Notice that the categorization of the learner $\phi_{FCP(k)}(T[i])$ is calculated from distribution only: any 'internal' similarities among the phrases with a given distribution are ignored.

b. Harris (1941) writes:[6]

> [I]t is pointless to mix phonetic and distributional contrasts. If phonemes which are phonetically similar are also similar in their distribution, that is a result which must be independently proved. For the crux of the matter is that phonetic and distributional contrasts are methodologically different, and that only distributional contrasts are relevant while phonetic contrasts are irrelevant.

[6]These ideas from Harris are discussed in Nevin (1993).

> This becomes clear as soon as we consider what is the scientific operation of working out the phonemic pattern. For phonemes are in the first instance determined on the basis of distribution. Two positional variants may be considered one phoneme if they are in complementary distribution; never otherwise. In identical environment (distribution) two sounds are assigned to two phonemes if their difference distinguishes one morpheme from another; in complementary distribution this test cannot be applied. . . [T]he distributional analysis is simply the unfolding of the criterion used for the original classification. If it yields a patterned arrangement of phonemes, that is an interesting result for linguistic structure.

c. A slightly different perspective is introduced by Clark (2009, 2010a). He calls reasoning about congruences of strings *primal*, and reasoning about contexts *dual*. While most of learners have been primal, the Clark,Eyraud&Habrard learner $\phi_{FCP(k)}(T[i])$ is instead based on a lattice of contexts, and takes the contexts as fundamental.

(99) **Ideas** that I think deserve more exploration

- Hypotheses about human grammars should have consequences for the question of how much context (i.e. how rich a set of features) is needed to define a natural language.
- Ignoring internal structure in inference seems inappropriate: the phrases that are DPs, for example, tend to have Ds in them, etc. Such regularities are ignored by $\phi_{FCP(k)}(T[i])$.

  A similar point has been made about using only transition probabilities in word segmentation Frank et al. (2010).
- If Clark&Eyraud are mistaken to dismiss constituency, we should be able to integrate the ideas in this section with categorizations with additional structure.

5.4 Evidence of constituency

5.4.1 Syntactic cues

(100) As we see in examples (185a-185b) just above, different constituencies do not entail any difference in the strings derived. Consider for example the basic subject-predicate analysis of English (100a, and contrast it with (100b):

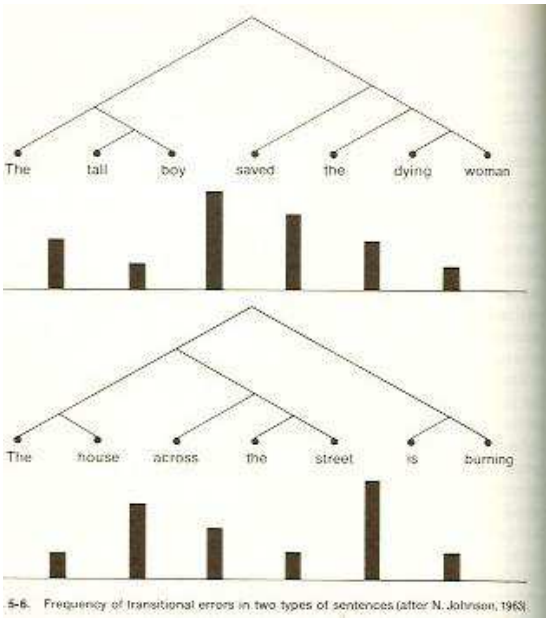
a. Ivan [bought the newspaper]

b. [Ivan bought] the newspaper

(101) The constituency (100b) seems to conflict with various other clues.

a. Chomsky suggests that a wide range of syntactic evidence supports (100a) over (100b), including (i) it fits with binding facts; and (ii) it fits with the fact that object incorporation is often possible while subject incorporation is not, which is, like causative formation, incorporation of the head of the complement. But he suggests that these arguments are only convincing because it is supported by similar evidence from various different languages, but the child is designed so that the [Subject [Verb Object]] constituency is adopted as "a matter of biological necessity" (Chomsky, 1988, pp.53-60).

b. Errors in recall at various points indicate a greater degree at major boundaries (fig from Fodor, Bever, and Garrett (1974, p.250), cf Mehler and Carey (1967)):

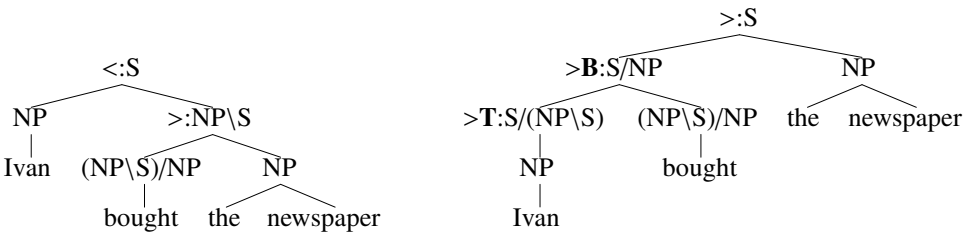


5-6. Frequency of transitional errors in two types of sentences (after N. Johnson, 1963).

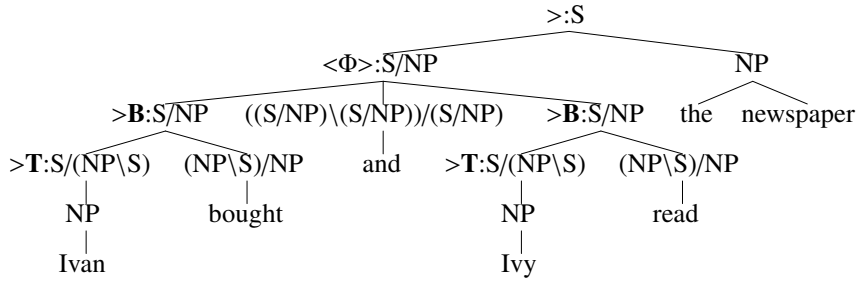
c. Clicks heard while listening to sentences are displaced towards major constituent boundaries (Ladefoged and Broadbent, 1960; Fodor and Bever, 1965).

d. When asked to judge relatedness, verb judged more closely related to the object than to the subject (Levelt, 1970).

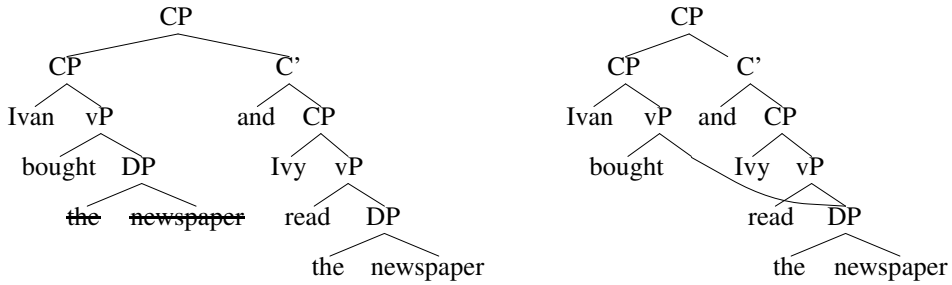
(102) Both constituencies (100a-100b) are allowed in a categorial grammar that has 'type lifting' rules (Steedman, 2000, §3.5):[7]



The main reason for allowing both of these derivations comes from right node raising:



(103) In Chomskian syntax, RNR structures are usually accounted for with either ellipsis or multidominance:



These analyses have the advantage that they do not predict the constituency (100b), but they both face other problems (Larson, 2012).

The type inference in §5.5 requires constituency and headedness to be marked in the data. Does the 'primary linguistic data' (PLD) available to the child include such evidence? That is, besides the 'internal' and 'distributional' evidence mentioned in (98) on page 99, is there 'external' evidence that is not explicit in the sample of strings as usually presented?

5.4.2 Statistical cues

(104) The results mentioned in (101b) and (101d) could signal that constituents have a semantic coherence, but they could also indicate some kind word-cluster coherence, something that might be indicated by word transition probabilities. (Magerman and Marcus, 1990; Cavar, 2010)

(105) Recent artificial grammar learning studies provide some confirmation that people do, in fact, notice this kind of 'statistical coherence' (Takahashi, 2009).

(106) A related idea was used by van Zaanen (2001)

⇒ more coming! ⇐

[7] These derivations are adapted from Steedman (2000, §3.5), but I draw them with trees and with the backslash interpreted in the Lambek style. I use Steedman's labels for forward application >, backward application <, subject type raising >**T**, forward composition >**B** and coordination <Φ>.

5.4.3 Prosodic cues

There is some evidence that speakers tend to provide prosodic clues to constituency when utterances would otherwise be ambiguous. For example, in reading arithmetic expressions, a series of studies by Keating and collaborators shows that prosodic cues are provided similarly across languages (Fougeron and Keating, 1997; Cho and Keating, 2001; Keating et al., 2003). (Wagner, 2005, p29) mentions some similar results in reading logical formulas. However, it remains unclear how available and how reliable this kind of evidence is. Consider this recent work on prosodic cues for alternative PP attachments:

(107) Snedeker and Trueswell (2003) considered examples like this in which the PP can be an NP or VP modifier:

a. Tap the frog [with the flower].

i. 'Tap the frog that has a flower' (NP modifier reading)

ii. 'Tap the frog by using the flower' (VP modifier reading)

They found that speaker use disambiguating prosody only when the context is compatible with both interpretations, and perhaps even then, only when they are aware of the ambiguity.

(108) Kraljic and Brennan (2005) considered examples like this:

a. Put the dog [in the basket] on the star.

i. 'Put the dog that is in the basket onto the star' (NP modifier reading)

ii. 'Put the dog into the basket that is on the star' (VP modifier reading)

In contrast to the Snedeker&Trueswell results, Kraljic&Brennan found that speaker use disambiguating prosody reliably. They propose 3 ideas about why their results differ

a. Unlike the Snedeker&Trueswell study, this one involved interacting with another participant, so interactive disambiguation would be possible if necessary.

b. Producing a single, repetitive structure may have led participants in the Snedeker&Trueswell to ignore prosody.

c. The stimuli in Kraljic&Brennan were longer, possibly prompting the use of prosodic boundaries.

(109) Considering the puzzling differences between the studies just mentioned, Hirsch and Wagner (2012) consider the possiblity that the relevant difference was in the syntax of the stimuli: In (108a) the [in the basket] has the option of forming a constituent with [on the star] to its right. To test whether this is really a factor, Hirsch&Wagner use more complex stimuli like this one, presented in a context supporting one of the 3 possible readings:

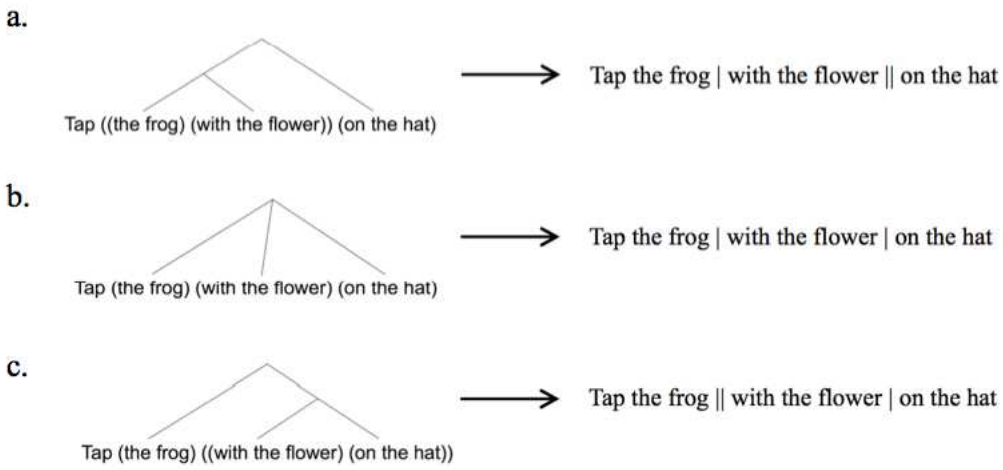
a. Tap the frog [with the flower] on the hat.

i. 'Tap the frog that has a flower and tap it on its hat' ('left', NP attach)

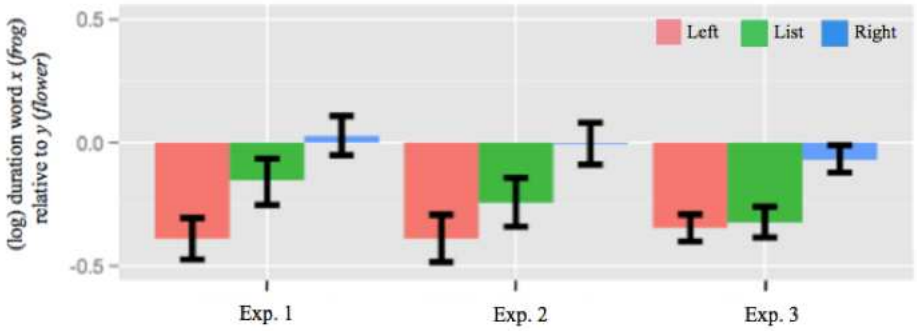
ii. 'Tap the frog by using the flower and tap it on its hat' ('list', VP attach)

iii. 'Tap the frog by using the flower that is on the hat' ('right', [PP PP])

Their predictions were



Preliminary results support these predictions, with awareness (Exp. 1) triggering the clearest disambiguation.



