Translation of English into Logical Expressions

V. R. Pratt

A thesis submitted for the degree of

Master of Science

The Basser Computing Department

within the University of Sydney

August 1969

Table of Contents

Abstract

A computer program to solve Lewis Carroll's syllogisms is considered. A logical decision method is evolved for dealing with syllogisms expressed as conjunctive normal form (CNF) propositions. For the translation of English into CNF, a theory of translation is presented. A computer program is exhibited which explicitly embodies each feature of the theory, and produces CNF translations of Carroll's syllogisms. It is claimed that the translation theory is the most significant result of the research. A translation approach to phrase-structure grammars enables their practical value to be studied more closely. It is shown that the position of phrase-structure grammars is stronger than that of transformational grammars in a utilitarian theory, as distinct from an explanatory theory.

Acknowledgements

- Dr. J.B. Hext, for supervising this work, for suggesting the idea of Carroll's syllogisms and for providing stimulating criticism.
- The Basser Computing Department, for making available a PDP-8 computer for the development of a program described in this thesis, and for providing financial assistance in the form of payment for casual work.
- My wife Margot, for invaluable help in discussing theories presented here, and for many hours of time spent typing this thesis.
- Dr. Max Clowes, formerly of C.S.I.R.O Division of Computing Research, Canberra, for arousing my interest in transformational approaches to English.
- Associates of Max, including Robin Stanton, Richard Zatorski, Don Langridge and Chris Barter, all of whose ideas have influenced my thinking.
- Dr. Daniel Bobrow, for encouraging me to write the program, and for helpful advice.

Introduction

Solving syllogisms is a practical goal, and practical means suggest themselves readily. At the outset it seemed obvious that a context-free (CF) grammar would be adequate to help determine the right places to segment English premises into logical terms; so a computer program that did exactly that was written. It worked just as predicted: not perfectly, but well. The most annoying feature of the grammar was the rapid increase in the number of rules when trying to cater for peculiarities of negative sentences.

To demonstrate that a large computer was not needed, a PDP-8 with 4096 12-bit words was chosen. A teletype was the only peripheral used. These computers currently cost as little as eight thousand dollars. The program could segment an English sentence and find a corresponding logical expression in 1 second, for a 10 word sentence. If it were ambiguous, each additional formula would take an extra 0.2 seconds to calculate. (This was not very interesting, as it took 10 seconds to output each one.) The program used a mixture of matrix and list-processing techniques.

This was all done despite criticism of CF grammars as a model of English, by those who put forward transformational grammars as a better model. The attack has been, from some quarters, most vigorous, and one cannot help feeling that, if the attack is justified, then either the program should not work, or else the programmer has unsuspectingly embodied the esence of a transformational grammar in the program.

To help analyze this question, one takes the program, decides which features cannot possibly be done without, and endeavours to find a theory of why it is just those features that make it work. Such a process usually yields two or more possible theories, so one thinks up a much more difficult problem (as a Gedankenexperiment), to weed out theories that are not likely to survive long.

This was done, and the final theory, while anything but perfect, was felt to be more satisfying (and useful) than transformational theory.

The real problem with CF grammars seems to be the low capacity of non-terminal symbols as communication channels. A recent phrase-structure development (indexed grammar) was invoked to deal with this problem (for the Gedankenexperiment, not the syllogism solver). The theory handled these just as well as CF grammars, despite their greater power.

The hardest single problem that could be thought of as being a stumbling-block for CF grammars seemed to be the "respectively" problem. There are two versions of this problem: how do you tell that "Jim and Jack like Mary respectively" is ungrammatical; and what does one do with "Jim and Jack like Mary and Jane respectively"? By itself, an indexed grammar can answer the first question. The second question is much harder. We maintain that the transformational school would attempt to produce its deep structure. For reasons that will become clear later, it is our contention that it is sufficient to produce "Jim likes Mary and Jack likes Jane" to demonstrate that the problem has been solved. The solution is given in section 3.10.

In solving this problem, we made it more difficult to criticize phrase-structure grammars on the grounds that they simply could not produce such sentences. (Such a criticism is called "weak descriptive adequacy" by Chomsky.)

