AD-A037 129 BROWN UNIV PROVIDENCE R I F/G 5/7
NETWORK PATTERN PROCESSORS, (U)
DEC 76 U. GRENANDER N00014-75-C-0461
UNCLASSIFIED N/L

| OF |
AD
A037129

END
DATE
FILMED
7-78
DDC

Network Pattern Processors

by

Ulf Grenander

(being chapter 7 of a forthcoming book: "Lectures on Pattern Theory, Vol. 2 - Pattern Analysis"); references are to "Lectures on Pattern Theory, Vol. 1 - Pattern Synthesis", Springer-Verlag 1976)

December 1976

Supported by Contract N00014-75-C-0461 between Office of Naval Research, Information Systems Program, and Brown University.

PATTERN PROCESSORS FOR LANGUAGE ABDUCTION

7.1. Abduction of regular structures.

In Chapter 6 we studied a network $\mathcal{N}$ that could modify itself in order to learn at least some of the patterns appearing in its environment. This was done by modifying the coupling coefficients of $\mathcal{N}$, i.e. by changing the image processor that $\mathcal{N}$ represents. The resulting pattern inference is of inductive type: with the aid of observations of what happens in the environment the network $\mathcal{N}$ will incorporate more and more of the surrounding pattern structure.

We shall now turn to pattern processors that also carry out inference about the structure, but in a way emphasizing the explicit generation of plausible hypotheses. Of course the previous network processor can also be said to generate hypotheses. Indeed, the changes in the geometry (see section 6.6) can be interpreted as giving greater credibility to some statements, or hypotheses, about the image algebra env(Ω), while other statements are made to appear less likely. This sort of hypothesis is implicit, however, in contrast to what will be studied here.

Using a term coined by C.S. Peirce (see Peirce (1955), p.150-156), we shall speak of abduction when the processor is applied to images from an incompletely known pattern structure in order to generate as output plausible hypotheses concerning the structure. Perhaps one could also call it "plausible reasoning", adopting Polya's terminology, but if so, only for the limited context that we shall describe in the next section.

Denote the incoming images by $I_1, I_2, I_3, \ldots$, all from some image algebra $\mathscr{T}$, and where the images will be assumed to be pure for the moment; the modifications needed when they are deformed are far from trivial but the discussion of this will be postponed till later.

It should be pointed out that the output of the abduction algorithm (or abduction machine) should be pattern structures, not just individual images for each input image. In other words we are not looking for a single image operator that realizes a certain task well, such as image restoration. Instead we operate one level of abstraction higher and the mapping is of the form

$$(1.1) \qquad \text{ABDUCTION: } \mathscr{T} \times \mathscr{T} \times \mathscr{T} \times \ldots \to \Pi$$

where Π stands for a collection $\{\mathscr{P}\}$ of pattern structures, let us say image algebras. Of course Π should not be completely general, the choice of $\mathscr{P}$'s is limited by what we know a priori about generators, bond relations, connection types, etc. The other extreme, that Π consists just of two or a few possible $\mathscr{P}$'s is also of limited interest so that Π will be assumed to contain a large or even infinite set of pattern structures.

Concerning the number of factors in the Cartesian product of the left hand side of (1.1) we shall assume that it is finite, but also that the abduction algorithm should be sequential, in the sense that when another input image arrives we should only have to do a moderate amount of additional computing to get a new hypothesis over what we already computed.

More importantly, we shall look for pattern processors that are robust, insensitive to errors, and natural. Robustness means that even though the algorithm has been designed to work well for a certain type of pattern structure, it will not break down completely when exposed to a slightly different type. Its performance may deteriorate but it should not cease functioning. Insensitivity to errors means that occasional errors in the computing or in the inputs should not have any serious lasting effects but their influence should die out as more images are processed. The concept "natural" is more difficult to make precise and what seems a natural algorithm to one researcher may appear as artificial to another. Anyway, we shall try to choose the algorithms in such a way that it is conceivable that they may be implemented in real world pattern processors. More about this later.

The reason why we emphasize these three properties as well as the inductive, rather than the deductive, type of logic is that we are looking for models with potential applicability to processors occurring in the real world even though they may be unrealistic in their details at present, see Vol. 1, p.266. On the other hand we shall pay less attention to other desirable properties, such as computational efficiency, speed of convergence, etc.

How would one go about the task of inferring the image algebra $\mathcal{T} \in \Pi$ having been given the sequence of images $I_1, I_2, I_3, \ldots$? If Π is denumerable, so that we can write

$$(1.2) \qquad \Pi = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \ldots\}$$

we could start testing as a hypothetical pattern structure the image algebra $\mathcal{T}_1$ for I_1, and if $I_1 \in \mathcal{T}_1$ continue by $I_2, I_3, \ldots$ As soon as we reach an image I_k for which $I_k \notin \mathcal{T}_1$ we go the next image algebra $\mathcal{T}_2$ and start again testing for $I_1 \in \mathcal{T}_2$, then $I_2 \in \mathcal{T}_2$ and so on. We stop when we reach an image algebra $\mathcal{T}^* \in \Pi$ for which no test fails.

To be able to guarantee that this leads to the correct hypothesis $\mathcal{T}$ in a finite number of steps we need in general access to an unlimited (potentially infinite) sequence of images as well as some guarantee that the sequence is"representative" for the whole image algebra $\mathcal{T}$. In the following we shall let the sequence be generated by a simple random source according to some probability measure P over $\mathcal{T}$. We must then of course require, in general, that the support of P is the entire $\mathcal{T}$.

We could then try to prove theorems of convergence, saying that the algorithm will converge to $\mathcal{T}$ in a finite number of steps with probability one. Such results can be obtained, assuming for example that each $\mathcal{T}_\ell$ is denumerable, but we shall not pursue this line of thought.

Indeed, such an algorithm can scarcely be said to be natural: it amounts to no more than trial and error. Furthermore it does not represent abduction as described above, since the successive hypothesis have not been generated to be plausible with respect to the observed sequence, they are just given by a fixed sequence.

In order to select plausible hypotheses let us rephrase the problem in accordance with current statistical doctrine. Given a finite section $I_1, I_2, \ldots I_n$ of the sequence of images, think of the true image algebra $\mathcal{T}$ as an unknown "parameter" to be estimated. We can then ask for the "best" estimator according to a specified criterion and apply statistical estimation techniques. Our "parameter" $\mathscr{T}$ can of course be something rather different from and more complicated than what is the case in standard statistical estimation theory, but we may still be able to solve such an estimation problem successfully.

The resulting estimator $\mathscr{T}^*$ can then be said to be plausible since it utilizes the given information as well as possible to select the hypothesis. The naturalness of such algorithms may be less convincing, though. Such estimation techniques may be based on the method of maximum likelihood, on Bayesian ideas, on the principle of least squares etc., and in order to carry out maximization they may employ numerical schemes like Newton's method, or carry out matrix inversion, or use search strategies to find a maximum. The idea that human intelligence, in particular language learning, would be based on, say, the ability of the human mind to do, for example, matrix inversions is less than appealing. We have to look elsewhere for more natural abduction algorithms.