Mean log duration of the word immediately preceding the first PP relative to the word immediately preceding the second PP by condition (left, list, right) for Exp. 1-3.

(110) The results mentioned above suggest that matters are complex and indicate even more clearly that they are not yet well understood, but there is, nevertheless, evidence of various kinds that prosody can be useful in phrase identification (Pate and Goldwater, 2011, for example)

⇒ more coming! ⇐

5.4.4 Semantic cues

It is conceivable that semantic cues could also be used to identify constituency. A variety of 'semantic bootstrapping' proposals can be found in the literature.

(111) The studies mentioned in (101b) suggest that some sort of 'semantic coherence' measure, some kind of 'chunking', could provide more evidence than we see in arbitrary strings over the lexicon.

(112) 'Cross situational learning' aims to find elements of situations that correlate with the presence of utterance constituents (Pinker, 1989; Siskind, 1996; Kobele et al., 2003).

⇒ more coming! ⇐

5.5 Type inference for CFGs

While the learner for reversible languages will conclude from data like `john eats` and `john eats apples`, that `apples` can be iterated any number of times, the learner introduced in this section will not draw that conclusion because `apples` in that sentence does not have the syntactic role of an adjoined modifier. But if, for example, we know that `slowly` is an adjoined modifier in `john eats slowly`, then learner *will* conclude that any number of elements with the same category as `slowly` can be adjoined at the end of the sentence. That is, the learner reasons about the categorial structures. This learner, introduced by Kanazawa (1996, 1998), uses *structured* positive data to identify the target language.

5.5.1 Categories

In categorial grammars, the categories have a regular structure that indicates their combinatorial options.

(113) Let the "basic types" be some finite set *BaseCat*, and then we let the categories be all the "types" that can be formed from these with forward and backward slashes, $CL(BaseCat, \{/, \backslash, \cdot\})$.

(114) Following Lambek (1958), Moortgat (1988) and others, we can think of each type as denoting a set of strings (or other structures). Given any set of vocabulary elements V, we let each "basic type" denote a subset of V^*. Then, the denotations for any $\mathtt{c}_1, \mathtt{c}_2 \in \wp(V^*)$, can be defined as follows:

$$[\![\mathtt{c}_1 \cdot \mathtt{c}_2]\!] = \{\mathtt{s}_1\mathtt{s}_2 \in V^* |\ \mathtt{s}_1 \in [\![\mathtt{c}_1]\!],\ \mathtt{s}_2 \in [\![\mathtt{c}_2]\!]\}$$
$$[\![\mathtt{c}_1/\mathtt{c}_2]\!] = \{\mathtt{s}_1 \in V^* |\ \forall \mathtt{s}_2 \in [\![\mathtt{c}_2]\!],\ \mathtt{s}_1\mathtt{s}_2 \in [\![\mathtt{c}_1]\!]\}$$
$$[\![\mathtt{c}_2\backslash\mathtt{c}_1]\!] = \{\mathtt{s}_1 \in V^* |\ \forall \mathtt{s}_2 \in [\![\mathtt{c}_2]\!],\ \mathtt{s}_2\mathtt{s}_1 \in [\![\mathtt{c}_1]\!]\}$$
$$Cat = CL(BaseCat, \{\cdot, /, \backslash\})$$

(115) With these definitions we can establish that right function application (<) and left function application (>) are not just arbitrary categorial rules, on a par with $S \rightarrow NP\ VP$. Rather, we can show that they are *sound*, in the sense that show that, in the case of <, if an expression is in $[\![(c_1/c_2) \cdot c_2]\!]$ then it is also in $[\![c_1]\!]$, and similarly for >.

(116) **Thm.** For all $c_1, c_2 \in Cat$, $[\![(c_1/c_2) \cdot c_2]\!] \subseteq [\![c_1]\!]$.

Proof: Suppose $s \in [\![(c_1/c_2) \cdot c_2]\!]$. By the definition of $\cdot$, there are strings $s_1 \in [\![(c_1/c_2)]\!]$, and $s_2 \in [\![c_2]\!]$ such that $s = s_1 s_2$. By then by the definition of /, since $s_2 \in [\![c_2]\!]$, $s_1 s_2 \in [\![c_1]\!]$. □

(117) **Thm.** For all $c_1, c_2 \in Cat$, $[\![c_2 \cdot (c_2 \backslash c_1)]\!] \subseteq [\![c_1]\!]$.

Proof: Suppose $s \in [\![c_2 \cdot (c_2 \backslash c_1)]\!]$. By the definition of $\cdot$, there are strings $s_1 \in [\![c_2]\!]$ and $s_2 \in [\![(c_2 \backslash c_1)]\!]$ such that $s = s_1 s_2$. By then by the definition of \, since $s_1 \in [\![c_2]\!]$, $s_1 s_2 \in [\![c_1]\!]$. □

(118) Obviously, it is not in general the case that everything in a set c is a concatenation of strings from other categories, so these results would not hold if $\subseteq$ were replaced by $\supseteq$.

(119) Taking some particular strings and categories for example, if we know

the ∈ [[(s/p)/n]]	
the ∈ [[(r\p)/n]]	
shines ∈ [[p]]	warms ∈ [[r]]
golden ∈ [[n/n]]	cool ∈ [[n/n]]
sun ∈ [[n]]	cat ∈ [[n]]

then it follows that [[s]] includes infinitely many strings:

the sun shines	the golden golden sun shines
the golden sun shines	the sun warms the cool cool cat
...	

(120) In the the lattice of §5.3.3, we can define a similar 'residuation'. Recalling (92), if $X = \langle S_x, C_x \rangle$ and $Y = \langle S_y, C_y \rangle$, then let (Clark, Eyraud, and Habrard, 2008, Def3)

$$X/Y = C(C_x \odot (\epsilon, S_y))$$
$$Y\backslash X = C(C_x \odot (S_y, \epsilon))$$

where for contexts $(l, r) \odot (l', r') = (ll', rr')$ Then we have: (Clark, Eyraud, and Habrard, 2008, Def3,Lemma1)

$$Y \leq X\backslash Z \text{ iff } X \circ Y \leq Z \text{ iff } X \leq Z/Y.$$

(121) If we trade in the symbols of *BaseCat* in for any others, so long as we do it one for one, the grammar is not changed in any substantial respect. The new grammar is sometimes called an *alphabetic variant* of the original. To be precise, we can define a notion of substitution, as in van Benthem (1989); Buszkowski and Penn (1990); van Benthem (1991, etc). Thinking of a grammar as a pairing of vocabulary elements with categories, we can apply substitutions pointwise to whole grammars

$$\begin{aligned} \mathcal{G}_1 = \{ \quad & \langle \texttt{the}, (\texttt{s}/\texttt{p})/\texttt{n} \rangle, & \\ & \langle \texttt{the}, (\texttt{r}\backslash\texttt{p})/\texttt{n} \rangle, & \\ & \langle \texttt{shines}, \texttt{p} \rangle, & \langle \texttt{warms}, \texttt{r} \rangle, \\ & \langle \texttt{golden}, \texttt{n}/\texttt{n} \rangle, & \langle \texttt{cool}, \texttt{n}/\texttt{n} \rangle, \\ & \langle \texttt{sun}, \texttt{n} \rangle, & \langle \texttt{cat}, \texttt{n} \rangle \ \}. \end{aligned}$$

5.5.2 Substitutions and unification in logic

(122) In logic, two expressions unify with each other just in case there is a substitution of terms for variables that makes them identical. To unify human(montague) and human(X) we substitute the term montague for the variable X. We will represent this substitution by the expression {X↦montague}. Letting θ = {X↦montague}, we have

$$\texttt{human(X)}\theta = \texttt{human(montague)}\theta = \texttt{human(montague)}.$$

Notice that the substitution θ has no effect on the term human(montague) since this term has no occurrences of the variable X.

(123) We can replace more than one variable at once. For example, we can replace X by s(Y) and replace Y by Z. Letting θ = {X↦s(Y),Y↦Z}, we have:

$$\texttt{sum(X,Y,Y)}\theta = \texttt{sum(s(Y),Z,Z).}$$

Notice that the Y in the first term has not been replaced by Z. This is because all the elements of the substitution are always applied simultaneously, not one after another.

(124) After a little practice, it is not hard to get the knack of finding substitutions that make two expressions identical, if there is one. These substitutions are called (most general) **unifiers**, and the step of finding and applying them is called **unification.**[8] The value of unification in (automated) logical inference was first noticed by Robinson (1965), developing the basic idea from the doctoral thesis of Herbrand (1930) that proofs can be sought in a syntactic domain defined by the language of the theory.

(125) To describe unification we need two preliminaries. **First**, we need to be able to recognize subexpressions. E.g. in:

$$\mathit{whats\_it}(s(Y, r(Z, g(Var))), \mathit{func}(g(\mathit{Argument}), X), W),$$

the subexpression beginning with *whats_it* is the whole expression;
the subexpression beginning with s is $s(Y, r(Z, g(Var)))$;
the subexpression beginning with *Argument* is *Argument*;
no subexpression begins with a parenthesis or comma.

Second, we need to be able to **compose** substitutions – we need to be able to build up substitutions by, in effect, applying one to another. Remember that substitutions are specified by expressions of the form

$$\{\texttt{V}_1 \mapsto \texttt{t}_1, \ldots, \texttt{V}_n \mapsto \texttt{t}_n\}$$

where the $\texttt{V}_i$ are distinct variables and the $\texttt{t}_i$ are terms $(\texttt{t}_i \neq \texttt{V}_i)$ which are substituted for those variables.

(126) **Def.** The **composition** of substitutions η, θ is defined as follows: Suppose

$$\eta = \{\texttt{X}_1 \mapsto \texttt{t}_1, \ldots, \texttt{X}_n \mapsto \texttt{t}_n\}$$

$$\theta = \{\texttt{Y}_1 \mapsto \texttt{s}_1, \ldots, \texttt{Y}_m \mapsto \texttt{s}_m\}.$$

The composition of η and θ, $\eta\theta$, is

$$\begin{aligned}\eta\theta = \quad & \{\texttt{X}_1 \mapsto (\texttt{t}_1\theta), \ldots, \texttt{X}_\texttt{n} \mapsto (\texttt{t}_\texttt{n}\theta), \texttt{Y}_1 \mapsto \texttt{s}_1, \ldots, \texttt{Y}_\texttt{m} \mapsto \texttt{s}_\texttt{m}\} - \\ & (\{\texttt{Y}_\texttt{i} \mapsto \texttt{s}_\texttt{i} \mid \texttt{Y}_\texttt{i} \in \{\texttt{X}_1, \ldots, \texttt{X}_\texttt{n}\}\} \cup \{\texttt{X}_\texttt{i} \mapsto \texttt{t}_\texttt{i}\theta \mid \texttt{X}_\texttt{i} = \texttt{t}_\texttt{i}\theta\}).\end{aligned}$$

(127) That is, to compute $\eta\theta$, first apply θ to the terms of η and then add θ itself, and finally remove any of the θ variables that are also η variables and remove any substitutions of variables for themselves. Clearly, with this definition, every composition of substitutions will itself satisfy the conditions for being a substitution. Furthermore, since the composition $\eta\theta$ just applies θ to η, $A(\eta\theta) = (A\eta)\theta$ for any expression A.

(128) **Example.** Let

$$\eta = \{\texttt{X}_1 \mapsto \texttt{Y}_1, \texttt{X}_2 \mapsto \texttt{Y}_2\}$$

$$\theta = \{\texttt{Y}_1 \mapsto \texttt{a}_1, \texttt{Y}_2 \mapsto \texttt{a}_2\}.$$

Then

$$\eta\theta = \{\texttt{X}_1 \mapsto \texttt{a}_1, \texttt{X}_2 \mapsto \texttt{a}_2, \texttt{Y}_1 \mapsto \texttt{a}_1, \texttt{Y}_2 \mapsto \texttt{a}_2\}.$$

And, on the other hand,

$$\theta\eta = \{\texttt{Y}_1 \mapsto \texttt{a}_1, \texttt{Y}_2 \mapsto \texttt{a}_2, \texttt{X}_1 \mapsto \texttt{Y}_1, \texttt{X}_2 \mapsto \texttt{Y}_2\}$$

Since $\eta\theta \neq \theta\eta$, we see that composition is thus not commutative, although it is associative.

(129) **Example.** Let

$$\eta = \{\texttt{X}_1 \mapsto \texttt{Y}_1\}$$

$$\theta = \{\texttt{Y}_1 \mapsto \texttt{X}_1\}.$$

Then although neither η nor θ is empty, $\eta\theta = \theta$ and $\theta\eta = \eta$.