Another ground for criticism is that phrase-structure grammars "assign too much structure". In translating "big bad dog" into "x = big y = bad z = dog: x.y.z.", no structure whatsoever was encountered during the translation. Thus a much more valid criticism would appear to be that they assign no structure at all. Since not structures but logical formulae were our objectives, it was not clear how either criticism was related to the use of a CF grammar to do the job. It turned out, once translation theory was formulated, that the criticism was entirely fallacious, and that one could translate into logical formulae, structural descriptions or even bad French, with the one theory. The status of a structural description is that of a sentence in a structural description language. The assignment of structure was entirely subject to the choice of a suitable structural description grammar. This is argued more rigorously in section 3.1.

The claim above that no structure whatsoever was encountered during the translation is quite accurate. When the program was being designed, there was no thought of descrediting any approach. The problem was simply, how does one make use of a general-purpose CF grammar, that may be expected to have a non-terminal vocabulary of about 100 symbols, in 4000 words of memory? There was no room for complete structural descriptions.

Chapter 1. Logical Theory

1.1 Syllogisms

A syllogism is a pair of sentences called the premises, from which a single sentence, the conclusion, is to be drawn: A normal-form syllogism is one for which each sentence is in one of four normal-form premises. These, and some of their short-hand forms, are summarized thus:

	Normal-form	Traditional abbreviation	Lower Predicate Calculus
F_1:	All X are Y	XaY	$(x)(X(x)\rightarrow Y(x))$
F_2:	No X are Y	XeY	$-(\exists x)(X(x).Y(x))$
F_3:	Some X are Y	XiY	$(\exists x)(X(x).Y(x))$
F_4:	Some X are not Y	XoY	$(\exists x)(X(x).-Y(x))$

(The reader with an eye for symmetry may prefer for F_4, $-(x)(X(x)\rightarrow Y(x))$, the negation of F_1. The other, while logically equivalent, is closer to the "spirit" of the English.)

X and Y traditionally stand for noun phrases (thus making the normal-form premises grammatical). (x) is an abbreviation for "for each object in the universe, to which, for the remainder of this proposition (logical assertion), we give the name x..."; this means that the following assertion is true of any object, and this object is identified within the assertion as x. More commonly, (x) is read as "for all x...". Similarly (∃x) is read as "there exists an x such that ..." x is called a quantified variable, and (x) and (∃x) are quantifiers. "-" means "it is not the case that...". → means "implies", the period means "and", and the symbol v (not used above) means "and/or" (called "or" from here on). X(x) means "x is X"; similarly for "Y(x)". Thus, for example, the Lower Predicate Calculus (LPC) form of F_2 is to be read as "it is not the case that there exists an x such that x is X and x is Y".

Some examples of syllogisms (with their conclusions) are

XaY	XaY	XiY	XaY
YaZ	YeZ	YaZ	ZeY
XaZ	XeZ	XiZ	XeZ

For the moment, we appeal to the reader's intuition to verify that these conclusions agree with experiment. The traditional set of rules, called syllogistic inference, for drawing non-trivial conclusions, will shortly be seen not to concern us.

An obvious extension to a syllogism is the addition of extra premises. Strictly, such an extended syllogism is called a sorites, but here we shall relax our usage of "syllogism" to embrace sorites. Examples are

XaY	XaY	XiY
YaZ	YeZ	WaZ
ZaW	WaZ	YeZ
XaW	XeW	VaW
		XoW

The usual method of solving sorites is to take an appropriate pair of premises to form a syllogism, and then combine the conclusion with another premise, proceeding until the premises are exhausted. It will be noted from the third example that sorites need not be arranged in an order that facilitates this pairing.

A universal premise is one which refers to every instance of its subject. Thus F_1 and F_2 alone are universal.

1.2 Lewis Carroll's Syllogisms

These form a set of 60 sorites, ranging in size from three to ten premises. A total of 226 sentences are involved. They are of interest not so much from the logic-solving viewpoint as from the linguistic, as their form departs radically from the simple normal-form above. Extreme cases include "No discussions in our debating-club are likely to rouse the British Lion, so long as they are checked when they become too noisy.", and "I never have any really ridiculous idea, that I do not at once refer to my solicitor".