When the images in the sequence are encountered one could attempt inference of $\mathscr{T}$ if the bond structure and bond relations could be observed, since this would give information, at least partial, about the connection type Σ and bond relation ρ . Then the combinatory rules $\mathscr{R} = \langle\Sigma,\rho\rangle$ could be inferred to some degree.

Unfortunately this is seldom possible since the internal bonds are usually not directly observable; Chapter 3 of Volume 1 contains many examples of this. To avoid this difficulty let us assume that the learner has access to a teacher in addition to the pure image sequence. The teacher's role is to help the learner by telling him whether other images belong to $\mathscr{T}$ or not. Then we could base the abduction on intentional deformations of the I_k by applying one or several deformations to each I_k and ask the teacher whether the deformed image is still within $\mathscr{T}$. The answers will contain information about the internal bonds although it will be less complete than would be the case if the whole configuration wereavailable for observation. Summing up, the problem has now been reformulated as follows.

Case 7.1:1 (inference by deformations). Given a sequence $I_1, I_2, I_3, \ldots$ of pure images from $\mathscr{T}$ and a deformation mechanism $\mathscr{D}$, use the knowledge of whether $I_k^{\mathscr{D}} = \mathscr{D} I_k \in \mathscr{T}$ or not to generate hypotheses about the pattern structure.

This is still too general to be of any real help since it does not say how to choose $\mathscr{D}$. We have seen in Volume 1, Chapter 4, tha there is a rich variety of deformation mechanisms and we now have to narrow down the choice.

Recalling that we are looking for information about $\mathscr{R} = \langle \Sigma, \rho \rangle$, it seems natural to use a $\mathscr{D}$ that leaves much of the image unchanged and only modifies a sub-image involving only one or a few bonds. In case $\mathscr{D} I_k$ is not in $\mathscr{T}$ we should concentrate our attention to the sub-image mentioned and its connection to the rest of I_k. More formally:

Case 7.1:2 (deforming sub-images). $\mathscr{D}$ is said to deform sub-images if, for any $I \in \mathscr{T}$, the image is broken up into

(1.3) $I = I_1 \sigma I_2$, I_1 and $I_2 \in \mathscr{T}$

and

(1.4) $I^{\mathscr{D}} = I_1^{\mathscr{D}} \sigma I_2$.

where $I_1^{\mathscr{D}}$ has the same external bond structure as I_1.

Note that this leaves the external bond structure as before but not always the external bond values. The reader may notice that we have encountered some related versions of Case 7.1:2 before. Indeed, a jittering deformation (see section 4.2 in Volume 1) restricted to $s_i = e$ for $i=m,m+1,\dots,n$ deforms only the sub-image containing the generators $g_1,g_2,\dots g_m$; $m < n$. The missing generator deformations (see Case 2.9, Volume 1) also belongs to this type: it annihilates a certain sub-image.

In order that we learn something from observing the regularity of $I_k^{\mathcal{D}}$ we must ask that the deformed images are occasionally outside $\mathcal{I}$, i.e. $\mathcal{D}$ should be heteromorphic. Otherwise, for automorphic deformations, we could not hope to find the true limitations upon the combinatory pattern structure. For example, a shift deformation (see equation (2.1) in section 4.2 in Volume 1) would not be powerful enough. On the other hand Case 2.4 in Volume 1 is a good example of a deformation that would seem a promising candidate for the abduction algorithm.

Another way of looking at the image relations in (1.3) and (1.4) is in terms of congruence (see Volume 1, p. 31). If I_1 is congruent with $I_1^{\mathcal{D}}$ then $I^{\mathcal{D}}$ will of course be regular, in $\mathcal{I}$. But if I_1 is not congruent with $I_1^{\mathcal{D}}$ then $I^{\mathcal{D}}$ may be irregular, not always, but for some I_2. Therefore the teacher's answer to the question whether $I^{\mathcal{D}} \in \mathcal{I}$ will tell us something about the congruence relation and hence about the pattern structure. Due to the "not always" we cannot claim with certainty to have arrived at a correct hypothesis, we are doing induction and not deduction.

Let us pursue this line of thought only a bit further here. Suppose that we tried jittering deformations of sub-images, we would run into the difficulty that the similarity group S may not be completely known to us a priori. To make sure that the deformations be drastic enough we should look for a set $\hat{S}$ of mappings $\mathcal{I} \to \mathcal{I}^{\mathcal{D}}$ containing S, $S \subseteq \hat{S}$, but probably larger since shifts may not be sufficient. We would then select elements of $\hat{S}$, apply them to a sub-image and observe the regularity or irregularity of the deformed image. We shall return to this in more detail in the next section.

After we have constructed an abduction algorithm the question arises how to implement it by physical devices. Expecially network processors like those in Chapter 6 present themselves as natural candidates for functioning as <u>abduction machines</u>. This will be attempted in section 7.3

7.2 Abduction of some language patterns.

Based on the general considerations of the last section we shall now attempt to construct abduction algorithms when the patterns come from some formal language. This will give a concrete illustration of how inference by deformations can be obtained using Case 7.1:2.

Before deciding what type of formal language we shall use, let us consider briefly the flow of information when we attempt abduction. The speaker Ω lives in an environment characterized by some image algebra env(Ω) in Figure 7.2.1. A given image $I \in$ env(Ω) can give rise to many different sentences belonging to a language L(Gr) described by a grammar Gr. This means that an image processor maps the image algebra env(Ω) into another image algebra, so that the image operator "transducer" in the figure takes microworld images into language images.

As an example of how such an image operator may work the reader is referred to section 2.4. Of course the mapping will be one-to-many, since a given image I in env(Ω) can give rise to many syntactically correct sentences, following Gr, and agreeing with I semantically.

The sentence from L(Gr) is then subjected to the deformation mechanism $\mathcal{D}$, the second image operator in the figure, that changes a sub-image. The result is presented to the teacher and is also stored in the short term memory of Ω together with the instruction from the teacher and the undeformed sentence. Say that the teacher only answers yes or no, according to the grammaticality of the sentence. The grammaticality function will be denoted gr(·) and is of course not known to Ω a priori. It takes the values YES and NO. The abduction algorithm processes these three inputs and saves the result in some form, yet to be specified, in the long term memory after which the short term memory is cleared. As more and more sentences are processed the algorithm is expected to converge to a limiting grammar weakly equivalent to Gr, and with performance parameters that characterize the probability distribution over L(Gr).

In this chapter we shall study abduction of the language L(Gr) when no semantic input is available, so that in Figure 7.2.1 env(Ω) is not connected to memory by the dotted lines. The situation with semantic input from the image algebra env(Ω) is a challenging research problem which will not, however, be studied here.

In the figure we have two image operators: the transducer with sentences from L(Gr) as output and $\mathcal{D}$, the deformation mechanism that was discussed briefly in the last section, whose outputs are strings over the same terminal vocabulary as used in its inputs. The specification of the abduction algorithm should define the latter one in detail and we now address ourselves to this question.