[8] In the computational linguistics literature (not to mention politics or physics), the term 'unification' is used in lots of ways. It will not always have the sense defined here. So-called "unification grammars" involve a related, but more elaborate notion of unification.

(130) **Example.** Let

$$\eta = \{\}$$

$$\theta = \{\mathtt{Y}_1 \mapsto \mathtt{X}_1\}.$$

Then $\eta\theta = \theta\eta = \theta$.

This empty substitution $\eta = \{\}$ is called an "identity element" for the composition operation.[9]

(131) Now we are ready to present a procedure for unifying two expressions E and F to produce a unifier $mgu(E, F)$.

Unification algorithm:

1. Put $k = 0$ and $\sigma_0 = \{\}$.
2. If $E\sigma_k = F\sigma_k$, stop. σ_k is a mgu of S. Otherwise, compare $E\sigma_k$ and $F\sigma_k$ from left to right to find the first symbol at which they differ. Select the subexpression E' of E that begins with that symbol, and the subexpression F' of F that begins with that symbol.
3. If one of E', F' is a variable V and one is a term t, and if V does not occur as a (strict) subconstituent of t, put $\sigma_{k+1} = \sigma_k\{V \mapsto t\}$, increment k to $k + 1$, and return to step 2. Otherwise stop, S is not unifiable.

(132) This algorithm produces a most general unifier (mgu) which is unique up to a renaming of variables, otherwise it terminates and returns the judgment that the expressions are not unifiable.[10]

(133) This unification algorithm, if implemented literally, is not efficient. The number of steps it requires increases as an exponential function of the sizes of the terms to be unified. This can be seen by considering this example from Bibel (1982, p.92). Suppose we want to compute the most general unifier of these two terms, which contain the variables $X_0, X_1, \ldots, X_n$:

$$mgu(p(X_1, X_2, \ldots, X_n), p(f(X_0, X_0), f(X_1, X_1), \ldots, f(X_{n-1}, X_{n-1}))) = \ldots$$

Beginning from the left, we see that we need $X_1 \mapsto f(X_0, X_0)$, and applying this first step, the problem is now

$$mgu(p(f(X_0, X_0), X_2, \ldots, X_n), p(f(X_0, X_0), f(f(X_0, X_0), f(X_0, X_0)), f(X_2, X_2), \ldots, f(X_{n-1}, X_{n-1}))) = \ldots$$

Proceeding then to the next argument of p, we need to add (by composing with the first step) the substitution $X_2 \mapsto f(f(X_0, X_0), f(X_0, X_0))$. Proceeding in this way, it is clear that the terms are growing quickly! The last argument has $2^n - 1$ occurrences of f! Fortunately, there are more efficient unification algorithms (Martelli and Montanari, 1982; Knight, 1989). So this is a case where the most intuitive approach is inefficient, but there are efficient ways to handle the problem. We will soon see problems (e.g. in §5.5.8 below!) that have inefficient intuitive algorithms, and where we have good mathematical reasons to believe that there is no efficient algorithm.

5.5.3 Substitutions and unification in categorial grammar

(134) For the purpose of comparing various CGs with substitutions, the designated start category and terminal symbols are constants; / and \ are function symbols, and the elements of *BaseCat* (other than the start category) are variables. Then, the standard treatment of substitutions and unification can be adapted to categories. Then we have, for example,

$$mgu(\mathtt{c}_1\backslash\mathtt{c}_2, \mathtt{c}_3) = \{c_3 \mapsto (\mathtt{c}_1\backslash\mathtt{c}_2)\},$$

where none of these categories are the designated start category. The categories $\mathtt{c}_1\backslash\mathtt{c}_2$ and $\mathtt{c}_3/\mathtt{c}_4$, on the other hand, cannot be unified.

From this perspective, the distinctive thing about a CG is not *Cat*, but the relations among the categories assigned to lexical items. As noted earlier, we can apply any substitution to this entire set of pairs to obtain another grammar, **a substitution instance** of the first grammar.

(135) **Thm.** If $\mathcal{G}_1\theta \subseteq \mathcal{G}_2$ for any substitution θ and any grammars $\mathcal{G}_1, \mathcal{G}_2$ with designated start symbol $\mathtt{s}$, then $[\![\mathtt{s}]\!]_{\mathcal{G}_1} \subseteq [\![\mathtt{s}]\!]_{\mathcal{G}_2}$.

Proof: Consider any derivation tree of $\mathcal{G}_1$ with root $\mathtt{s}$. Applying any substitution to the categories in this tree, the result is clearly a derivation tree in $\mathcal{G}_2$ with root $\mathtt{s}$ and with the same yield. □

[9] In algebra, when we have a binary associative operation on a set with an identity element, we say we have a **monoid**. So the set of substitutions, the operation of composition, and the empty substitution form a monoid. Another monoid we will discuss below is given by the set of sequences of words, the operation of appending these sequences, and the empty sequence.

[10] Our presentation of the unification algorithm is based on Lloyd (1987), where this result about the algorithm is established as Thm 4.3. The result also appears in the classic source, Robinson (1965).

(136) Intuitively, a substitution can "enrich" a category, replacing some basic categories by complex ones, but it cannot "impoverish" the categories. Let's say that two grammars $\mathcal{G}_1$ and $\mathcal{G}_2$ are **alphabetic variants** iff there is a 1-1 substitution of basic categories for basic categories θ such that $\mathcal{G}_1\theta = \mathcal{G}2$. Then one perspective on the intuition about substitutions enriching categories is provided by the following result:

(137) **Thm.** If $\mathcal{G}_1\theta_1 = \mathcal{G}_2$ and $\mathcal{G}_2\theta_2 = \mathcal{G}_1$ then $\mathcal{G}_1$ and $\mathcal{G}_2$ are alphabetic variants.

Proof: Suppose $\mathcal{G}_1\theta_1 = \mathcal{G}_2$ and $\mathcal{G}_2\theta_2 = \mathcal{G}_1$. Let θ be the result of deleting from θ_1 all the substitutions for categories that do not occur in $\mathcal{G}_1$. Clearly, θ and θ_1 will have the same effect on $\mathcal{G}_1$: $\mathcal{G}_1\theta = \mathcal{G}_1\theta_1 = \mathcal{G}_2$. We establish the theorem by showing that θ is a 1-1 substitution of basic categories for basic categories.

Assume for contradiction that θ is not a 1-1 substitution of basic categories for basic categories. That is, either (i) θ maps some basic category of $\mathcal{G}_1$ to a complex category, or (ii) θ maps basic categories to basic categories but is not 1-1. Each case leads to a contradiction.

(i) Suppose θ maps some basic category c of $\mathcal{G}_1$ to a complex category. But then there cannot be any θ_2 such that $\mathcal{G}_2\theta_2 = \mathcal{G}_1$ because no substitution can replace a complex category by a simpler one.↯

(ii) Suppose θ maps basic categories c_1 and c_2 of $\mathcal{G}_1$ both to some category c. Then again there cannot be any θ_2 such that $\mathcal{G}_2\theta_2 = \mathcal{G}_1$ because no substitution can replace c by both c_1 and c_2.↯ □

(138) CGs related by substitutions will be of considerable interest in §5.5.6.

5.5.4 Elasticity

(139) Consider the following sets:

$\mathbb{N} = \{0, 1, 2, \ldots\}$ the set of natural numbers
$\mathcal{E}ven$ = the set of finite subsets of even numbers, $\{0, 2, 4 \ldots, \}$
$\mathcal{O}dd$ = the set of finite subsets of odd numbers, $\{1, 3, 5 \ldots, \}$

Clearly $\mathscr{L}_1 = \mathcal{O}dd \cup \mathcal{E}ven \cup \{\mathbb{N}\}$ is identifiable. All the sets in $\mathcal{O}dd \cup \mathcal{E}ven$ are finite, learnable with the simple conservative induction by enumeration ϕ_e, mentioned in the discussion of Gold's basic results. We extend this learner just so that if it gets any text with both an even number and an odd number, it guesses $\mathbb{N}$.

The set $\{L_1 \cup L_2 |\ L_1, L_2 \in \mathscr{L}_1\}$, on the other hand, contains all finite subsets of $\mathbb{N}$ together with $\mathbb{N}$ itself – a class of languages that we know is not identifiable. In this case, seeing an even number and an odd number does not generally indicate that the learner should generalize, and so learning is impossible.

This is worrying. In our account of human language learning, we want to allow not only for the possibility of learning English and Chinese, but for the possibility of learning both at once. A simple model of the multilingual environment is provided by taking unions of the various languages, and we see that this can make the learnability problem significantly harder, or impossible. We address this problem by studying some learnable classes with the properties that the class of unions of their elements is also identifiable. Of course, the problem we have set here is highly idealized, abstracting away from very many important aspects of multilingual language acquisition. The justification for the idealization is that it leads us to some insights about learning might work. In particular, it suggests some interesting ideas about how a learner could go about classifying lexical items and generalizing to structures that have not been seen.

Following Kanazawa (1996, 1998), we begin with another corollary of Angluin's basic theorem, one that draws attention to an idea that will be of some interest.

(140) **Def** A class of languages $\mathscr{L}$ **has a limit point** L iff there are languages $L_0, L_1, L_2, \ldots \in \mathscr{L}$ such that

$$L_0 \subset L_1 \subset L_2 \subset \ldots$$

and $L \in \mathscr{L}$ is such that

$$L = \bigcup_{n \in \mathbb{N}} L_n$$

(141) Consider the classes of languages mentioned just above:

$$\mathscr{L}_1 = \mathcal{O}dd \cup \mathcal{E}ven \cup \{\mathbb{N}\}$$
$$\mathscr{L}_2 = \{L_1 \cup L_2 |\ L_1, L_2 \in \mathscr{L}_1\}$$

The class $\mathscr{L}_2$ obviously has a limit point:

$$\{0\} \subset \{0, 1\} \subset \{0, 1, 2\}, \ldots$$

with the limit $\mathbb{N}$. Alternatively,

$$\{0, 1, 2\} \subset \{0, 1, 2, 3, 4, 5\} \subset \{0, 1, 2, 3, 4, 5, 6, 7, 8\}, \ldots$$

with the limit $\mathbb{N}$ – there are infinitely many different infinite sequences of languages from $\mathscr{L}_2$ with this limit point. The class $\mathscr{L}_1$, on the other hand, has no such limit point.

(142) **Corollary.** If $\mathscr{L}$ has a limit point, then it is not identifiable. (Kanazawa, 1998)

Proof: Suppose for contradiction that ϕ learns $\mathscr{L}$, where $\mathscr{L}$ has limit point L. Then by the Blum & Blum theorem on locking sequences, there is a locking sequence t for ϕ and L. Since t is finite and $L = \bigcup_{n\in\mathbb{N}} L_n$, there must be a language $L_i \subset L$ such that $L(t) \subseteq L_i$. But then on any text for L_i that begins with t, ϕ will converge on L and fail to learn L_i.↯ □

(143) What is perhaps more interesting is a related property that will guarantee learnability. Finite elasticity is one...

(144) **Def.** A class of languages $\mathscr{L}$ has **infinite elasticity** if and only if there is a sequence of sentences $s_0, s_1, \ldots$ and a sequence of languages $L_0, L_1, \ldots$ in $\mathscr{L}$ such that for all $n \geq 0$,

$$s_n \notin L_n, \text{ and}$$

$$\{s_0, \ldots, s_n\} \subseteq L_{n+1}$$

A class of languages has **finite elasticity** if and only if it does not have infinite elasticity.

(145) Consider again the classes of languages mentioned just above:

$$\mathscr{L}_1 = \mathcal{O}dd \cup \mathcal{E}ven \cup \{\mathbb{N}\}$$
$$\mathscr{L}_2 = \{L_1 \cup L_2 |\ L_1, L_2 \in \mathscr{L}_1\}$$

Notice that $\mathscr{L}_1$ has infinite elasticity, as witnessed for example by the sequences

0,	2,	4,	6,	...
$\emptyset$	{0},	{0, 2},	{0, 2, 4}	...

The class L_2 also has infinite elasticity, as witnessed by the same sequences. Infinite elasticity does not tell us whether the class is learnable or not. Finite elasticity, on the other hand, guarantees learnability as we will see now.

Wright (1989) uses the notion of elasticity to establish the following additional corollary of Angluin's Subset Theorem.

(146) **Thm.** If $\mathscr{L}$ has finite elasticity, then it is learnable.

Proof: Assume $\mathscr{L}$ is a class as defined in the theorem, a class with finite elasticity. Assume that the elements of Σ^* are ordered alphabetically. We show that each language $L \in \mathscr{L}$ has a distinguished subset D with the properties specified in Angluin's Subset Theorem. Consider an arbitrary $L \in \mathscr{L}$. Define D to be the set

$$\{x | \text{ for some } L_i \in \mathscr{L}, x \text{ is the least sentence in } L - L_i\}$$

Notice that there may be languages $L_j \supseteq L$, in which case $L - L_j = \emptyset$, so intuitively, supersets of L do not contribute any members to D. We now show that D has the properties required for application of Angluin's Theorem. Suppose there is some language $L_i \in \mathscr{L}$ such that $D \subseteq L_i$. Then L_i must be a language such that $L - L_i = \emptyset$; that is $L_i \supseteq L$. Hence $L_i \not\subset L$. If we can show that D is finite, then by Angluin's Subset Theorem, $\mathscr{L}$ is identifiable.