All of Carroll's premises are universal. This has the advantage of simplifying the logic aspects, allowing more attention to be paid to the language problems, in particular to that of finding an equivalent restatement of each premise that permits the application of simple rules.

1.3 Evolution of a Decision Method for Syllogisms

Although it is possible in the case of Carroll's premises to reduce each to a form "all X are Y" or "No X are Y", it can require considerable ingenuity. In addition, a "universe" is often specified. The third of Carroll's syllogisms, for example, mentions "potatoes of mine" in two premises, varying it in another premise to "my potatoes". In Carroll's formulation of the problem, "my potatoes" is given explicitly, at the end of the syllogism, as the universe for the syllogism; thus it may be neglected during the inference process, to be restored in the conclusion "none of my potatoes in this dish are new".

It was decided that the "helps" given by Carroll at the end of each of his syllogisms, which specified which segments (terms) of each premise were relevant, and which ones were universes, would not be given to the computer. As the segmentation problem is the hardest, it is by the same token the most interesting. That the computer must therefore determine the "universe", if any, is even more interesting.

The Lower Predicate Calculus forms were given above, as these are the forms most often used by modern logicians when considering decision methods. This is partly due to the versatility of LPC (many tortuous English propositions are readily translated into LPC) and partly to the established decision methods in the Propositional Calculus (PC), which help in evaluating LPC expressions. In the decision method to be described, it will appear that little reference to LPC is made, and that we could have assumed the PC in the first place.

However, translation directly into PC often appears unconvincing, and in these circumstances, a translation that considers the equivalent LPC proposition lends plausibility. Plausibility is the only criterion for translation into logic; the "correct" translation can often be open to interpretation and argument, as we shall see in computer-generated translations.

With this in mind, we quote without proof that

$$-(\exists x)(F) = (x)(-F)$$

where F is any formula. Thus F_2 becomes

$$(x)(-(X(x).Y(x))), \text{ or}$$

$$(x)(X(x) \rightarrow -Y(x)).$$

Abstracting the distinct features of F_1 and F_2, we are left with a briefer notation:

F_1 : $X \rightarrow Y$ F_2 : $X \rightarrow -Y$.

This notation, though it closely resembles that of PC, is derived from an LPC form, and we will argue in the LPC notation when there is a danger of confusion, namely, when two quantifying variables are involved, e.g., in two separate sentences.

It often happens that the subject or the predicate of a premise is itself a combination of logical terms, either their conjunction ("and") or their disjunction ("or"). This may arise because of the inclusion of the universe term, or because the combination need not be separated for the solution of the syllogism, or because a term in the subject is repeated (redundantly) in the predicate. Although it does not happen in Carroll's syllogisms, we may also have examples such as, "All dogs are furry mammals; all mammals are animals", where we want to deduce that all dogs are animals, ignoring the furry question.

It is possible, but messy, to use some theorems and/or axioms in PC. In this case, we call the following propositions axioms.

X1. $((A \to B).(B \to C)) \to (A \to C)$ (transitivity)

X2. $A.B \to A$ (abstraction)

X3. $(A \to B) = (A \to (A.B))$ (multiplication by the left)

X4. $A.B = B.A$ and $AvB = BvA$ (commutativity)

for a general purpose conclusion drawer. For example, rephrase the problem immediately above as

(D→F.M) and (M→A).

In LPC, this means "for all x, if x is a dog, then x is furry and x is a mammal , and for all y, if y is a mammal then y is an animal".

Now by X2, X4: F.M→M

By X1, (D→F.M).(F.M→M)→(D→M)

By X2, ((D→M).(M→A))→(D→A).

Thus, by careful choice of axioms we reach the required conclusion.

X3 would be used to cope with universes, e.g., the dogs in

"All red dogs are big; all big dogs are fierce":

(R.D→B) (P1) and (B.D→F) (P2)

Now R.D→R.D.B (X3 and P1)

R.D.B→B.D (X2)

B.D→F (P2)

R.D→F (X1 twice, on last 3 results)

that is, "red dogs are fierce".

A nice feature of this method is that it deals automatically with the universe term; that is, we did not have to discard it before we started the inference process.

These two examples demonstrate that syllogism-solving competence of a high order <u>can</u> be achieved using only ~~three~~ four axioms. However, a simple methodological approach is not immediately apparent. Further, to deal with X→-Y we must add

X5:   (X→-Y) = (Y→-X).