First we must decide what type of grammars to use here. A simple but not trivial type is the finite state grammar and this is what we shall use for Gr; see Notes for further discussion as well as for historical remarks.

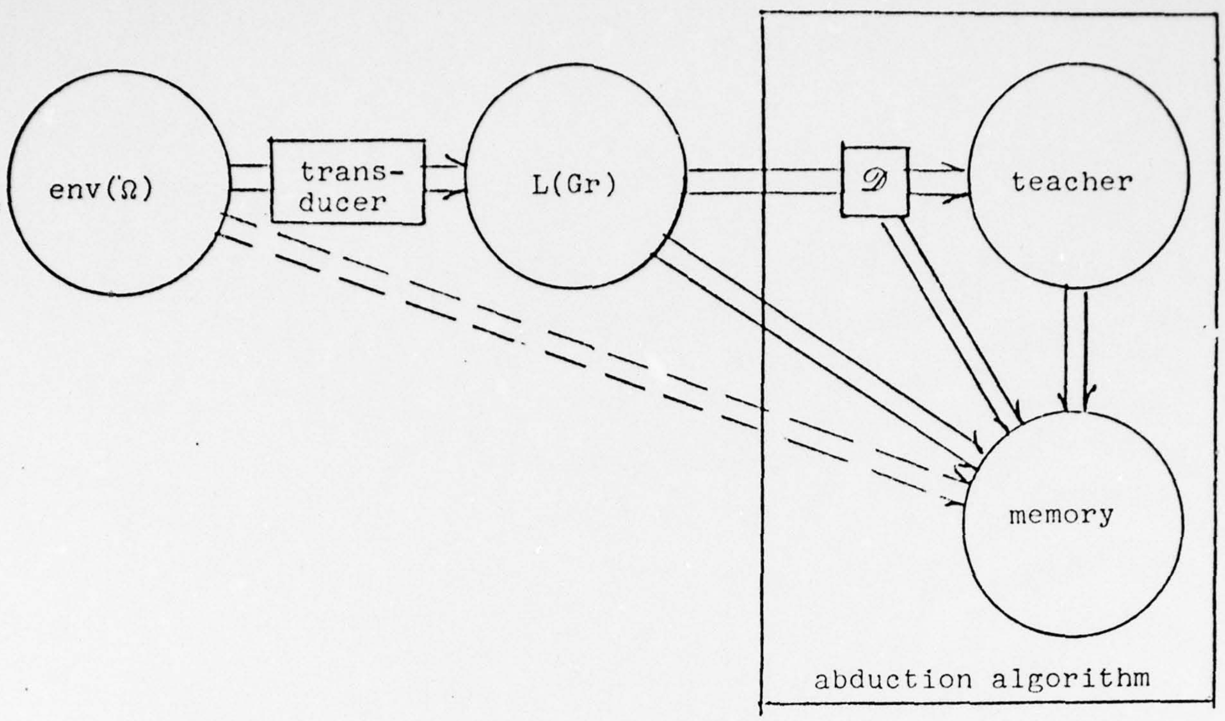


Figure 7.2.1

We use the same assumptions and notation as in Volume 1, sections 2.4, 2.10, and 3.2. The terminal vocabulary V_T contains n_T words denoted generically as x,y,... or these letters subscripted as needed. The syntactic variables, the non-terminals, form a set V_N with n_N elements, denoted by i,j,... or these letters subscripted as needed. The rewriting rules are of the form

$$(2.1) \quad \begin{cases} i \to xj; & x \in V_T; \quad i,j \in V_N \\ i \to x\ ; & x \in V_T; \quad i \in V_N \end{cases}$$

the latter type resulting in termination of the derivation. The number of rewriting rules is denoted n_r. The corresponding probabilities are denoted $p_{ij}(x)$ and $r_i(x)$ forming a matrix P(x) and a vector r(x) respectively. Also

$$(2.2)\quad \begin{cases} P = \sum\limits_{x\in V_T} P(x) \\ r = \sum\limits_{x\in V_T} r(x) \end{cases}$$

With the usual assumptions the consistency question of the syntax-controlled probability model is automatically answered in the affirmative, see Theorem 10.7 of Chapter 2, Volume 1, p.90. Hence there exists a well-defined probability measure over $\mathscr{C}(\mathscr{R})$. Recall that regular configurations here means linear strings over V_T and that bonds take values in V_N. Of course the internal bonds have to satisfy the bond relation ρ = EQUAL. The corresponding images are "phrases" with one in-bond and one out-bond.

A generator is a rewriting rule, as in (2.1), so that it can be regarded as an element from V_T, a word, together with its in- and out-bonds, (i,j) or (i,F) where F represents the final state. The initial state will be chosen as $i=1$.

It is tempting to think of a generator as just a word from V_T. This is not correct since it can very well happen that one and the same word x appears in two different rewriting rules $i \to xj$ and $i' \to xj'$. Therefore the mapping $V_T \to G$ can be one-to-many which will have important consequences later on.

Let x and y be fixed in V_T and pick two arbitrary strings $u,v \in V_T^*$. If it is always true that the concatenated strings uxv and uyv are grammatical or ungrammatical together we say that $x \equiv y$, they are equivalent (or congruent). In other words

$$(2.3)\quad x \equiv y \Leftrightarrow gr(uxv) = gr(uyv), \quad \forall u,v \in V_T^* .$$

This equivalence partitions V_T into equivalence classes. Note that equivalence in (2.3) demands that the right hand side hold for all u,v. Hence an infinite number of tests would be required since V_T^* is infinite. Testing it for a single case (u,v) or a finite number of cases does not suffice. Nevertheless one feels that if the relation holds for many (u,v)-combinations then x and y are likely to be equivalent, in some sense that has not yet been made precise.

We shall need the following simple statement.

Lemma 2.1. In order that $x \equiv y$ it is necessary and sufficient that for any generator of the form $i \to xj$ there exist one of the form $i \to yj$ and vice versa.

Proof: Consider two words x and y, and strings uxv and uyv. If for any generator $i \to xj$ there is one $i \to yj$ it is clear that $gr(uxv) = gr(uyv)$, and since this holds for any $u, v \in V_T^*$ it follows $x \equiv y$.

On the other hand, let us choose x and y such that $x \equiv y$. If uxy is grammatical and u takes the initial state into the i^{th} state, x takes i into j, and v takes j into F, then, in order that uyv also be grammatical it is necessary that the internal bonds fit. Hence there must be a rewriting rule of the form $i \to yj$ and the proof is complete.

To emphasize the equivalence property we choose as our similarity transformations the set of all those permutations of generators that leave the equivalence unchanged, so that if $g = i \to xj$ then $sg = i \to x'j$ with $x \equiv$ some x'. This determines the generator classes G^α invariant with respect to S.

Of course S is unknown ab initio and should be learnt during the abduction process. We will start with some set $\hat{S}$ of transformations of the images, where $\hat{S}$ need not be a group. As our first task, to be carried out in sections 7.3-7.4, we shall take the determination of the (unknown) word classes.