Suppose for contradiction that the set D is infinite. By the definition of D, for every $s_0 \in D$, there is a language $L_0 \in \mathscr{L}$ such that s_0 is the least sentence in $L - L_0$. Since D is infinite, for any s_0 there is a longer sentence s_1 in D, and so then again by the definition of D, there is another language $L_1 \in \mathscr{L}$ such that s_1 is the minimum element of $L - L_1$. Note also that it must be that $s_0 \in L_1$, since otherwise s_0 would be the minimum element of $L - L_1$, since shorter strings come before longer ones in the standard alphabetical order. So in general, for every $s_i \in D$, there is a language $L_{i+1} \in \mathscr{L}$ such that $L - L_{i+1} \neq \emptyset$ and $s_i \in L_{i+1}$. But then, there is an infinite sequence of sentences $s_0, s_1, \ldots$ and an infinite sequence of languages $L_0, L_1, \ldots$ in $\mathscr{L}$ such that

$$s_n \notin L_n, \text{ and}$$

$$\{s_0, \ldots, s_n\} \subseteq L_{n+1}.$$

That is, $\mathscr{L}$ has infinite elasticity.↯ □

(147) **Thm.** (Wright, 1989; Motoki, Shinohara, and Wright, 1989; Kanazawa, 1998) If $\mathscr{L}_1$ and $\mathscr{L}_2$ have finite elasticity, then so does $\{L_1 \cup L_2 |\ L_1 \in \mathscr{L}_1, L_2 \in \mathscr{L}_2\}$.
(See the cited sources for proofs.)

5.5.5 Rigid CGs have finite elasticity

We now turn to a simple class of languages which has finite elasticity. This presentation of results in this section is distilled primarily from Kanazawa (1998).[11]

(148) We can define CGs with the following succinct format, as a set of assignments of categories to vocabulary elements:
Grammar 5: Let this grammar contain just these 3 pairs,

$$\langle \mathtt{a}, \mathtt{s/x} \rangle$$
$$\langle \mathtt{a}, \mathtt{(s/x)/s} \rangle$$
$$\langle \mathtt{b}, \mathtt{x} \rangle$$

Given any such grammar $\mathcal{G}$, V is the set of strings on the left sides of the $\mapsto$, $Cat = CL(BaseCat, \{/, \backslash\})$ where $BaseCat$ is just the simple categories in the grammar. In Grammar 5, $BaseCat = \{s, x\}$. The lexicon $Lex = V$ and the structure building functions are $\mathcal{F} = \{F_{\mathtt{c}} | \ \mathtt{c} \in Cat\} \cup \{<, >\}$, where the functions $F_{\mathtt{c}}$ with non-empty domains are exactly the ones given in the succinct grammar. That is,

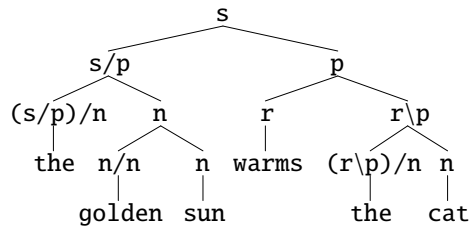
$$Dom(F_{\mathtt{s/x}}) = \{\mathtt{a}\} \qquad F_{\mathtt{s/x}}(\mathtt{a}) = \mathtt{s/x}$$
$$Dom(F_{\mathtt{(s/x)/s}}) = \{\mathtt{a}\} \qquad F_{\mathtt{(s/x)/s}}(\mathtt{a}) = \mathtt{(s/x)/s}$$
$$Dom(F_{\mathtt{x}}) = \{\mathtt{b}\} \qquad F_{\mathtt{x}}(\mathtt{b}) = \mathtt{x}$$

As usual, we can let $L(\mathcal{G}) = CL(Lex, \mathcal{F})$, but what we are interested in is the derivations themselves. For this grammar, there are infinitely many different derivations that establish $\mathtt{s} \in CL(Lex, \mathcal{F})$, and each of the corresponding derivation trees has a different yield. For any $\mathtt{c} \in Cat$, let $[\![\mathtt{c}]\!]$ be the set of yields of derivation trees of $\mathcal{G}$ with root c. So in the case of Grammar 5, $[\![\mathtt{s}]\!] = \{a^n b^n | \ n > 0\}$ and $[\![\mathtt{x}]\!] = \{b\}$.

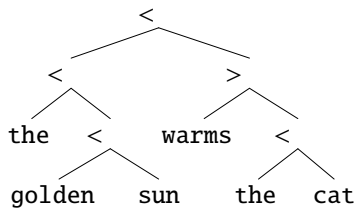
(149) **Def.** A CG is **rigid** iff the set of pairs in the grammar is a function. That is, each lexical item in $Lex = V$ is in the domain of at most one function in $\{F_{\mathtt{c}} | \ \mathtt{c} \in Cat\}$. Each lexical item has at most one category.

(150) Grammar 5 is not rigid, because it assigns two types to a.

(151) The first learning algorithm we will consider accepts as data not strings, but what Buszkowski (1988) calls f-structures, function-argument structures. These are derivation trees in which name of the generating function applied replaces the category at the internal node labels, except nodes that would then be labeled with lexical functions $F_{\mathtt{c}}$ are eliminated. For example, consider the derivation tree,



The corresponding f-structure is



The types are gone, but we know that under any node formed by < we will find two constituents which are naturally regarded as the **function** and **argument**, respectively. Similarly, under any > node we will find the **argument** and **function** respectively.

Obviously, any CG $\mathcal{G}$ defines a set $\mathscr{FL}(\mathcal{G})$ of f-structures of this sort. If the CG has a designated start category s, we will let $\mathscr{FL}(\mathcal{G})$ be just the f-structures corresponding to derivations of s.

[11] See also Kanazawa (1996); de Jongh and Kanazawa (1995).

(152) **Def.** $\mathcal{G}_{rigid}$ is the class of rigid grammars over some particular vocabulary V. When indicated, we can suppose that each grammar comes with a specification of some particular category s as the "start symbol."
$\mathcal{FL}_{rigid}$ is the class of sets of f-structures defined by rigid grammars over V.
$\mathcal{L}_{rigid}$ is the class of sets of strings $[\![\mathtt{s}]\!]$ where s is the start symbol for the rigid grammars over V.

(153) **Thm.** $\mathcal{FL}_{rigid}$ with designated start category s has finite elasticity. (Kanazawa, 1998)

(154) Kanazawa points out that it follows immediately from (153) and (146) that $\mathcal{FL}_{rigid}$ is identifiable.

5.5.6 A learner for $\mathcal{G}_{rigid}$

The following learner from Kanazawa (1998, 1996) accepts a finite text $T[i]$ of f-structures (and possibly also occurrences of the non-linguistic element #). The learner outputs a rigid CG, $RG(t_i)$, in which s is the designated start category.

(155) **Learner** ϕ_{rigid}

Input: f-structures $content(T[i]) = \{F_1, \ldots, F_j\}$

1. i. assign s to the root of each of $F_1, \ldots, F_j$

ii. assign unique base categories to each of the argument nodes of $F_1, \ldots, F_j$

iii. calculate the categories of the function nodes in $F_1, \ldots, F_j$

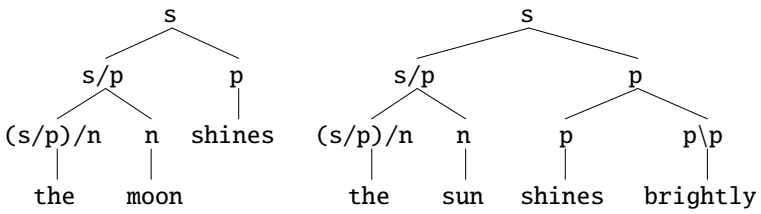
2. Collect the lexical categories from $F_1, \ldots, F_j$ to get a CG $GF(T[i])$.

3. Unify the categories assigned to each vocabulary element to get a rigid CG, $RG(T[i])$. (This will always be possible if T is a text for the language of f-structures of a rigid CG.)

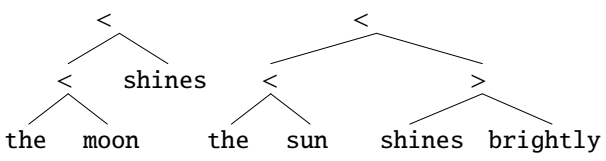
(156) Consider how this learner deals with a text defined by a grammar like the following:

⟨the, (s/p)/n⟩
⟨sun, n⟩
⟨moon, n⟩
⟨shines, p⟩
⟨brightly, p\p⟩

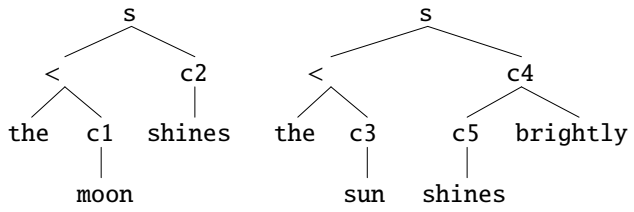
(157) With this grammar we can derive:



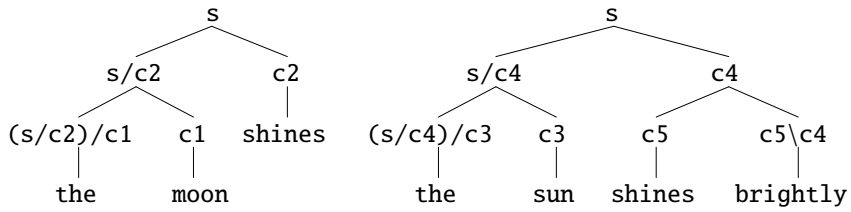
(158) Let's consider how our learner responds to a sequence $T[2]$ containing the two corresponding f-structures:



Assigning s to the root, and unique categories to each argument node, we obtain:



Then, we calculate the categories of the functions in these structures:



Now, letting the categories of the lexical items define a grammar we obtain $RG(T[2])$:
Grammar $GF(T[2])$**:**

$\langle$the, (s/c2)/c1, (s/c4)/c3$\rangle$
$\langle$moon, c1$\rangle$
$\langle$sun, c3$\rangle$
$\langle$shines, c2, c5$\rangle$
$\langle$brightly, c5\c4$\rangle$

The next step is to unify the categories of each lexical item to get a rigid grammar, $RG(T[2])$. Let's do this one lexical item at a time. We can unify the categories of the with the substitution $\{\texttt{c4} \mapsto (\texttt{c2}), \texttt{c3} \mapsto (\texttt{c1})\}$. (Remember, this substitution says, replace c4 by c2 and replace c3 by c4.) Applying this substitution to the grammar we have:

$\langle$the, (s/c2)/c1$\rangle$
$\langle$sun, c1$\rangle$
$\langle$moon, c1$\rangle$
$\langle$shines, c2, c5$\rangle$
$\langle$brightly, c5\c2$\rangle$

Notice that this first step has already assigned both sun and moon to the same category. Now we can unify the categories of shines with the substitution $\{\texttt{c5} \mapsto (\texttt{c2})\}$. Applying this substitution to the grammar we obtain our output
Grammar $RG(t_2)$**:**

$\langle$the, (s/c2)/c1$\rangle$
$\langle$sun, c1$\rangle$
$\langle$moon, c1$\rangle$
$\langle$shines, c2$\rangle$
$\langle$brightly, c2\c2$\rangle$

This is $RG(T[2])$, the output of the learner. Notice that $RG(T[2])$ is an alphabetic variant of the target grammar. On the basis of 2 f-structures, the learner gets to the grammar which generates an infinite set.

(159) Notice that this learner correctly identifies the recursion through the adverbs on the basis of the f-structures of expressions with no more than 1 level of recursion. Lightfoot (1989) calls 1 level of embedding 'degree-1' data. The second tree in the text we just considered comes from a tree that has p-embedding of degree-1. When the class of languages is as simple as this one, allowing languages in which some but not all categories recursive, degree-1 data is required to identify recursion. (Is there a learner who identifies recursion in any infinite class of languages on the basis of degree-0 data?)

(160) Returning to the matters at hand, to get a good understanding of the learner in Algorithm ϕ_{rigid}, let's go through a series of results which lead up to the conclusion that algorithm does, in fact, identify arbitrary rigid CGs in the limit from texts of f-structures. We begin with a simple result, analogous to the earlier (135) presented in §5.5.3.