Otherwise, we could not draw the right conclusion from "All elephants are animals; no plants are animals".

An unpromising (at first sight) representation of PC expressions is that called Conjunctive Normal Form (CNF). A formula is the conjunction (the logical "and") of a set of disjunctions (the logical "or") of a set of (possibly negated) variables, e.g., (-Av-BvCvF).(-BvE).(Dv-E).
To express A→B in this form we write (-AvB).
It would appear that we have lost the transitivity theorem, X1, by ignoring the possibility of the → symbol in the new form. On the other hand, we no longer need to know that (X→-Y) = (Y→-X), as both

become (-Xv-Y) in the new form.

Let us rewrite X1, partly in CNF, but leaving untouched the main implication symbol. We have:

X1′:   (-AvB).(-BvC)→(-AvC).

Thus, if in two disjuncts ((-AvB) is an instance of a disjunct) we can find contradictory terms, then by cancelling them, and concatenating the remainder we have a disjunct for a valid conclusion.

This technique, which we shall call CNF inference, is a powerful method of dealing with Carroll's syllogisms. It extends beyond syllogisms, in that it can deal with, e.g.,

All black dogs are happy;
All of my pets are dogs;
All my pets are black.

Using the obvious abbreviations, we write (-Bv-DvH).(-Mv-PvD).(-Mv-PvB), noting that -(A.B) = (-Av-B) in rewriting (A.B→C). Cancelling contradictory Dogs,

(-BvHv-Mv-P).(-Mv-PvB),

and also for Black,

(Hv-Mv-Pv-Mv-P).

Noting that AvA = A, we have

(Hv-Mv-P), which is ((M.P)→H), i.e.,

"all my pets are happy."

That the method works with "All elephants are animals; no plants are animals." is seen from

(-EvA).(-Pv-A)

i.e., (-Ev-P) (by cancelling opposite Animals),

i.e., "no elephants are plants."

1.4 Rigorous Justification

We demonstrated the ease with which the method solves problems; its validity can to an extent be determined from X1′ above. However, as the technique is fundamental to the success of the syllogism solver, a more formal proof is in order.

Theorem 1: For any expressions E, F, and G, and a variable A, (E.(Fv-A).(AvG)) → (FvG).

Proof: (E.(Fv-A).(AvG)) = (E.(-F→-A).(-A→G))

since -AvB may be rewritten A→B.

Using X1, ((-F→-A).(-A→G)) → (-F→G).

Using X2, E.((-F→-A).(-A→G)) → ((-Fv-A).(-A→G)).

From these 3 results, and noting that $(-F \to G) = (F \vee G)$, we have the result, applying X1 twice.

Lemma 1: $(E.X \to B) = (E.X \to E.B)$

Proof: $(E.X \to B) = E.X \to E.X.B$ (X3)

$= X.E \to X.E.B$ (commutativity)

$= X.E \to E.B$ (X3)

$= E.X \to E.B$ (commutativity).

Theorem 2: $E.(F \vee -A).(A \vee G) \to E.(F \vee G)$

Proof: by writing $(F \vee -A).(A \vee G)$ for X and $(F \vee G)$ for B, in lemma 1,

$(E.(F \vee -A).(A \vee G) \to (F \vee G)) = (E.(F \vee -A).(A \vee G)) \to E.(F \vee G)$,

that is, theorems 1 and 2 are either both true or both false; theorem 1 is already proved.

Theorem 3: $E.(F \vee -A).H.(A \vee G).J \to E.(F \vee G).H.J$

Proof: Trivially, by theorem 2 and commutativity under "and".

Theorem 4: $E.(F \vee -A \vee K).H.(L \vee A \vee G).J \to E.(F \vee K \vee L \vee G).H.J$

Proof: Again trivially, by theorem 3 and commutativity under "or".

In theorems 3 and 4, H, J, K, L are any expressions.

Theorem 4 says that given two disjuncts embedded anywhere in the conjunction of a set of disjuncts, such that contradictory terms may be embedded anywhere in each disjunct, it is valid to draw a conclusion in the manner implied by the theorem. This in fact will be precisely the decision method we shall use for drawing conclusions. While sufficiently powerful to solve any sorites, it is sufficiently simple to warrant its choice for a problem-solving program.