To make the following as concrete as possible we shall use a "test grammar". Of course we could have chosen one using a vocabulary consisting of abstract symbols since we are not concerned with natural language processing here. For didactic reasons, however, we have instead selected one generating English-like strings so that the output is easier to read. Since the semantic background has been left out it will be necessary to "fudge" the grammar to avoid completely meaningless sentences from being grammatical.

The grammar has $n_T = 52$ including the punctuation mark "." and a list of the terminal vocabulary is given in Table 2.1. These 52 "words" are arranged in 23 word classes denoted by, for example, DET for determines, NH for human norm and so on.

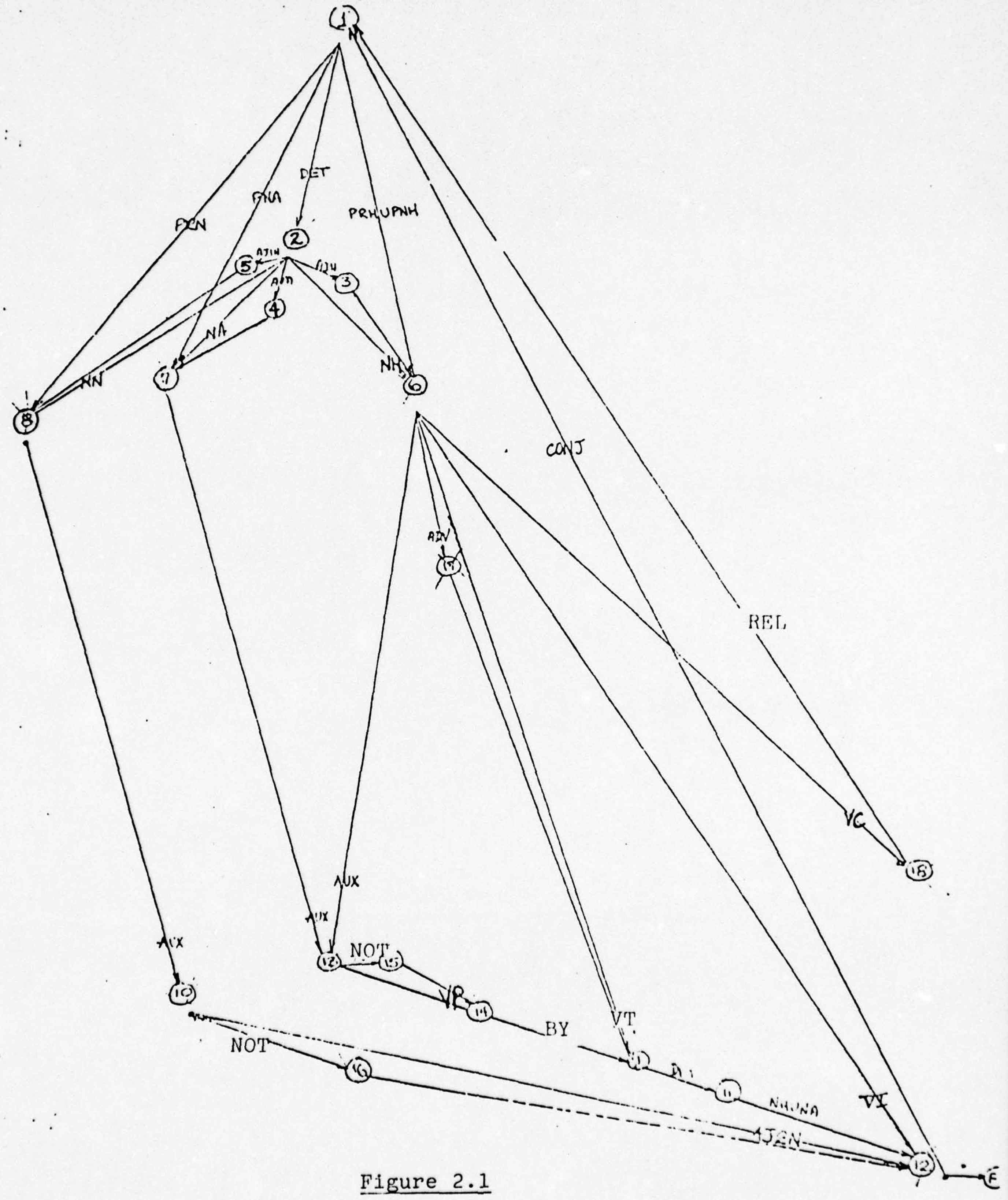
	Class Code		Words
1	1		*A
2	1	DET	THE
3	1		SOME
4	2		*TALL
5	2	AJH	CLEVER
6	2		SHORT
7	2		YOUNG
8	3		*SPOTTED
9	3	AJA	FRISKY
10	4		*FINE
11	4	AJ1N	NEW
12	4		VALUABLE
13	5		*BLUE
14	5	AJ2N	ORANGE
15	5		GREEN
16	6		*MAN
17	6	NH	BOY
18	6		WOMAN
19	6		GIRL
20	7		*CAT
21	7	NA	KITTEN
22	7		DOG
23	7		PUPPY
24	8		*TABLE
25	8	NN	CHAIR
26	8		DESK
27	9		*IS
28	9	AUX	WAS
29	10		*SEEN
30	10	VP	HURT
31	10		HELPED
32	11		*LIKES
33	11	VT	DISLIKES
34	12		*SPEAKS
35	12	VI	SINGS
36	13		*AND
37	13	CONJ	WHILE
38	14		*HE
39	14	PRH	SHE
40	15	PRN	*IT
41	16		*MARY
42	16	PNH	JOHN
43	17		*TOUKA
44	17	PNA	ROVER
45	18		*IMMENSELY
46	18	ADV	VIOLENTLY

47	19	VC	*SAYS
48	19	VC	CLAIMS
49	20	REL	*THAT
50	21	BY	*BY
51	22	NOT	*NOT
52	23	DOT	* .

The test grammar Gr has 19 states including the final one F=19 as in Figure 2.1. This corresponds to the generators listed in Table 2.2, where for example 1 → PNA,7 really represents two re-writing rules since the word class PNA contains two words. In all we have 87 rewriting rules.

A program generates sentences from L(Gr) as described in section 3.2 of Volume 1. The performance will of course depend upon the probabilities associated with the generators. Many of the sentences are quite reasonable, such as HE IS HELPED BY A BOY, or JOHN SPEAKS, or THE DESK IS NOT BLUE. Some are a bit doubtful, such as JOHN CLAIMS THAT JOHN SINGS, or SOME VALUABLE TABLE IS NOT GREEN. More seldom one gets a very strange sentence, for example, HE CLAIMS THAT THE MAN CLAIMS THAT A WOMAN IS HELPED BY THE DOG, or SHE VIOLENTLY LIKES THE BOY WHILE HE SPEAKS. It may also be mentioned that some perfectly reasonable looking English sentences over the given terminal vocabulary are not accepted by Gr, for example ROVER LIKES THE GIRL. For our purpose the grammar represents a sufficiently difficult task however.