(161) **Thm.** If $\mathcal{G}_1\theta \subseteq \mathcal{G}_2$ then $\mathscr{FL}(\mathcal{G}_1) \subseteq \mathscr{FL}(\mathcal{G}_2)$. (Buszkowski and Penn, 1990)

Proof: As in the proof of (135), consider any derivation tree $\mathcal{G}_1$ with root $\mathtt{c}$. Applying any substitution to the categories in this tree, the result is clearly a derivation tree in $\mathcal{G}_2$ with root $\mathtt{c}$ and with the same yield. Furthermore, these two trees have the same f-structures. □

(162) The grammar GF does not generalize at all; it defines just the structures provided in the text. By the definition of $GF(T[i])$, for each f-structure in $T[i]$, every argument is assigned a unique basic category, and then the categories of the function nodes are calculated. Consequently, each category can combine only with the node it has as a sister in this step from the f-structure. No category of $GF(T[i])$ occurs more than once in any derivation of $\mathtt{s}$, and no category other than $\mathtt{s}$ occurs in more than one derivation of $\mathtt{s}$.

(163) **Thm.** Where $T[i]$ is a finite sequence of f-structures, $\mathscr{FL}(GF(T[i])) = \mathit{content}(T[i])$. (Buszkowski and Penn, 1990)

Proof: ($\supseteq$) It is clear from the definition of GF that for every f-structure in $T[i]$, $GF(T[i])$ is a grammar with a derivation tree that will have that same f-structure.
($\subseteq$) We use an induction on the depth d of the subtrees in f-structures in $\mathscr{FL}(GF(T[i]))$ to show that all the subtrees in $\mathscr{FL}(GF(T[i]))$ are subtrees in $\mathit{content}(T[i])$, from which it follows that $\mathscr{FL}(GF(T[i])) \subseteq \mathit{content}(T[i])$ since every tree is a subtree of itself.

($d = 0$) Consider any depth 0 subtree in any f-structure in $\mathscr{FL}(GF(T[i]))$, consisting of just a root node labeled with the terminal symbol $\mathtt{w}$. This subtree occurs in an f-structure when there is a pair $\langle \mathtt{w}, \mathtt{c}\rangle \in GF(T[i])$. By the definition of GF, such a rule will occur only when there is a depth 0 subtree in some f-structure in $T[i]$, labeled with $\mathtt{w}$.

(IH) Assume that the result holds up to depth $d = k$.

($d = k + 1$) Consider any depth $k + 1$ subtree in any f-structure in $\mathscr{FL}(GF(t_i))$. This subtree will either be labeled $>$ or $<$.

($>$) If it is labeled $>$, then it dominates two subtrees $t1$ and $t2$ (in that order) of depth $\leq k$, so by (IH) they are both subtrees in $\mathit{content}(T[i])$. Furthermore, by the definition of GF we know that $t1$ corresponds to a derivation of some category $\mathtt{c}_1$ and $t2$ corresponds to a derivation of a corresponding category $\mathtt{c}_1\backslash\mathtt{c}_2$, where these are unique categories introduced in producing a $GF(T[i])$ derivation tree from an element of t_i. This only happens when some tree in $T[i]$ contains subtree in which a node $>$ dominates trees $t1$ and $t2$. Hence this subtree with $>$ dominating $t1$ and $t2$ occurs in $T[i]$.

($<$) The argument for this case is just like the case ($>$), except that the argument subtree $t1$ and the function subtree $t2$ occur in the order: $t2$ $t1$. □

(164) The previous theorem shows that any $T[i]$ determines a GF that generates exactly the f-structures in the text $T[i]$. It is slightly more interesting that the rules of GF all have instances in the target grammar...

(165) **Thm.** $\mathit{content}(T[i]) \subseteq \mathscr{FL}(\mathcal{G})$ iff there is some substitution θ such that $GF(T[i])\theta \subseteq \mathcal{G}$. (Buszkowski and Penn, 1990)

Proof: ($\Leftarrow$) The $\subseteq$ part of (163) tells us that if $\mathcal{G} = GF(T[i])$, then $\mathit{content}(T[i]) \subseteq \mathscr{FL}(\mathcal{G})$. Here we make the slightly stronger claim that if there is any substitution θ such that $GF(T[i])\theta \subseteq \mathcal{G}$, then $\mathit{content}(T[i]) \subseteq \mathscr{FL}(\mathcal{G})$. It is easy to see that this follows. If $\mathcal{G}$ is not $GF(T[i])$, but contains instances of every rule in $GF(T[i])$, $\mathcal{G} \supseteq GF(T[i])\theta$ for some θ, then (as we saw in the proofs of (135) and (161)), the derivation trees of $GF(T[i])$ correspond, under the substitution θ, to derivation trees of $\mathcal{G}$.
($\Rightarrow$) Assume $\mathit{content}(T[i]) \subseteq \mathscr{FL}(\mathcal{G})$, where $\mathit{content}(T[i]) = \{T_0, T_1, \ldots, T_j\}$. For any $0 \leq k \leq j$, let P_k be the (unique) derivation tree of $GF(T[i])$ corresponding to T_k, and let Q_k be the corresponding (unique) derivation tree of $\mathcal{G}$. Let θ be a substitution which maps each basic category in $GF(T[i])$ that labels a node in a P_k to the corresponding category in Q_k. A simple induction shows that this substitution in fact maps all labels in trees P_k to corresponding labels in trees Q_k. It follows that if $\langle(\mathtt{w}, \mathtt{c})\rangle \in GF(T[i])$, then $\langle(\mathtt{w}, \mathtt{c}\theta)\rangle \in \mathcal{G}$. Hence, $GF(T[i])\theta \subseteq \mathcal{G}$. □

(166) The unification step which takes us from the general form grammar $GF(T[i])$ to the rigid grammar $RG(T[i])$ may generalize; that is, the grammar $RG(t_i)$ may define structures that do not appear in the text. Nevertheless, we can be sure that it is still the case that the target grammar contains instances of every rule in $RG(T[i])$:

(167) **Thm.** $\mathit{content}(T[i]) \subseteq \mathscr{FL}(\mathcal{G})$ iff there is some substitution η such that $RG(T[i])\eta \subseteq \mathcal{G}$. (Buszkowski and Penn, 1990)

Proof: ($\Leftarrow$) This follows immediately from the (165) and the fact that $RG(T[i])$ is an instance of $GF(T[i])$.
($\Rightarrow$) Assume $content(T[i]) \subseteq \mathscr{FL}(\mathcal{G})$. By (165), there is a substitution θ such that $GF(T[i])\theta \subseteq \mathcal{G}$. Since $\mathcal{G}$ is rigid, so is $GF(T[i])\theta$. Since $RG(T[i])$ is $GF(T[i])\nu$, where ν is the most general unifier of the categories assigned to each lexical item in $GF(T[i])$, we know that $GF(T[i])\theta$ is an instance of $GF(T[i])\nu$. That is, there is some substitution η such that $\theta = \nu \circ \eta$. Then $RG(T[i])\eta \subseteq \mathcal{G}$. □

(168) We can now identify a distinguished subset $D_{\mathcal{G}_i}$ of the f-structures defined by any grammar $\mathcal{G}_i$.

(169) **Def.** Category c is **useless** in grammar $\mathcal{G}$ with designated start category s iff no derivation tree with root s contains c.

(170) *From here on, we restrict our attention now to rigid grammars with no useless categories: grammars with only categories that occur in successful derivations.*

(171) **Thm.** Since a CG assigns categories to only finitely many lexical items (and all useless categories are eliminated), the CG will have only finitely many categories altogether: the lexical categories and the categories derivable from those by left and right application.

(172) **Def.** For any category c in a grammar $\mathcal{G}$, let $root(\mathsf{c})$ denote a smallest derivation tree with root c.
For any category c in a grammar $\mathcal{G}$, let $leaf(\mathsf{c})$ denote a smallest derivation tree with root s that is complete except for having a leaf c.

(173) In these terms, if we take $root(\mathsf{c})$ and attach it to the leaf labeled c in $leaf(\mathsf{c})$, we obtain a complete derivation tree.

(174) **Def.** For any rigid grammar $\mathcal{G}$ with no useless categories, define the **characteristic subset** $D_{\mathcal{G}} \subseteq \mathscr{FL}(\mathcal{G})$ of f-structures as follows. It is the smallest set containing all of the following elements:

a. for all categories c that appear in derivations from G, the f-structure of the derivation tree that results from attaching $root(\mathsf{c})$ to the leaf labeled c in $leaf(\mathsf{c})$;

b. for all categories c and terminal elements w such that $\langle(\mathsf{w}, \mathsf{c})\rangle \in \mathcal{G}$, the f-structure corresponding to the derivation tree that results from attaching w to the leaf labeled c in $leaf(\mathsf{c})$;

c. for all categories $\mathsf{c}_0/\mathsf{c}_1$, the f-structure of the derivation tree that results from attaching $root(\mathsf{c}_0/\mathsf{c}_1)$ and $root(\mathsf{c}_1)$ respectively to the leaf labeled c_0 in $leaf(\mathsf{c}_\emptyset)$;

d. for all categories $\mathsf{c}_1\backslash\mathsf{c}_0$, the f-structure of the derivation tree that results from attaching $root(\mathsf{c}_1)$ and $root(\mathsf{c}_1\backslash\mathsf{c}_0)$ respectively to the leaf labeled c_0 in $leaf(\mathsf{c}_\emptyset)$;

(175) Notice that the trees introduced into $D_{\mathcal{G}}$ introduced by one condition in this definition may also be introduced by another condition. Roughly, we have just defined $D_{\mathcal{G}}$ so that it has at least one tree with each category, at least one tree with each vocabulary element that can occur in any sentence, at least one application of $<$ involving each category of the form $\mathsf{c}_0/\mathsf{c}_1$, and at least one application of $>$ involving $\mathsf{c}_1\backslash\mathsf{c}_0$.
Some basic properties of any such $D_{\mathcal{G}}$ are obvious...

(176) **Thm.** For any rigid grammar $\mathcal{G}$ with no useless categories, $D_{\mathcal{G}}$ is finite.

(177) **Thm.** For any rigid grammar $\mathcal{G}$ with no useless categories, $D_{\mathcal{G}} \subseteq \mathscr{FL}(\mathcal{G})$.

(178) By now, it is no surprise that any grammar that has $D_{\mathcal{G}_i}$ among its f-structures is one that contains instances of rules of $\mathcal{G}_i$:

(179) **Thm.** $D_{\mathcal{G}_i} \subseteq \mathscr{FL}(\mathcal{G}_j)$ iff there is some substitution θ such that $\mathcal{G}_i\theta \subseteq \mathcal{G}_j$. (Kanazawa, 1998)

Proof: ($\Leftarrow$) The definition of $D_{\mathcal{G}_i}$ guarantees that $D_{\mathcal{G}_i} \subseteq \mathscr{FL}(\mathcal{G}_i)$. Assuming there is some substitution θ such that $\mathcal{G}_i\theta \subseteq \mathcal{G}_j$, it then follows from (161) that $D_{\mathcal{G}_i} \subseteq \mathscr{FL}(\mathcal{G}_j)$.
($\Rightarrow$) Assume $D_{\mathcal{G}_i} \subseteq \mathscr{FL}(\mathcal{G}_j)$. Then G_j has a unique derivation tree for each f-structure in $D_{\mathcal{G}_i}$.
By the definition of $D_{\mathcal{G}_i}$, we know that for each basic category $\mathsf{c} \in \mathcal{G}_i$, the f-structure of $root(\mathsf{c})$ appears as a substructure in some f-structure in $D_{\mathcal{G}_i}$, and $\mathcal{G}_j$ has a unique derivation tree T corresponding to that substructure. Let θ be the substitution which maps every basic category c of $\mathcal{G}_i$ to the category at the root of the corresponding $\mathcal{G}_j$ derivation tree T. We can prove by induction that this substitution is the one we need for the theorem: it maps every element of $\mathcal{G}_i$ to a corresponding element of $\mathcal{G}_j$. □

(180) With the following last preliminary result, the stage is set for the main result of this section, that Algorithm (155) identifies rigid grammars from positive texts of structures.

(181) **Thm.** if $D_{\mathcal{G}} \subseteq content(T[i]) \subseteq \mathscr{FL}(\mathcal{G})$ then $RG(T[i]) = \mathcal{G}$. (Kanazawa, 1998)

Proof: Assume $D_{\mathcal{G}} \subseteq content(T[i]) \subseteq \mathcal{FL}(\mathcal{G})$. By (167), $RG(T[i])$ exists and is such that $RG(T[i])\theta \subseteq \mathcal{G}$ for some θ. We know by (163) that $content(T[i]) \subseteq \mathcal{FL}(GF(T[i]))$, and since $RG(T[i])$ is an instance of $GF(T[i])$, we know by (161) that $content(T[i]) \subseteq \mathcal{FL}(RG(T[i]))$. By our assumption then, $D_{\mathcal{G}} \subseteq \mathcal{FL}(RG(T[i]))$, and so it follows by (179) that there is a substitution η such that $\mathcal{G}\eta \subseteq RG(T[i])$. Therefore, G and $RG(T[i])$ are the same up to alphabetic variance. □

(182) **Thm.**Algorithm (155) identifies $\mathcal{G}_{rigid}$ from structures, and hence also $\mathcal{FL}_{rigid}$ and $\mathcal{L}_{rigid}$. (Kanazawa, 1998)

Proof: Any text of f-structures for $\mathcal{G}$ contains every element of $\mathcal{FL}(\mathcal{G})$. Since $D_{\mathcal{G}} \subseteq \mathcal{FL}(\mathcal{G})$ is finite, there will be some finite j such that $D_{\mathcal{G}} \subseteq L(T[j])$. By (181), for all $i \geq j$, $D_{\mathcal{G}} \subseteq content(T[i]) \subseteq \mathcal{FL}(\mathcal{G})$ and so $RG(T[i]) = \mathcal{G}$. □

(183) Inspecting the line of reasoning that leads to the previous result, we see that none of it relies on the fact that all the grammars in $\mathcal{G}_{rigid}$ use the same (finite nonempty) vocabulary V. With only finitely many categorial possibilities for this finite vocabulary, the class has finite elasticity.