Chapter 2. Syntactic Theory

2.1 Models of English

In considering the design of a logic system for solving syllogisms, we have presupposed that English premises can be decomposed into conceptual units, to which we may attach labels. The current approach, favoured by the followers of the "generative grammar" school of thought, is to postulate a mechanism for for the composition of sentences from conceptual units, and to perform decomposition by running this mechanism backwards. The extent to which this approach is practical can be judged partly by the extent to which the method fails to work, and partly by the efficiency of the method when it does work. In adopting this approach, we proceed on the assumption that the criteria that affect us fall into one or the other of these two categories.

Within schools of thought dominated by the generative ideology, there is a fairly clear-cut division into phrase-structure and transformational approaches. At the risk of misinterpreting the situation, we shall attempt a summary of the distinction between them.

2.2 Phrase-structure Systems

For the phrase-structure approach, it is suggested, but not espoused, by Chomsky (Chomsky, 1959) that sentences are the result of a one-dimensional symbol-string rewriting process. Starting with a given symbol, one erases it and replaces it by one or more other symbols, consistent with a set of constraints (or rewriting rules, or productions). The new symbols are then themselves subjected to the same process, which continues until no symbol may be rewritten under the constraints. The resulting string of symbols is then a sentence.

A phrase-structure grammar enumerates the symbols (vocabulary), usually partitioning them into rewritable (non-terminal) and terminal symbols (and occasionally more, e.g., in Aho, 1968). It also specifies the constraints, and nominates a starting symbol chosen from the non-terminal vocabulary. Actual examples of grammars only enumerate the constraints, as this is sufficient information to deduce the rest. S is traditionally the starting symbol, being suggestive of "sentence". We give such an example:

$$S \rightarrow NV$$
$$N \rightarrow \text{dogs}$$
$$V \rightarrow \text{eat}$$

Here the non-terminals are S, N and V, while the terminals are "dogs" and "eat". The word "symbol" is used loosely to denote any recognizable pattern that could plausibly be called an entity, with the exception of →, which serves mainly to delimit the symbol to be rewritten from the others, when specifying the rules.

The most general form of constraint has a string of non-terminals to the left of the →, and a string of symbols to the right, the latter possibly null, that is, having no symbols. Chomsky demonstrated that it was possible to constrain the constraints themselves, such that there was a set of sentences (or language, using Chomsky's definition) generated by a given grammar, that could not be generated by a more restricted grammar. In fact, he produced a hierarchy of four classes of grammars in this way. This hierarchy has since been considerably subdivided by other workers, and even extended to a lattice (that is, a system with a partial ordering, as distinct from a well-ordered hierarchy) (Ginsburg, 1967). The details of the hierarchy are beyond the scope of this discussion. However, the motive for considering the hierarchies is that while less restricted grammars generate a wider variety of sets of sentences, it is easier to analyze sentences generated by more restricted grammars. The equilibrium of this system seems to be stable, to judge from the amount of work done on grammars in the middle of the hierarchy, rather than on the extreme grammars. Arguments within the phrase-structure school can often be traced to the difficulty of estimating a pay-off function that can be used to find an optimum class of grammars for a given situation, though little attention has been paid this problem.

An objective justification for symbol-rewriting systems is that they model the process of articulation of objects in a one-dimensional universe. In the sample grammar above, the rule S → NV may be regarded as corresponding to an articulation possibility, that is, given an object having the property S, it may possibly be found to consist of an object having the property N, followed immediately by one having property V. Going in the opposite direction, we may say that, given an N, and a V following, we may regard the whole as an S. This particular justification is at its most powerful near the centre of the grammar hierarchy. Very powerful grammars do not model quite such a simple process. For example, the rule XYZ → ABCD would be interpreted as "Given an A, a B, a C and a D, the whole may be regarded as an X, a Y and a Z, in order". This is "less natural" than, say, interpreting XYZ → XABZ as "Given an A and a B, the whole may be regarded as a Y, provided it is preceded by an X and followed by a Z". The first example is permitted only in the most powerful (called type 0 by Chomsky) grammars, while the second is permitted in lesser grammars, called context-sensitive (the context in the example is X...Z).