Consider now the four generators of the form 11 → NH,12. The words in NH are certainly equivalent to each other. Similarly the four generators of type 11 → NA,12 use the words NA which are equivalent to each other. All these eight generators go from state 11 to 12 and one may be tempted to believe that the elements in NA are equivalent to the elements in NH. This is not the case however, since Lemma 2.1 tells us that in order that this hold we must have, for example, for the generators 2 → NH,6 generators of the form 2 → NA,6. The latter ones do not appear in Gr, so that



DET
PRN
PRHUPNH
AJN
AJN
AJN
NA
NN
NH
CONJ
REL
ADV
VC
AUX
AUX
AUX
NOT
VP
NOT
BY
VT
NH/NA
VI
AJEN
E

Figure 2.1

Table 2.2

generator number	generator
1	1 → PRN,8
2,3	1 → PNA,7
4,5,6	1 → DET,2
7,8	1 → PRH,6
9,10	1 → PNH,6
11,12,13,14	2 → AJH,3
15,16	2 → AJA,4
17,18,19	2 → AJ1N,5
20,21,22,23	2 → NH,6
24,25,26,27	2 → NA,7
28,29,30	2 → NN,8
31,32,33,34	3 → NH,6
35,36,37,38	4 → NA,7
39,40,41	5 → NN,8
42,43	6 → VT,9
44,45	6 → VI,12
46,47	6 → AUX,13
48,49	6 → ADV,17
50,51	6 → VC,18
52,53	7 → AUX,13
54,55	8 → AUX,10
56,57,58	9 → DET,11
59,60,61	10 → AJ2N,12
62	10 → NOT,16
63,64,65,66	11 → NH,12
67,68,69,70	11 → NA,12
71,72	12 → CONJ,1
73	12 → ·,F
74,75,76	13 → VP,14
77	13 → NOT,15
78	14 → BY,9
79,80,81	15 → VP,14
82,83,84	16 → AJ2N,12
85,86	17 → VT,9
87	18 → REL,1

the equivalence between *NA* and *NH* does not hold, which will introduce an essential difficulty which will be studied in the next section.

7.3. Word class partitioning. A crucial part of the abduction algorithm tries to find the partition of V_T into classes of equivalent words. If this can be achieved we have reduced the "combinatorial size" of the task considerably since we can then operate on the level of pre-terminals rather than terminals. In the test grammar the number is then reduced from 52 to 23. In natural language it is likely that the reduction would be even greater.

But this is not the only reason why we shall pay so much attention. We shall see later that the partitioning problem presents the main mathematical difficulty: once it has been solved the full abduction problem for finite state grammars can be obtained by a similar construction. Therefore we shall examine this sub-problem in considerable detail.

Let us look at what is a real obstacle preventing us from using one of the standard algorithms. Consider two words $x,y \in V_T$ and ask whether they are equivalent or not. For this purpose we assume that some test procedure has been arrived at that will be applied to a given sentence I generated according to the syntax-controlled probability model.

As describe in the previous section the sentence I will be deformed by $\mathscr{D}$ into $I^{\mathscr{D}}$ by replacing one or several occurrences of x in I by y. If x does not occur in I no change is made. We will have to describe exactly how this replacement is done when we analyze $\mathscr{D}$ mathematically, but for the moment it will be enough to point out that the algorithm could not possibly be completely correct so that we have to introduce the probabilities of an error.

$$(3.1)\quad \begin{cases} \varepsilon = \text{the probability that the test says yes although } x \not\equiv y \\ \delta = \text{the probability that the test says no although } x \equiv y. \end{cases}$$

It is clear that $\varepsilon > 0$ since it can happen with positive probability that $x \equiv y$ but that x can be substituted for y in some sentences without destroying their grammaticality, see the last section. The second probability, δ, will be zero though, since if $x \equiv y$ then any substitution $x \to y$ will leave the sentence grammatical. However, if we allow for an imperfect teacher, or, what is the same thing, that the learner lives in an imperfect linguistic environment then we should also allow δ to be positive.

To calculate ε we must specify the testing algorithm precisely and we present one instance of such a calculation. For variations on the testing algorithm the expression will not hold exactly although it may indicate the order of magnitude of ε. Say that we pick the first occurrence of x in I, if any, and replace it by y. To find ε_{xy} for x and y fixed, we can reason as follows.

The probability of generating a sentence with s in the first position and of length L is

$$(3.2)\qquad \Sigma p_{1i_1}(x)p_{i_1i_2}(x_2)\dots p_{i_{L-2}i_{L-1}}(x_{L-1})r_{i_{L-1}}(x_L)$$

summed over $x_2, x_3, \dots x_L \in V_T$ and over the i's and interpreted as $r_1(x)$ if $L=1$. Similarly, to get a string with the first occurrence (if any) of x in the second position is

(3.3) $\Sigma p_{1i_1}(x_1)p_{i_1i_2}(x)p_{i_2i_3}(x_3)\ldots p_{i_{L-2}i_{L-1}}(x_{L-1})r_{i_{L-1}}(x_L)$

summed over $x_1 \in V_T - x$, $x_2,\ldots x_L \in V_T$ and over the i's, and we can do this for any position up to L.

Define

$$(3.4)\quad \begin{cases} t_{ij}(x) = \begin{cases} 1 & \text{if there is a rewriting rule } i \overset{x}{\to} j \\ 0 & \text{else} \end{cases} \\ \tau_i(x) = \begin{cases} 1 & \text{if there is a rewriting rule } i \overset{x}{\to} F \\ & \text{else} \end{cases} \end{cases}$$

In order that the deformed image $I^{\mathcal{D}} \in \mathcal{T}$ we must have when the first occurrance of x is the kth position

$$(3.5)\quad 1 = t_{1i_1}(x_1)t_{i_1i_2}(x_2)\ldots t_{i_{k-1}j_k}(y)t_{j_kj_{k+1}}(x_{k+1})\ldots t_{j_{L-2}j_{L-1}}(x_{L-1})\tau_{j_{L-1}}(x_L)$$

for some j-sequence. The event that I contains an x and $I^{\mathcal{D}} \in \mathcal{T}$ will then have a probability p_{xy} that can be obtained from the expressions in (3.2)-(3.3) by including the (3.5) right handside as a jack in the terms and summing as indicated but else over L=1,2,3,... . If we do this we get

$$(3.6)\quad p_{xy} = r_1(x)\tau_1(y) + \sum_{L=2}^{\infty} \Sigma p_{1i_1}(x)t_{1j_1}(y)p_{i_1i_2}(x_2)t_{j_1j_2}(x_2)\ldots p_{i_{L-2}i_{L-1}}(x_{L-1})t_{j_{L-2}j_{L-1}}(x_{L-1})r_{i_{L-1}}(x_L)\tau_{j_{L-1}}(x_L)$$

$$+\sum_{L=3}^{\infty} \Sigma p_{1i_1}(x_1)p_{i_1i_2}(x)t_{i_1j_2}(y)\ldots p_{i_{L-2}i_{L-1}}(x_{L-1})t_{j_{L-2}j_{L-1}}(x_{L-1})$$
$$r_{i_{L-1}}(x_L)\tau_{j_{L-1}}(x_L)$$

$$+\sum_{L=4}^{\infty} \Sigma \ldots$$

where the second summation signs in each term indicate summation over i's and j's and over x's but only for $x_1 \neq x$ in the second sum, x_1 and $x_2 \neq x$ in the third sum, and so on. The expression can be written more conveniently introducing the arrays

$$(3.7)\quad \begin{cases} P_x = \sum_{\xi\neq x} P(\xi) = P - P(x) \\ M = \{\sum_{\xi} p_{\alpha\gamma}(\xi)t_{\beta\delta}(\xi)\} \\ N = \{\sum_{\xi} r_{\alpha}(\xi)\tau_{\beta}(\xi)\},\ n = \{r_{\alpha}(x)\tau_{\alpha}(y)\} \\ S(x,y) = \{p_{\alpha\gamma}(x)t_{\alpha\delta}(y)\},\ d = \mathrm{col}(1,0,\ldots 0) \end{cases}$$