5.5.7 An incremental learner for $\mathcal{G}_{rigid}$

We can define a learner which computes the same guesses about the language as Algorithm (155), but does so "incrementally," on the basis of the previous guess and the next f-structure. At each step, this incremental learner revises its previous guess on the basis of just one f-structure, beginning with the empty grammar $\mathcal{G}$. At every step, given a rigid grammar and an f-structure with designated start category `s`, this learner outputs another (possibly unchanged) rigid CG, $RG(\mathcal{G}, F)$.

(184) **Learner** $\phi_{rigid,inc}$

Input: A rigid CG $\mathcal{G}$ and an f-structure F

1. i. Assign `s` to the root of F

ii. Assign unique base categories to each of the argument nodes of F

iii. Calculate the categories of the function nodes in F

2. Collect the lexical categories from F and add them to the types of $\mathcal{G}$ to get a CG $GF(\mathcal{G}, F)$.

3. Unify the categories assigned to each vocabulary element to get a rigid CG, $RG(\mathcal{G}, F)$.

5.5.8 Learning $\mathcal{L}_{rigid}$ from strings

(185) It is clear that identification of particular grammars from strings is impossible – different grammars will have the same string sets. For example, consider the grammars (185a-185b) which define the same string language:

a. `polaris` $\mapsto$ `s/p`, `shines` $\mapsto$ `p`

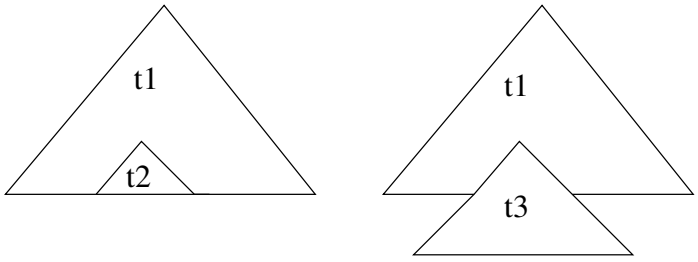
b. `polaris` $\mapsto$ `n`, `shines` $\mapsto$ `n\s`

Obviously, the string sets $[\![\texttt{s}]\!]$ do not identify the grammars they are defined by.

(186) Any string is the yield of only finitely many binary branching trees, and so there are only finitely many possible f-structures for any string altogether, and so also for any finite set of strings. Kanazawa (1998) shows how to go from this observation to learning rigid languages from strings. However, this method is not practical – not only is this particular learning strategy inefficient, but the problem is 'NP-hard' (Costa Florêncio, 2001), which strongly suggests that no algorithm can compute this learning function efficiently.

5.5.9 Reflections and extensions

Suppose we have some vocabulary that is known, and we get a function structure that contains a subtree with some unknown elements. For example, in the following trees, suppose that we know all the categories in trees t1 and t2, and then we get a function structure that has t3 in the place of t2, where t3 has some words the learner has never seen.



In categorial grammars, it is not always the case that the category of the root of t3 is the same as the category at the root of t2,[12] but nevertheless we have a way to determine all the categories in the limit. Furthermore, if only one word is unknown in t3, then the category of the missing element will be unambiguously determined.[13] It does not quite assume that two subtrees in the same context must have the same category, nor does it assume that every type of phrase has an element that is a single word (like pronouns for DPs, intransitive verbs for VPs,...).

5.6 Weights for CFGs

We have not used weights for CF inference in this chapter, but we noted in (61) on page 92 that various ways to do this are already clear from earlier work on regular inference. As in the case of regular automata/grammars, setting weights to fit a sample is trivial when the grammars are unambiguous, and non-trivial otherwise. There is a very substantial literature devoted to setting these weights in such a way that the most probable parse is the one a human would intend in most situations – a goal of obvious engineering interest but of little scientific value if, as seems clear, the determinants of what will be said are largely nonlinguistic.

(187) **Sample weights.** For an unambiguous CFG, we parse the sample, and assign each rule $A \rightarrowtail X$ the probability $cnt(A \rightarrowtail X) / \sum_{Y \in (\Sigma \cup \mathrm{Cat})^*} cnt(A \rightarrowtail Y)$.
But CFGs can be ambiguous, even infinitely ambiguous, so if we parse a sample exhaustively, there can be infinitely many parses. (This is also true for non-deterministic FSAs with ϵ transitions.)

(188) If we have a treebank, we can calculate the value $cnt(A \rightarrowtail X) / \sum_{Y \in (\Sigma \cup \mathrm{Cat})^*} cnt(A \rightarrowtail Y)$ using the preferred parses for each string – that is, the ones in the treebank.

(189) In general, we want to calculate

$$\mathrm{argmax}_{\gamma}\, P(Data|\gamma).$$

When the grammar is ambiguous, how can this be calculated? The 'inside-outside' algorithm is standardly used to estimate it – a 'hill climbing' expectation-maximization (EM) method liable to local maxima (Lari and Young, 1990; Lafferty, 2000). Many implementations are available – e.g. from Mark Johnson's page http://www.cog.brown.edu/ mj/Software.htm

(190) **Smoothed weights.** Short summary from Eisner (2002):

> To smooth the distribution p(RHS|LHS), one can define it in terms of a set of parameters and then estimate those parameters. Most researchers have used an n-gram model (Eisner, 1996; Charniak, 2000) or more general Markov model (Alshawi, 1996) to model the sequence of nonterminals in the RHS. The sequence Vput NP PP in our example is then assumed to be emitted by some Markov model of VPput rules (again with backoff from put). Collins (1997, model 2) uses a more sophisticated model in which all arguments in this sequence are generated jointly, as in a Treebank grammar, and then a Markov process is used to insert adjuncts among the arguments.

Many other ideas are in the literature, cf. e.g. Andrews and Eisner (2011).

[12] One tree could be a 'lifted' verson of the other category, for example.

[13] This kind of reasoning is also similar to the 'type inference' that is done in compiling many programming languages. In fact, the idea of using unification to determine categories ('types') originated there in the work of Hindley (1969) and Milner (1978). This is covered in introductory computer science texts like Pierce (2002, §22).

References

References

Alshawi, Hiyan. 1996. Head automata for speech translation. In *Proceedings of the International Conference on Spoken Language (ICSLP)*.

Andrews, Nicholas and Jason Eisner. 2011. Transformation process priors. In *NIPS Workshop on Bayesian Nonparametrics: Hope or Hype?*, Sierra Nevada, Spain. Extended abstract (3 pages).

van Benthem, Johan. 1989. Categorial grammar meets unification. In *Proceedings Titisee Workshop on Unification Formalisms*, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart.

van Benthem, Johan. 1991. *Language in Action*. North-Holland, Amsterdam.

Bibel, Wolfgang. 1982. *Automated Theorem Proving*. Vieweg & Sohn, Braunschweig.

Booth, Taylor L. and Richard A. Thompson. 1973. Applying probability measures to abstract languages. *IEEE Transactions on Computers*, C-22:442–450.

Boullier, Pierre. 1999. A cubic time extension of context free grammars. Technical Report 3611, Projet Atoll, INRIA, Rocquencourt.

Buszkowski, Wojciech. 1988. Generative power of categorial grammars. In Richard T. Oehrle, Emmon Bach, and D. Wheeler, editors, *Categorial Grammars and Natural Language Structures*. Reidel, Amsterdam.

Buszkowski, Wojciech and Gerald Penn. 1990. Categorial grammars determined from linguistic data by unification. *Studia Logica*, 49:431–454.

Cavar, Damir. 2010. On statistical metrics for selection and phrasality. In T. Hanneforth and G. Fanselow, editors, *Language and Logos*. Akademie Verlag, Berlin.

Charniak, Eugene. 2000. A maximum-entropy inspired parser. In *Proceedings, North American chapter of the Association for Computational Linguistics, NAACL*.

Chi, Zhiyi. 1999. Statistical properties of probabilistic context free grammars. *Computational Linguistics*, 25:130–160.

Chi, Zhiyi and Stuart Geman. 1998. Estimation of probabilistic context free grammars. *Computational Linguistics*, 24:299–306.

Cho, T. and P. Keating. 2001. Articulatory strengthening at the onset of prosodic domains in Korean. *Journal of Phonetics*, 28:155–190.

Chomsky, Noam. 1981. *Lectures on Government and Binding*. Foris, Dordrecht.

Chomsky, Noam. 1988. *Language and Problems of Knowledge: The Managua Lectures*. MIT Press, Cambridge, Massachusetts.

Clark, Alexander. 2009. A learnable representation for syntax using residuated lattices. In *Proceedings of the 14th Conference on Formal Grammar*.

Clark, Alexander. 2010a. Disibutional learning of some context-free languages with a minimally adequate teacher. In *Proceedings of the International Conference on Grammatical Inference, ICGI'10*.

Clark, Alexander. 2010b. Learning context free grammars with the syntactic concept lattice. In José Sempere and Pedro Garcia, editors, *Grammatical Inference: Theoretical Results and Applications. Proceedings of the International Colloquium on Grammatical Inference*. Springer, pages 38–51.

Clark, Alexander and Remi Eyraud. 2005. Identification in the limit of substitutable context free languages. In *Proceedings of The 16th International Conference on Algorithmic Learning Theory*, pages 283–296, Springer-Verlag.

Clark, Alexander and Rémi Eyraud. 2007. Polynomial identification in the limit of substitutable context-free languages. *Journal of Machine Learning Research*, 8:1725–1745.

Clark, Alexander, Rémi Eyraud, and Amaury Habrard. 2008. A polynomial algorithm for the inference of context free languages. In *Proceedings of the International Conference on Grammatical Inference, ICGI 2008*.

Clark, Alexander, Rémi Eyraud, and Amaury Habrard. 2009. A note on contextual binary feature grammars. In *Proceedings of the European Association for Computational Linguistics, Workshop on Computational Linguistic Aspects of Grammatical Inference*, pages 33–40.

Clark, Alexander and Shalom Lappin. 2011. *Linguistic Nativism and the Poverty of the Stimulus*. Wiley-Blackwell, NY.

Collins, Michael. 1997. Three generative, lexicalized models for statistical parsing. In *Proceedings of the 35th Annual Meeting, ACL'96*.

Costa Florêncio, Christophe. 2001. Consistent identification in the limit of the class k-valued is NP-hard. In *Logical Aspects of Computational Linguistics, 4th International Conference, LACL 2001, Le Croisic, France, June 27-29, 2001, Proceedings*, volume 2099 of *Lecture Notes in Computer Science*, pages 125–138, Springer-Verlag.

Eisner, Jason. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings, International Conference on Computational Linguistics, COLING'96*, pages 340–345.

Eisner, Jason. 2002. Transformation priors over grammars. In *Proceedings of the ACL Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 63–70.

Fodor, Jerry A. and Thomas G. Bever. 1965. The psychological reality of linguistic segments. *Journal of Verbal Learning and Verbal Behavior*, 4:414–420.

Fodor, Jerry A., Thomas G. Bever, and Merrill F. Garrett. 1974. *The Psychology of Language: An Introduction to Psycholinguistics and Generative Grammar*. McGraw-Hill, NY.

Fougeron, C. and P. A. Keating. 1997. Articulatory strengthening at edges of prosodic domains. *Journal of the Acoustic Society of America*, 101:3728–3740.

Frank, Michael C., Harry Tily, Inbal Arnon, and Sharon Goldwater. 2010. Beyond transitional probabilities: Human learners impose a parsimony bias in statistical word segmentation. In *Proceedings of the 32nd Annual Conference of the Cognitive Science Society*.

Ghomeshi, Jila, Ray Jackendoff, Nicole Rosen, and Kevin Russell. 2004. Contrastive focus reduplication in English. *Natural Language and Linguistic Theory*, 22(2):307–357.

Gonzalez, Rafael C. and Michael G. Thomason. 1978. *Syntactic Pattern Recognition*. Addison-Wesley, London.

Grenander, Ulf. 1967. Syntax-controlled probabilities. Technical report, Brown University, Providence, Rhode Island.

Harris, Theodore Edward. 1963. *The theory of branching processes*. Springer, Berlin.

Harris, Zellig S. 1941. Review of tubetzkoy's *grundzüge der phonologie*. *Language*, 17(4):345–349.