The non-terminals in the symbol model correspond here to properties, while the terminals correspond to the actual primitive objects of the universe. The danger inherent in this objectification of the model is that properties are not always sufficient to identify the objects implied by the symbols. This is seen by some (e.g. Bach, 1966, p.38) as a fault of phrase-structure grammars, rather than of the objectification. For example, a pair of symbols may be reversed in a context-sensitive language, e.g., AB → CB CB → CA CA → BA. However, if the preceding objective view is taken, it would appear that, not the objects, but only their properties, have changed place. Bach says, "if we have PS (phrase-structure) rules which bring about a rearrangement of nouns and verbs, verbs will be analyzed as nouns and nouns as verbs". He is here criticising PS grammars. Presumably, this would evoke from the PS school the reply "the end justifies the means", that is, as the only observables we have are sentences, who cares how the grammar produces them, as long as they can be produced. The transformational school has a ready answer.

1.3 Transformational Systems

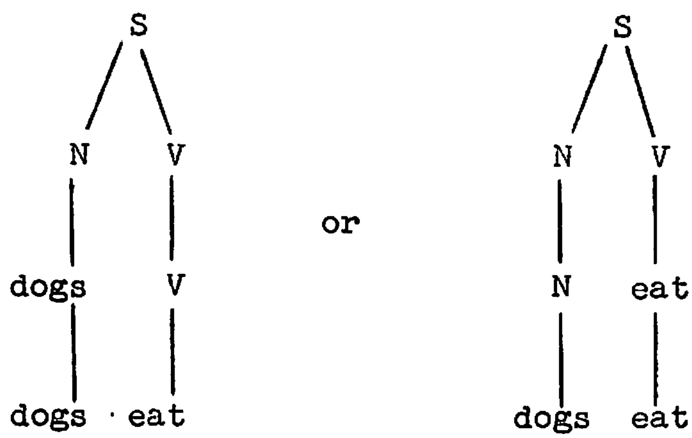
Chomsky contends that the function of a grammar is to "assign a structural description"(Chomsky, 1957, 1965, 1966, etc.). Moreover, a grammar must be able to rewrite not only symbols but parts of structural descriptions (or map them, but to claim (Clowes, 1969) that mapping is not rewriting is a verbal dispute).

As structural descriptions (SD's) are, we maintain, irrelevant to the PS school, they were omitted from the preceding discussion. There are various ways of looking at SD's (Chomsky, 1957, p.27; Clowes, 1969, p.3, etc.); Chomsky's will do for the moment, though we shall see later that Clowes' is nearer the mark.

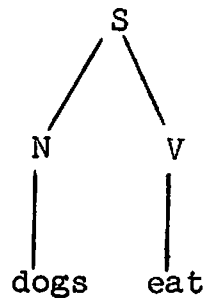
In the rewriting process (for PS grammars) an SD is a representation of this process. The most obvious way to start is by writing out the string generated so far at each step, e.g., in the earlier example:

S	or	S
NV		NV
dogs V		N eat
dogs eat		dogs eat

Chomsky calls this a derivation. The steps to form a structural description are given most explicitly in Postal (1964). Lines are drawn to indicate better the underlying mechanism of each step ("Elements are connected...to <u>identities</u>...which have replaced them" (italics mine)):



Then "all but the highest identical elements...are erased", thus:



in both cases.

This may seem long-winded, but Postal continues, "No other precise method of assigning such structural descriptions to infinite sets of sentences has, however, ever been described". (One of the less interesting results of the translation theory advanced in this thesis remedies this.) Postal uses this argument to justify the impossibility of having "correct" structural descriptions for a PS system that permits the rewriting of more than one symbol at a time.

To talk of elements being connected to identities, and to demonstrate that SD's are not feasible (or "correct") in the most powerful PS grammars, suggests that Postal is concerned with the identity of symbols, or of objects havng properties represented by those symbols. That is, Postal may be assuming the objectification we described earlier, without making it explicit, although there is room for debate.

However, once Postal, or for that matter, any exponent of transformational grammars, arrives at the section on transformations, there is no doubt that this is what is happening. In each rule, each symbol is tagged, using numbers, to ensure that its identity is not mistaken during the transformation process.