We can then write (3.6) as

$$(3.8)\quad p_{xy} = d^Tn + \sum_{L=2}^{\infty} d^TS(x,y)M^{L-2}N + d^Tp_xn + \sum_{L=3}^{\infty} d^Tp_xS(x,y)M^{L-3}N$$
$$+d^Tp_x^2n + \sum_{L=4}^{\infty} d^Tp_x^2S(x,y)M^{L-4}N + \ldots$$

This gives us

$$(3.9)\quad p_{xy} = d^Tn + d^TS(x,y)(I-n)^{-1}N+d^Tp_xn+d^Tp_xS(x,y)(I-n)^{-1}N$$
$$+d^TP_x^2 + d^TP_x^2S(x,y)(I-n)^{-1}N + \ldots$$
$$= d^T(I+P_x+P_x^2+\ldots)n + d^T[I+P_xP_x^2+\ldots]S(x,y)(I-n)^{-1}N =$$
$$= d^T(I-P_x)^{-1}[n+S(x,y)(I-n)^{-1}N].$$

Note that S(x,y) and M are linear operators represented by 3- and 4-dimensional arrays respectively and the second identity operator I stands for the dientity as indexed by four subscripts $I = \{\delta_{\alpha\beta}\delta_{\gamma\delta}\}$.

On the other hand the prbability that a sentence from the syntax-controlled probability model will not contain any instance of the word x is

$$(3.10) \qquad d^T r_x + \sum_{L=2}^{\infty} \Sigma p_{1i_1}(x_1) p_{i_1 i_2}(x_2) \ldots p_{i_{L-2} i_{L-1}}(x_{L-1}) r_{i_{L-1}}(x_L)$$

summed over $x_1, x_2, \ldots x_L \neq x$, or using (3.7)

$$(3.11) \qquad d^T r_x + \sum_{L=2}^{\infty} d^T P_x^{L-1} r_x = d^T (I-P_x)^{-1} r_x$$

with $r_x = r - r(x)$.

The ε_{xy} is the conditional probability that $x \neq y$ is not detected if x occurs in the sentence and is replaced by y. Hence

$$(3.12) \qquad \varepsilon_{xy} = \frac{p_{xy}}{1-d^T(I-P_x)^{-1}x}$$

The appearance of the factor $(I-n)^{-1}$ in (3.9) would seem to make this expression difficult to compute directly. It is possible to give a direct and helpful interpretation of the matrix $Q = (I-n)^{-1}N$. Consider Figure 3.1 which represents the generation of the two strings nxv and nyv under consideration. Returning to (3.6) it is not difficult to see that the entry $q_{\gamma\delta}$ of Q means the probability of generating a string starting in state γ and ending in F such that the same terminal string also leads from state δ to F. Q need not be symmetric.

The q's must satisfy the recursion

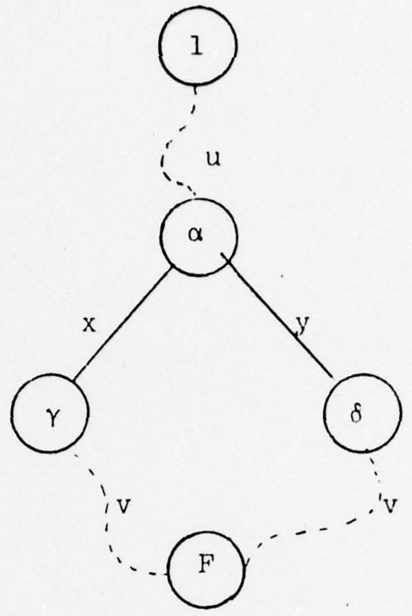


Figure 3.1

(3.13) $q_{ij} = N_{ij} + \sum_{\alpha,\beta} M_{i\alpha j\beta}\, q_{\alpha\beta}$

which can also be seen, of course, from the algebraic definition of Q. What is more important, however, is to realize that q_{ij} must be zero if there is no word ξ that exits from both i and j. In other words q_{ij} can be positive only if there are rewriting rules $i \overset{\xi}{\to} k$, $j \overset{\xi}{\to} \ell$.

This implies that Q will be an extremely sparse matrix where we can a priori fill in lots of zeroes just by looking at the diagram. Further, for a main diagonal element, q_{ii}, this value means just the probability of generating a string leading from state i to F. According to the assumptions that we have adopted this probability will automatically be equal to one.

Introduce a relation E between states and with iEj if ξ V_T and k, such that $i \overset{\xi}{\to} k$, $j \overset{\xi}{\to} \ell$.

Let E* be the transitive closure of E so that E* is an equivalence relation. Then E* partitions the set of states into classes and the Q-matrix be partitioned into block structure, with the rows and columns subscripts of the blocks corresponding to the classes induced by E*. The computational problem associated with (3.12) is therefore manageable.

Combining (3.11) with (3.9) we get the probability e_{xy} that the deformation $d(x \to y)$ does not detect that $x \not\equiv y$ as

$$(3.14) \qquad e_{xy} = d^T(I-P_x)^{-1}[r_x+n+S(x,y)Q].$$

Repeating this deformation on successive sentences we can therefore expect to have to repeat this a number of times, with the number of the order $1/1-e_{xy}$. For large values of e_{xy} considerable testing will be needed but we shall try to reduce this by improving the form of the partitioning algorithms.

It was observed empirically when the $\mathcal{D}$ described was simulated, that the selection of x from I could be done better. Mechanical criteria as selecting the first occurrence of x, where x itself is picked at random from V_T is wasteful. The same holds for selecting one at random of the (possible) occurrences of x in I.

Instead we should select x and y in such a way that we test critical choices, where we have reason to really suspect that $x \not\equiv y$, and not wast our effort by testing x and y often if we already believe that $x \equiv y$. This idea was implemented as follows.

Partitioning Algorithm: Step 1. Initialize by creating a single class consisting of all V_T and choose one word as the prototype of the class.

Step 2. The algorithm LISTEN produces a sentence from L(Gr) according to the (unknown) syntax controlled probability model.

Step 3. The algorithm ATTENTION (see below) selects an $x \in I$.