Harris, Zellig S. 1946. From morpheme to utterance. *Language*, 22:161–183.

Heinz, Jeffrey. 2010. Learning long-distance phonotactics. *Linguistic Inquiry*, 41(4):623–661.

Herbrand, Jacques. 1930. *Recherches sur la théorie de la démonstration*. Ph.D. thesis, University of Paris. Chapter 5 of this thesis is reprinted in Jean van Heijenoort (ed.), From Frege to Gödel: A Source Book in Mathematical Logic, 1879 – 1931. Cambridge, Massachusetts: Harvard University Press.

Hindley, Roger J. 1969. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60.

Hirsch, Aron and Michael Wagner. 2012. Syntactic differences in the reliability of prosodic disambiguation. Forthcoming. McGill University.

Jeż, Artur. 2007. Conjunctive grammars can generate non-regular unary languages. In *Proceedings of the 11th international conference on Developments in language theory*, DLT'07, pages 242–253, Springer-Verlag, Berlin.

Jeż, Artur and Alexander Okhotin. 2009. One-nonterminal conjunctive grammars over a unary alphabet. In Anna Frid, Andrey Morozov, Andrey Rybalchenko, and Klaus Wagner, editors, *Computer Science – Theory and Applications*, volume 5675 of *Lecture Notes in Computer Science*. Springer, pages 191–202.

Johnson, Kyle. 2004. Introduction to transformational grammar. Technical report, University of Massachusetts, Amherst. http://people.umass.edu/kbj/homepage/Content/601_lectures.pdf.

Johnson, N. F. 1965. The psychological reality of phrase structure rules. *Journal of Verbal Learning and Verbal Behavior*, 4:469–475.

de Jongh, Dick and Makoto Kanazawa. 1995. Learnability theory. Distributed at ESSLLI Summerschool, Barcelona.

Kanazawa, Makoto. 1996. Identification in the limit of categorial grammars. *Journal of Logic, Language, and Information*, 5:115–155.

Kanazawa, Makoto. 1998. *Learnable Classes of Categorial Grammmars*. CSLI Publications, Stanford, California.

Keating, Patricia, Taehong Cho, Cécile Fougeron, and Chai-Shune Hsu. 2003. Domain-initial articulatory strengthening in four languages. In R. Temple J. Local, R. Ogden, editor, *Phonetic Interpretation (Papers in Laboratory Phonology 6)*. Cambridge University Press, NY, pages 143–161.

Keenan, Edward L. and Edward P. Stabler. 2003. *Bare Grammar*. CSLI Publications, Stanford, California.

Knight, Kevin. 1989. Unification: a multidisciplinary survey. *ACM Computing Surveys*, 21(1):93–124.

Kobele, Gregory M. 2007. Argument! a reply to Pullum and Rawlins. UCLA, Chicago manuscript, http://home.uchicago.edu/~gkobele/files/2008-Kobele08XOrNoX.pdf.

Kobele, Gregory M., Jason Riggle, Travis Collier, Yoosook Lee, Ying Lin, Yuan Yao, Charles Taylor, and Edward Stabler. 2003. Grounding as learning. In *Language Evolution and Computation Workshop, ESSLLI'03*.

Koopman, Hilda, Dominique Sportiche, and Edward Stabler. 2012. *An Introduction to Syntactic Analysis and Theory*. UCLA Lecture Notes, forthcoming.

Kraljic, Tanya and Susan E. Brennan. 2005. Prosodic disambiguation of syntactic structure: For the speaker or for the addressee? *Cognitive Psychology*, 50(2):194–231.

Ladefoged, Peter and D.E. Broadbent. 1960. Perception of sequence in auditory events. *Quarterly Journal of Experimental Psychology*, 13:162–170.

Lafferty, John. 2000. A derivation of the inside-outside algorithm from the EM algorithm. Technical report RC21636 (97537), IBM.

Lambek, Joachim. 1958. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170.

Lari, K. and S. J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56.

Larson, Bradley. 2012. Right node raising as a test for constituenthood. *Linguistic Inquiry*, 43(1):143–150.

Levelt, Willem J.M. 1970. A scaling approach to the study of syntactic relations. In G.B. Flores d'Arcais and W.J.M. Levelt, editors, *Advances in Psycholinguistics*. Elsevier, NY.

Lightfoot, David. 1989. The child's trigger experience: degree-0 learnability. *Behavioral and Brain Sciences*, 12:321–375.

Lloyd, John W. 1987. *Foundations of Logic Programming*. Springer, Berlin.

Magerman, David M. and Mitchell P. Marcus. 1990. Parsing a natural language using mutual information statistics. In *Proceedings of the 8th National Conference on Artificial Intelligence*.

Manaster-Ramer, Alexis. 1986. Copying in natural languages, context freeness, and queue grammars. In *Proceedings of the 1986 Meeting of the Association for Computational Linguistics*.

Martelli, Alberto and Ugo Montanari. 1982. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282.

Mehler, J. and P. Carey. 1967. Role of surface and base structure in the perception of sentences. *Journal of Verbal Learning and Verbal Behavior*, 6:335–338.

Milner, Robin. 1978. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348–375.

Moortgat, Michael. 1988. *Categorial Investigations*. Foris, Dordrecht.

Motoki, T., T. Shinohara, and K. Wright. 1989. The correct definition of finite elasticity; corrigendum to identification of finite unions. In *The Fourth Annual Workshop on Computational Learning Theory*, page 375, Morgan Kaufmann, San Mateo, California.

Nevin, Bruce E. 1993. Harris the revolutionary: Phonemic theory. In *History of Linguistics 1993: Papers from the Sixth International Conference on the History of the Language Sciences, ICHoLS VI*, Amsterdam Studies in the Theory and History of Linguistic Science, Vol. 78, pages 349–358, John Benjamins, Philadelphia.

Okhotin, Alexander. 2003. An overview of conjunctive grammars. *Bulletin of the European Association for Computer Science (EATCS)*, 79:145–163.

Pate, John and Sharon Goldwater. 2011. Unsupervised syntactic chunking with acoustic cues: Computational models for prosodic bootstrapping. In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics (CMCL), 49th Annual Meeting of the Association for Computational Linguistics*.

Pierce, Benjamin C. 2002. *Types and Programming Languages*. MIT Press, Cambridge, Massachusetts.

Pinker, Steven. 1989. *Learnability and Cognition: The acquisition of argument structure*. MIT Press, Cambridge, Massachusetts.

Pullum, Geoffrey K. and K. Rawlins. 2007. Argument or no argument? *Linguistics and Philosophy*, 30(2):277–287.

Robinson, J. A. 1965. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12:23–41.

Siskind, Jeffrey Mark. 1996. A computational study of cross-situational techniques for learning word-to-meaning mappings. *Cognition*, 61:39–91.

Snedeker, Jesse and John Trueswell. 2003. Using prosody to avoid ambiguity: Effects of speaker awareness and referential context. *Journal of Memory and Language*, 48(1):103–130.

Stabler, Edward P. 2004. Varieties of crossing dependencies: Structure dependence and mild context sensitivity. *Cognitive Science*, 93(5):699–720.

Stabler, Edward P. 2012. Learnability and the autonomy of syntactic categories. In *Forthcoming*.

Steedman, Mark J. 2000. *The Syntactic Process*. MIT Press, Cambridge, Massachusetts.

Takahashi, Eri. 2009. *Beyond Statistical Learning in the Acquisition of Phrase Structure*. Ph.D. thesis, University of Maryland.

van Zaanen, Menno. 2001. Alignment-based learning. *ArXiv*. http://arxiv.org/abs/cs/0104006.

Wagner, Michael. 2005. *Prosody and Recursion*. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Wetherell, C.S. 1980. Probabilistic languages: A review and some open questions. *Computing Surveys*, 12(4):361–379.

Wright, K. 1989. Identification of unions of languages drawn from an identifiable class. In *The Second Annual Workshop on Computational Learning Theory*, pages 328–333, Morgan Kaufmann, San Mateo, California.

Yokomori, Takashi. 2003. Polynomial-time identification of very simple grammars from positive data. *Theoretical Computer Science*, 298:179–206.

Yoshinaka, Ryo. 2008. Identification in the limit of k,l-substitutable context-free languages. In *Proceedings of the 9th International Colloquium on Grammatical Inference, ICGI-2008*, Lecture Notes in Artificial Intelligence 5278, Springer-Verlag, Berlin.