A transformation rule defines a structural change. It consists of a structural description part, which specifies conditions to be met by a structure before the rule can, and sometimes must, be applied, and a structural change part, which permits the permutation, addition or deletion of sub-structures. An example from Chomsky (1957, p.43), concerning the passive transformation, is:

$NP_1 - Aux - V - NP_2 \rightarrow NP_2 - Aux + be + en - V - by + NP_1$.

This means that, given some cross-section of an SD, a grammatical passive sentence can be formed by rearranging the structure as indicated. (Chomsky is at pains to point out: not "the passive sentence with the same meaning".) For example, if "John admires sincerity" is a grammatical sentence with a structure matching the left-hand description above, then "Sincerity is admired by John" is equally grammatical. The tagging of the NP's ensures that the sentence "John is admired by sincerity" is not also proved to be grammatical in this way. If this seems a peculiar reason for tagging objects, it must be remembered that Chomsky (Chomsky, 1957, p.93) sets himself the goal of constructing grammars without appeal to meaning. Thus a transformation may preserve grammaticality, but not meaning, for example (p.100), the passive of "everyone in the room knows at least two languages" does not mean the same as its active form.

More recent instances of transformation rules (e.g., Chomsky, 1964, p.227) make the distinction between properties and identities more clearly. For example:

1. Passive:

Structural Description: (NP, Aux, V_t, NP, $\{\frac{Adv}{-}\}$)

Structural Change: $X_1 - X_2 - X_3 - X_4 - X_5 \rightarrow$

$$X_4 - X_2 - be + en + X_3 - by + X_1 - X_5.$$

Why does a transformation operate on a whole structure, rather than on the result of a partial derivation? There are various reasons, but all of them are oriented to ensuring that the sentence possessing the transformed structure is no more or less grammatical than that possessing the untransformed structure. For example, most questions of the grammaticality of "golf plays John" are equally relevant to "John is played by golf". When the derivation of "golf plays John" reaches the stage "NP, V_t, NP", if this string of symbols were to be rewritten "NP, be, en, V_t, by, NP", and the derivation of "John is played by golf" carried out from there, some other means would then have to be introduced to deduce that this is ungrammatical, (assuming that "golf plays John" is ungrammatical). A mechanism that establishes the grammaticality of a sentence in the course of its derivation is more satisfactory, for Chomsky's ends, than one which requires additional external mechanisms to achieve the same effect. In addition, this scheme permits Chomsky to attack semi-grammaticality, a field not open to the PS school.

The precise mechanism for evaluating quirks of sentences like "golf plays John" is not germane to the syllogism translation process; if we say "golf plays all idiots; John is an idiot", then rather than object to the premises on the grounds that they are ungrammatical, we should conclude, equally ungrammatically, that "golf plays John". In fact, to a limited extent, the drawing of conclusions resembles a transformation in that it may preserve grammaticality. This observation, that we do not always want a total analysis of a sentence, will be seen to be important when we come to translating syllogisms.

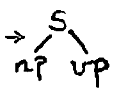
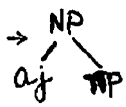
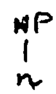
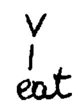
This account of transformational grammars is far too brief to do them justice; rather, we have attempted to determine what makes them superior to PS grammars. For more complete accounts, there are several good sources. For an efficiently economical account there is Clowes (1969). Extensive examples of transformational grammars may be found in Chomsky (1964, p.224) and Woods (1967, p.A19). A nearly complete treatment of the underlying mechanisms appears in Chomsky (1965, chap. 2, 3, 4) (it is felt that chapter 1 is considerably misleading in some places, and irrelevant in most others). The remainder of the literature is either concerned with ramifications of the material covered by the above references, or with most unprofessional attacks on other schools of thought, the most fallacious of these being in Bach (1966). There are several computer models of transformational grammars (Petrick, 1966; Zwicky, 1965; Thorne, 1967; Friedman, 1969; Rosenbaum, 1966), and to varying degrees they provide additional insight into the nature of transformational grammars. More importantly, though, they highlight practical shortcomings of the theory. Woods (Woods, 1967, p.4-4) observes, "The only existing algorithm for general transformational recognition (Petrick,1965) may take as much as an hour to recognize a single simple sentence with a very simple grammar". Since then, there has been improvement; Thorne's algorithm produces surface structures (the final structural description in the course of a transformational derivation) of sentences of 4 to 20 words, in the order of one second. Bobrow (Bobrow, 1969) accounts for the improvement in terms of better programming and a departure from "the detail of the processing required (commanded) by Chomsky". However, the algorithm currently in use by this author on a PDP-8 would, if implemented on a KDF-9 (the machine used by Thorne's programmers), produce deep or surface structural descriptions in the order of 10 milliseconds.