Step 4. The algorithm SPEAK produces the deformed string $I = d(x \rightarrow y)I$ with y equal to the prototype of the class x belongs to currently. The grammaticality of $I^{\mathcal{D}}$ is obtained. Go to Step 6 if $gr(I^{\mathcal{D}})$ = FALSE.

Step 5. If $gr(I^{\mathcal{D}})$ = TRUE the algorithm STRENGTHEN increases the plausibility of our belief that $x \equiv y$ by moving x closer to y in this class if possible. In other words the positions of x and the element next closer to y are permuted unless the next element happens to be y. Go to Step 2 (or stop).

Step 6. Move x to the end of the next class if there is one, else go to Step 7, and go to Step 4.

Step 7. Create a new class with x as its single element and ototype.

The lists representing the provisional classes can be visualized as in Figure 3.2 where typical movements of words have been indirected.

The algorithm ATTENTION is intended to avoid wasteful testing and concentrate the abduction process on hypotheses that seem uncertain at the moment. It considers the set of words in the current class and is convenient to think of them as a list ordered from left, with the prototype far left, to right. For each word measure its distance from the leftmost element, the prototype. ATTENTION selects one word at random but not with uniform distribution but with probabilities monotonically increasing with the distance.

The algorithm STRENGTHEN updates our current belief that $x \equiv y$ by moving x leftwards into a position associated with a higher plausibility of being equivalent to the prototype.

Theorem 3.1. The algorithm produces partitions that converge with probability in a finite number of steps one to the true partition.

Proof: Consider the sequence of sentences produced by LISTEN, $I^{(1)}, I^{(2)}, \ldots I^{(t)}, \ldots$ and denote by c_t the number of classes established after t sentences have been heard. Since c_t is non-decreasing in t and bounded by n_T it converges to some limiting random variable c_∞. If c_∞ is less than the true number of classes there exists at least one word, say x, not equivalent to any of the c_∞ prototypes. Let E be the probability given the c_∞ prototypes, that a sentence is produced that contains x, that x is selected and tested against each prototype and that these tests fail. Then P(E) is positive, although perhaps quite small. If E occurs then a new class will be established by Step 7 in the algorithm. Hence this will happen with probability one for the infinite sequence of sentences produced so that c_∞ must be equal to the truer number of classes. The way the prototypes have been selected they are all mutually non-equivalent. Therefore, in the

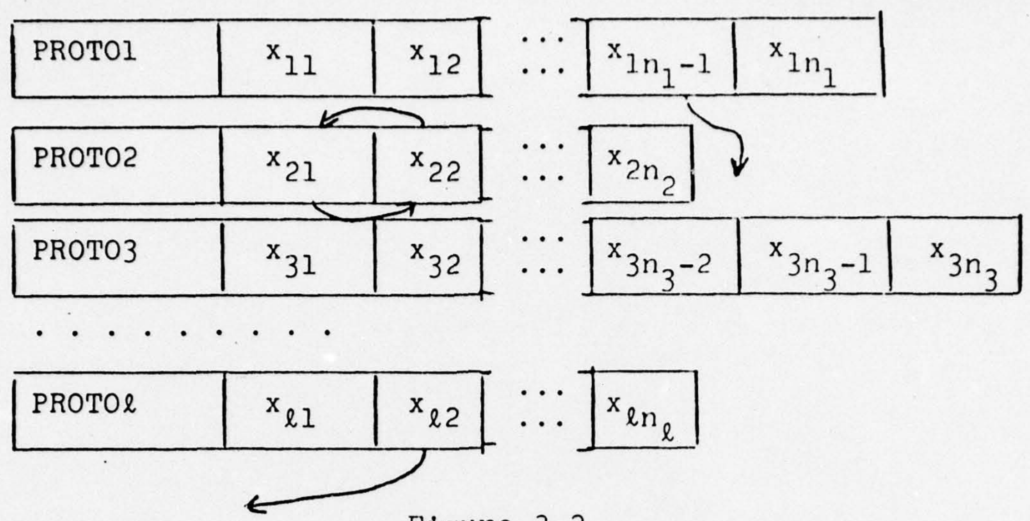


Figure 3.2

probability one, they represent each of the true equivalence classes.

The movements caused by the algorithm in Figure 3.2 is partly within classes and partly between classes. The first type of movement does not influence the partition (directly). The second type moves a word downwards to another class or to a new class. This implies that once n_∞ has been reached the algorithm only moves words away from classes to which they do not belong. A word x will not stay for more than a finite number of situations in the wrong class. Hence the partitions converge after a finite number of steps to the true partition.

The theorem guarantees that this pattern processor is consistent but it does not say anything about the speed of the convergence. To learn about this the algorithm has been implemented, actually in several quite different versions, and executed on the computer.

For the test grammar of section 7.2 words a classified rapidly into equivalence classes during the early part of execution.

The learning rate then slows down considerably. For a typical run after 150-200 sentences have been heard and processed most of the 23 equivalence classes have been established with a couple of words misclassified out of the 52.

Graphically this looks like Figure 3.3 showing the number of words classified correctly with number of sentences as abscissas, and Figure 3.4 which shows number of discovered word classes.

In some respect this abduction algorithm satisfies the requirements of section 7.1. Whether it is "natural" or not may be answered differently by different persons but at least it appears more natural than some alternative ones. It is fairly fast, although we do not claim any optimality. It is insensitive to the ε-error which plays a fundamental role in plausibility inference.

If we change the pattern structure, however, the algorithm appears less attractive. More precisely, if the answer to the learner as to the value of $gr(I^{\mathcal{D}})$ is not always correct so that the learner will be told that $I^{\mathcal{D}} \notin \mathcal{T}$ although $I^{\mathcal{D}} \in \mathcal{T}$ we have $\delta > 0$, see last section.

Senstinizing the algorithm it can then be seen that Step 7 may be taken when x is actually equivalent to the prototype of one of the provisional equivalence classes. Hence it can happen with positive probability that too many provisional equivalence classes are set up. Since the algorithm has no step involving coalescence of equivalence this mistake will never be corrected.

Therefore <u>the algorithm is not robust to small</u> (δ small) <u>changes in the pattern structure</u> of the linguistic environment of the learner. The question arises how to compensate for this.

One possibility is to include a coalescing step into the algorithm to be executed only occasionally. Although it may be possible to do this it will not be attempted since it would destroy the elegant simplicity of the algorithm. Instead a very different sort of algorithm will be examined.