5.7 *Appendix:* First implementation of IIL

```
"""
file: cky.py    E. Stabler, 2012-11-25

Simple CKY parser for CFGs in Chomsky normal form.
We separate the lexical productions (PL) from the non-lexical ones (P),
and the non-lexical ones are always binary A->BC, represented as (A,B,C)
"""
#PL0 = [('A',0),('B',1)]
#P0 = [('S','A','B'),('S','A','R'),('R','S','B')]
#w0 = [0,1]
#w0 = [0,0,1,1]
#w0 = [0,1,1]
#w0 = [0,0,0,1,1,1]

def initializeChart(w,PL,chart): # insert lexical items w=w1 w2 ... wn
    for (i,wi) in enumerate(w):            # enumerate numbers the elements of w from 0
        for (lhs,rhs) in PL:
            if rhs == wi:
                chart[i][i+1].append(lhs)  # so if lhs -> w1, lhs in chart[0][1]

def closeChart(P,chart,n):
    for width in range(2,n+1):
        for start in range(n-width+1):     # so then range stops with n-width
            end = start + width
            for mid in range(start+1,end): # so then range stops with end-1
                for (lhs,y,z) in P:
                    if y in chart[start][mid] and z in chart[mid][end]:
                        chart[start][end].append(lhs)

def accepts((PL,P),w):
    n = len(w)
    chart = [ [ [] for i in range(n+1) ] for j in range(n+1) ]
    initializeChart(w,PL,chart)
    closeChart(P,chart,n)
    print 'chart=',chart
    if 'S' in chart[0][n]:
        return True
    else:
        return False

#print accepts((PL0,P0),w0)
```

```
"""
file: ckyBFG.py    E. Stabler, 2012-11-25
  almost exactly like standard cky parsing, except that for each substring,
  instead of collecting its categories,
  the categories are sets and we take their union,
  coming from all decompositions.
See Clark,Eyraud&Habrard'09 "A note on contextual binary feature grammars"

This differs from ckyCBFG.py only in the pretty printer
"""
```

```
def subset(c1,c2):
    for x1 in c1:
        if not(x1 in c2):
            return False
    return True

def initializeChart(w,PL,chart): # insert lexical items
    for (i,wi) in enumerate(w):  # enumerate numbers elements of w from 0
        wiCat = []
        for (lhs,rhs) in PL:
            if rhs == wi:
                chart[i][i+1].extend(lhs)

def closeChart(P,chart,n):
    for width in range(2,n+1):
        for start in range(n-width+1):     # so then range stops with n-width
            end = start + width
            for mid in range(start+1,end): # so then range stops with end-1
                for (lhs,y,z) in P:
                    if subset(y,chart[start][mid]) and subset(z,chart[mid][end]):
                        chart[start][end].extend(lhs)

def accepts((PL,P),w):
    n = len(w)
    chart = [ [ [] for i in range(n+1) ] for j in range(n+1) ]
    initializeChart(w,PL,chart)
    closeChart(P,chart,n)
    if ([],[]) in chart[0][n]:
        return True
    else:
        return False


## example grammar for {a^nb^nc^nd| n>0}
PL0 = [
    ( ["A","A'"] , 'a' ),  ( ["B"] , 'b' ),
    ( ["C","C'"] , 'c' ),  ( ["D"] , 'd' )
    ]
P0 = [
    ( ["S"]          , [ "AB'","BC'" ]        , [ "D" ] ),
    ( ["AB'"]        , [ "AB" ]               , [ "C'" ] ),
    ( ["AB"]         , [ "AAB" ]              , [ "B" ] ),
    ( ["AB"]         , [ "A" ]                , [ "B" ] ),
    ( ["AAB"]        , [ "A" ]                , [ "AB" ] ),
    ( ["BC'"]        , [ "A'" ]               , [ "BC" ] ),
    ( ["BC"]         , [ "BBC" ]              , [ "C" ] ),
    ( ["BC"]         , [ "B" ]                , [ "C" ] ),
    ( ["BBC"]        , [ "B" ]                , [ "BC" ] ),
    ( ["A'"]         , [ "A'" ]               , [ "A" ] ),
    ( ["C'"]         , [ "C'" ]               , [ "C" ] )
    ]

w0 = ['a','b','c','d']
#w0 = ['a','a','b','b','c','c','d']
#w0 = ['a','b','b','c','c','d'] # should be false
#w0 = ['a','a','a','b','b','b','c','c','c','d']
```

```
## BEGIN pretty printer for chart
def prettyChart(c):
    print
    for (r,row) in enumerate(c):
        for (c,col) in enumerate(row):
            if len(col)>0:
                print (r,c),"=",",".join(col)
## END pretty printer for chart

def ckyChart((PL,P),w): # like accepts, but shows chart
    n = len(w)
    chart = [ [ [] for i in range(n+1) ] for j in range(n+1) ]
    initializeChart(w,PL,chart)
    closeChart(P,chart,n)
    prettyChart(chart)
    if 'S' in chart[0][n]:
        return True
    else:
        return False

print ckyChart((PL0,P0),w0)
```

```
"""
file: ckyCBFG.py    E. Stabler, 2012-11-25
  almost exactly like standard cky parsing, except that for each substring,
  instead of collecting its categories,
  the categories are sets and we take their union,
  coming from all decompositions.
See Clark,Eyraud&Habrard'09 "A note on contextual binary feature grammars"

This differs from ckyBFG.py only in the pretty printer
"""

def subset(c1,c2):
    for x1 in c1:
        if not(x1 in c2):
            return False
    return True

def initializeChart(w,PL,chart): # insert lexical items
    for (i,wi) in enumerate(w):  # enumerate numbers elements of w from 0
        wiCat = []
        for (lhs,rhs) in PL:
            if rhs == wi:
                chart[i][i+1].extend(lhs)

def closeChart(P,chart,n):
    for width in range(2,n+1):
        for start in range(n-width+1):     # so then range stops with n-width
            end = start + width
            for mid in range(start+1,end): # so then range stops with end-1
                for (lhs,y,z) in P:
                    if subset(y,chart[start][mid]) and subset(z,chart[mid][end]):
                        chart[start][end].extend(lhs)
```

```
def accepts((PL,P),w):
    n = len(w)
    chart = [ [ [] for i in range(n+1) ] for j in range(n+1) ]
    initializeChart(w,PL,chart)
    closeChart(P,chart,n)
    if ([],[]) in chart[0][n]:
        return True
    else:
        return False

## example grammar for {a^nb^n| n>0}
PL0 = [
    ( [ ([],['b']), ([],['a','b','b']) ] , 'a' ),
    ( [ (['a'],[]), (['a','a','b'],[]) ] , 'b' )
    ]
P0 = [
    ( [ ([],[]) ]    , [ ([],['b']) ]         , [ (['a','a','b'],[]) ] ),
    ( [ ([],[]) ]    , [ ([],['a','b','b']) ] , [ (['a'],[]) ] ),
    ( [ ([],['b']) ] , [ ([],['a','b','b']) ] , [ ([],[]) ] ),
    ( [ (['a'],[]) ] , [ ([],[]) ]            , [ (['a','a','b'],[]) ] )
    ]

#w0 = ['a','b']
#w0 = ['a','a','b','b']
w0 = ['a','b','b']
#w0 = ['a','a','a','b','b','b']

## BEGIN pretty printer for chart
def prettyContext((lc,rc)):
    sep = " "   # put space if some lex items have > 1 character
    plc = sep.join(map(str,lc))
    prc = sep.join(map(str,rc))
    result = "(" + plc + "," + prc + ")"
    return result

def prettyChart(c):
    print
    for (r,row) in enumerate(c):
        for (c,col) in enumerate(row):
            if len(col)>0:
                print (r,c),"=",",".join(map(prettyContext,col))
## END pretty printer for chart

## BEGIN pretty printer for grammar
def prettyGrammar((PL,P)):
    for (lhs,rhs) in PL:
        print '\t',','.join(map(prettyContext,lhs)),' -> ',rhs
    for (lhs,y,z) in P:
        print '\t',','.join(map(prettyContext,lhs)),' -> ',','.join(map(prettyContext,y)),'   ',','.join(m
## END pretty printer for grammar

def ckyChart((PL,P),w): # like accepts, but shows chart
    n = len(w)
    chart = [ [ [] for i in range(n+1) ] for j in range(n+1) ]
    initializeChart(w,PL,chart)
    closeChart(P,chart,n)
```

```
    prettyChart(chart)
    if ([],[]) in chart[0][n]:
        return True
    else:
        return False

#prettyGrammar((PL0,P0))
#print ckyChart((PL0,P0),w0)
```

```
"""
file: IIL.py   naive implementation of Clark,Eyraud&Habrard'08 algorithm
E. Stabler  2012-11-25
"""

""" if this directory is not in PYTHONPATH, add it with these:
from sys import path
path.append('/path/to/this/directory')
"""
import ckyCBFG
import cky

def add_nesubstrings(sofar,w):  # non-empty substrings of w added to list sofar
    max=len(w)
    for i in range(max):
        for j in range(max+1):
            if i<j:
                s=w[i:j]
                if not(s in sofar):
                    sofar.append(s)
    sofar.sort()

def add_all_nesubstrings(sofar,s): # non-empty substrings of strings in s added
    for w in s:
        add_nesubstrings(sofar,w)
    sofar.sort()

"""
x = []
add_all_nesubstrings(x,[['the','cat','smiles'],[0,1]])
print x
"""

def add_contexts(sofar,w):  # contexts of substrings of w added to list sofar
    max=len(w)
    for i in range(max):
        for j in range(max+1):
            if i<j:
                lc=w[0:i]
                rc=w[j:max]
                c=(lc,rc)
                if not(c in sofar):
                    sofar.append(c)
    sofar.sort()

def add_all_contexts(sofar,s): # contexts for every w in s
    for w in s:
```

```
        add_contexts(sofar,w)
    sofar.sort()

"""
x = []
add_contexts(x,['the','cat','smiles'])
print x
"""

def odot((lc,rc),string):
    product=lc[:]
    product.extend(string)
    product.extend(rc)
    return product

def odotCartesianProduct(C,S):
    product=[]
    for c in C:
        for s in S:
            new = odot(c,s)
            product.append(new)
    return product

def FL(F,w,target):
    ok = []
    for c in F:
        cw = odot(c,w)
        accepted = cky.accepts(target,cw)
        print 'For FL, oracle says ',accepted,':',cw
        if accepted:
            ok.append(c)
    return ok

def g(K,F,target): # from CEH'08, Def.7 on p.6
    PL=[]
    P=[]
    for w in K:
        lhs = FL(F,w,target) # oracle called for this
        if len(w)==1:
            rhs = w
            rule = (lhs,rhs)
            if not(rule in PL):
                PL.append(rule)
        else:
            lenw=len(w)
            for i in range(lenw):
                if 0<i and i<lenw:
                    fl1=FL(F,w[0:i],target) # oracle called for this
                    fl2=FL(F,w[i:lenw],target) # oracle called for this
                    rule = (lhs,fl1,fl2)
                    if not(rule in P):
                        P.append(rule)
    return(PL,P)

def iil(S,target):  # target grammar is for oracle computation
    K = []
```

```
    D = []
    F = [([],[])]
    (PL,P) = g(K,F,target)
    T = []
    for w in S:
        print 'adding input:', w
        D.append(w)
        ConD = []
        add_all_contexts(ConD,D)
        SubD = []
        add_all_nesubstrings(SubD,D)
        T = odotCartesianProduct(ConD,SubD)
        N = len(T)
        print 'test set T now has ',N,' elements'
        for s in T:
            inLanguage = cky.accepts(target,s)
            print 'In main, oracle says ',inLanguage,':',s
            byHypothesis = ckyCBFG.accepts((PL,P),s)
            print 'In main, hypothesis says ',byHypothesis,':',s
            if inLanguage and not(byHypothesis):
                print 'adding K,F for ',s
                K = SubD
                F = ConD
            elif byHypothesis and not(inLanguage):
                print 'adding F for ',s
                F = ConD
            else:
                print 'nothing to be done for ',s
        (PL,P) = g(K,F,target)
        print 'grammar=',(PL,P)
        ckyCBFG.prettyGrammar((PL,P))
    return g(K,F,target)

""" for {0^n1^n| n>0}
sample0 = [[0,1],[0,0,1,1],[0,0,0,1,1,1]]
PL0 = [('A',0),('B',1)]
P0 = [('S','A','B'),('S','A','R'),('R','S','B')]
cky.accepts((PL0,P0),[0,1])
iil(sample0,(PL0,P0))
"""

""" from Koopman,Sportiche,Stabler ISAT chapter 3
"""
sample0 = [
    ['she','will','put','it','there','then'],
    ['she','will','put','this','one','there','then'],
    ['she','will','put','it','on','this','one','then'],
    ['this','one','will','put','it','there','then'],
    ['this','girl','in','the','red','coat','will','put','a','picture','of','Bill','on','your','desk','befo
    ['she','will','put','a','picture','of','Bill','on','your','desk','before','tomorrow'],
    ['this','girl','in','the','red','coat','will','put','it','on','your','desk','before','tomorrow'],
    ['this','girl','in','the','red','coat','will','put','a','picture','of','Bill','there','before','tomorr
    ['this','one','will','put','a','picture','of','Bill','on','your','desk','before','tomorrow'],
    ['this','girl','in','the','red','coat','will','put','a','picture','of','Bill','on','it','before','tomo
    ['this','girl','in','the','red','one','will','put','a','picture','of','Bill','on','your','desk','befor
    ]
```

```
PL0 = [('DP','she'),
       ('DP','it'),
       ('DP','Bill'),
       ('DP','tomorrow'),
       ('D','this'),
       ('D','the'),
       ('D','a'),
       ('D','your'),
       ('A','red'),
       ('NP','one'),
       ('N','girl'),
       ('N','picture'),
       ('N','coat'),
       ('N','desk'),
       ('P','in'),
       ('P','of'),
       ('P','on'),
       ('P','before'),
       ('T','will'),
       ('V','put'),
       ('D','a'),
       ('P','of'),
       ('PP','there'),
       ('PP','then'),
      ]
P0 = [('S','DP','TP'),
      ('DP','D','NP'),
      ('DP','D','N'),
      ('NP','N','PP'),
      ('PP','P','DP'),
      ('N','A','N'),
      ('NP','A','NP'),
      ('TP','T','VP'),
      ('VP','VP','PP'),
      ('VP','V','V2P'),
      ('V2P','DP','PP')
      ]
for w in sample0:
    ok = cky.accepts((PL0,P0),w)
    print ok,':',w
"""
iil(sample0,(PL0,P0))
"""
```

Possible squib topics

If you have any doubts about whether your topic is close enough to class material to make me happy, check with me first! The squib is due, by email, by midnight of the last day of finals week, **Friday Dec 14**. I will be around through finals week, so feel free to come see me if you get stuck on something!

- One ideal kind of squib topic would be a learning problem you are interested in but that we did not get to. If the problem is hard, one thing you could do would be to write a description of the problem and a proposal about how it can effectively be attacked.
- Another ideal kind of squib topic would be to go back and fill in your background on some aspect of the material that we covered too quickly. My suggestions below of exercises that were skipped cover some such things, but you could choose any reasonable exercise on any aspect of material we covered (or something closely related).
- Another appropriate squib topic would be a brief report on a paper (or part of a paper) we did not get to. We did not get to all of the papers on the website, but obviously the literature is enormous. If you choose this option, please check with me first – I can help you assess how close your topic is to class material.
- Another appropriate squib topic would be an implementation and some test runs of one (or more) of the learners we have discussed, with a brief report of what you do. If you choose this option, please send me the source code files so I can run them too when I read your report.

Here are some particular suggestions coming out of the notes.

(191) **Gold paradigm basics: Heinz's ϕ_{prec} learner.** We briefly discussed exercise 5 on page 27 of ho1 in class. (i) Do that exercise, (ii) briefly summarize and assess Heinz's claim about that learner, and consider the prospects for answering Meaghan Fowlie's worry that ϕ_{prec} learns dissimilation effects that are not subject to intervention, so maybe that learner does not capture the right generalization. Heinz discusses this in Heinz (2010, §4.6).

(192) **PAC learning.** Do exercise (23) on page 37 of ho2.

(193) **PAC learning.** Do exercise (28) on page 37 of ho2.

(194) **Angluin'82 k-reversible.** In ho3, the zero reversible learner is presented, but not the k-reversible learner. (i) Define the k-reversible, (ii) define a language L that is k-reversible but not 0-reversible, (iii) show the steps of the k-reversible identifying your language L, as we did for the 0-reversible learner.

(195) **Clark&Thollard'04 PDFA**. Fill out my sketch of an example run in §4.7.2 in ho4. (Or sketch a run of that algorithm on some other example, a small enough example for us to understand what is happening at each step.)

(196) **Clark&Eyraud'04 Substitutable CF**. The language $a^n b^n$ is not SCF, but $a^n c b^n$ is. Is there a way to augment English (e.g. with boundary markers of some kind, like c in that simple example) so that this 'augmented English' would be substitutable, with no counterexamples like the ones from Keenan&Stabler on p.86 of ho5? Does thinking about how such boundaries could be identified suggest a more general learning strategy?

(197) **Clark&Eyraud'04 Substitutable CF**. Apply the SCF learner to an example slightly more interesting than the one in (51) of ho5.

(198) **Clark,Eyraud,Habrard'08 CF with finite kernel**. Sketch an application of the learner (87) on page 98 of ho5, from CEH'08, to a simple example, or run an experiment with the code in the appendix and report what you did.

Or implement the slightly more elaborate learner from Clark'10, on page 97.

Index