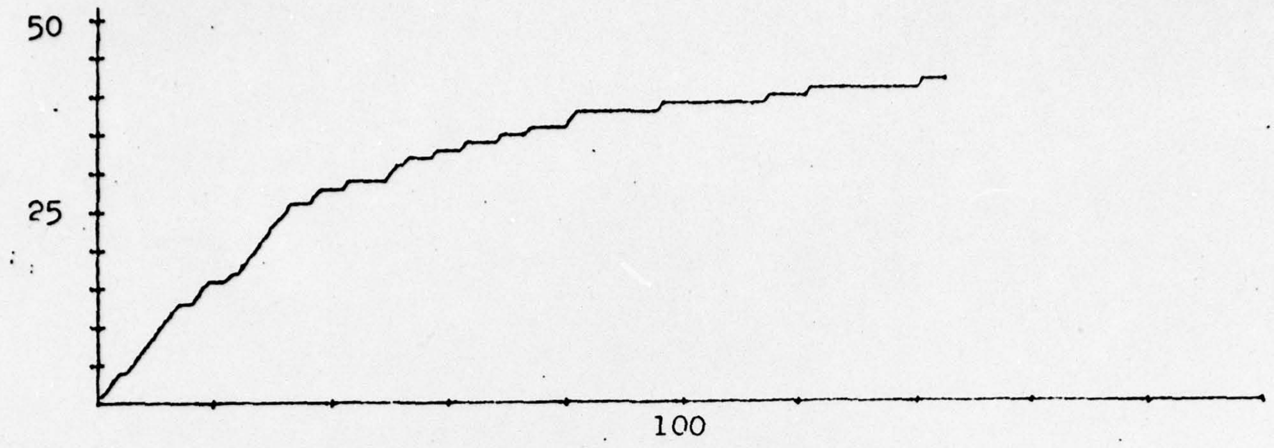


Figure 3.3

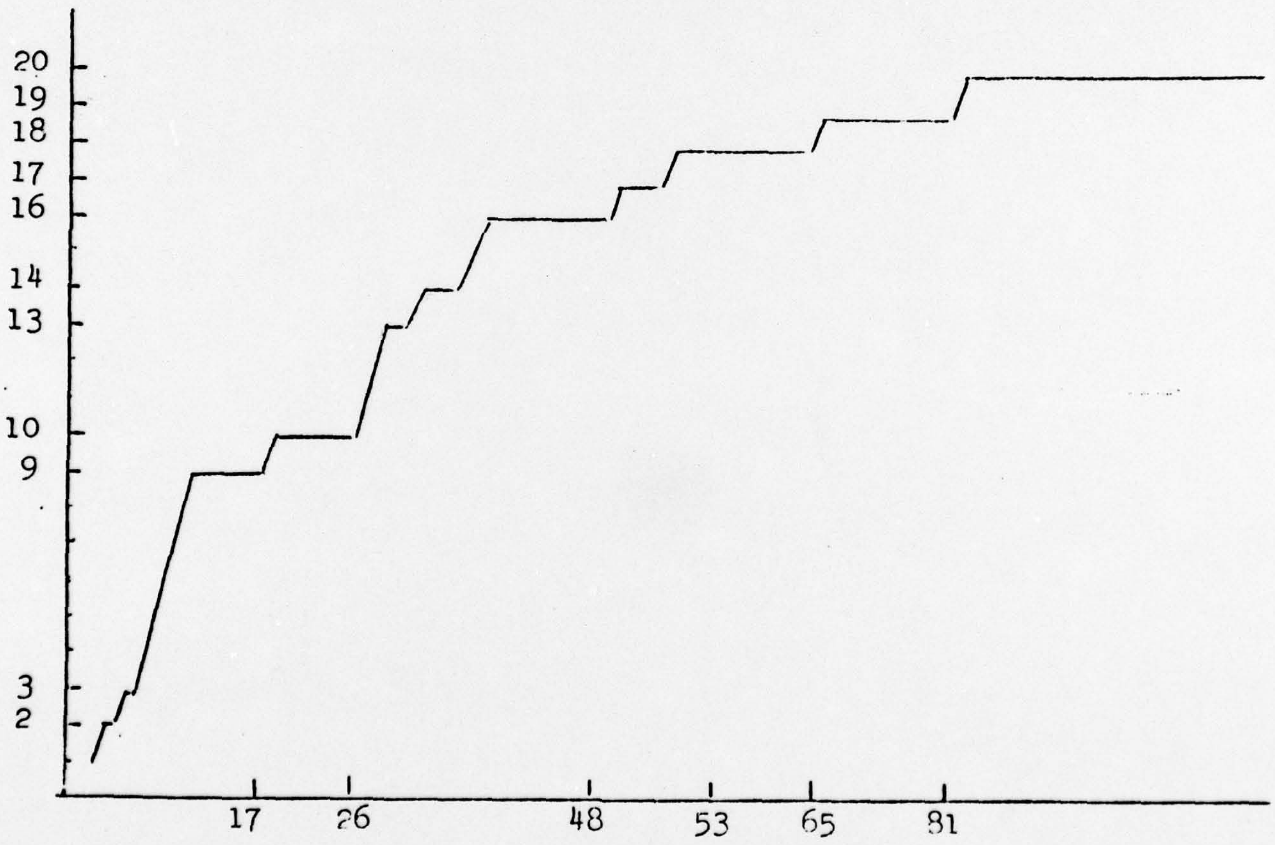


Figure 3.4

Notes

The approach in this section was strongly influenced by ideas due to the late A. Špaček on deduction-induction under imperfect conditions. An example of such ideas can be found in Špaček (1960). Unfortunately Špaček was not given the opportunity to complete his innovative thinking. His published work on this topic deserves to be better known.

Pattern inference in general, including pattern abduction, can be viewed as inductive behavior, just as J. Neyman (1950,1966) suggested that statistical inference can be described by this term. See also section1.1 in the current volume.

There is also some relation to work in artificial intelligence: mechanization of proofs, heuristic programs, etc. The reader could consult e.g. Hunt (1975), see Chapters IX to XII in particular, and the bibliography.

A particularly interesting attempt to formalize the induction process is due to R. Solomonoff, see Solomonoff (1964a,b).

Notes

7.2. Grammatical inference in general is what grammarians have been doing for thousands of years. Formal grammatical inference is of more recent origin but even so the literature is already voluminous. Most of it is only marginally related to the abduction study in this section. The interested reader may be referred to Hunt (1975), Chapter VII, Fu ( ), Chapter where many more references can be found. See also Patal (1972), Maryanski (1974).

More relevant to this section is the early discussion in Miller-Chomsky (1957) which suggests substitution as a natural principle on which to base the algorithm. Another important reference is Solomonoff ( ) which is based on the theorem. It is not known to the author if these early attempts were followed up by detailed analysis of robustness error sensitivity etc.

The material in sections 7.2- is based on a study begun in 197 as a result of a discussion between the author and L. Cooper, W. Freiberger, and H. Kucera. Some preliminary results were reported in Grenander ( ) but the algorithms were not analyzed in sufficient detail. A more complete analysis was presented in Shrier (1977) together with mathematical experiments illustrating the strengths and weaknesses of one particular abduction scheme.

The choice of finite state languages is very restrictive. It should be remarked here that our goal is not to study abduction of natural language, but to see how abduction can be organized in a concrete setting and what are the mathematical difficulties that one will then encounter, e.g. the determination of ε and δ , robustness, convergence.

When choosing similarity transformations one has to decide what are the relevant properties that we want to concentrate on, what sort of "sameness" are we interested in at the moment. Often we will have to operate with more than one S. In the present section we want to examine the validity of the bonds which leads to the definition used.