Chapter 3. Translation Systems

3.1 A Demonstration

Many new theories are better, or more unified, formulations of old theories, and can therefore best be introduced by demonstrating this relationship. Although the theory to be described falls into this class, the degree of incoherence of the old theories precludes any such demonstration sufficiently brief to be spectacular. Thus we shall first demonstrate a simple success of the theory.

We noted that Chomsky claims that a PS grammar can assign a structural description . We noted Postal's claims concerning the absence of precise methods of assigning structural descriptions, besides his own. We adapt formalizations given implicitly in recent literature (Chartres, 1969; Clowes, 1969) and exhibit the following instance of a translation system. Our goal is to discredit a criticism of phrase-structure grammars, that they "assign too much structural description", by showing that the assignment process can be realized effectively as a translation process.

S → NP VP	s → $[_{S}$ np vp]	s → 
NP → AJ NP	np → $[_{NP}$ aj np]	np → 
NP → N	np → $[_{NP}$ n]	np → 
VP → V	vp → $[_{VP}$ v]	vp → 
N → dogs	n → $[_{N}$ dogs]	n → 
V → eat	v → $[_{V}$ eat]	v → 
AJ → gentle	aj → $[_{AJ}$ gentle]	aj → 
AJ → neat	aj → $[_{AJ}$ neat]	aj → 

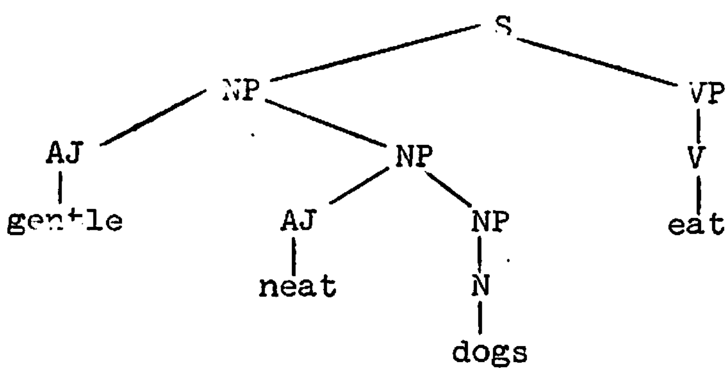
Formally, we define a translation system to be a set of grammars, and a correspondence between those grammars. We define a grammar to be a set of significant features of a language, and a correspondence between grammars to be a correspondence between their significant features.

In the above example, we have exhibited two phrase-structure grammars, and a crude picture grammar (Crude because such questions as exactly where the new symbols go when erasing the rewritten non-terminals are not immediately answered from the grammar; nor are those of orientation unless we assume that → preserves orientations. Chomsky dismisses similar questions in linear languages, such as the need for 1/10" of room for each terminal letter, and a line change at regular intervals, as questions of performance . We shall do likewise here.) In each grammar, the significant features are represented as rewriting rules for symbols. We invoke an earlier definition of non-terminal symbol, that is, one that can be rewritten using the rules. When writing grammars for structural description languages, one regrets the absence of more than upper and lower case letters on cheap typewriters, as can be seen from the difficulty involved in describing a language with both cases of terminal symbols. Thus the need for some other criterion for recognizing non-terminals than their case. At any rate, in the last two grammars, there is clearly no provision for rewriting lines, capital letters, English words or brackets.

Now consider a sentence generated by the first grammar, "gentle neat dogs eat". If we apply the same process that generated this to the other two grammars, we get:

$$[_S[_{NP}[_{AJ}\text{gentle}][_{NP}[_{AJ}\text{neat}][_{NP}[_N\text{dogs}]]]][_{VP}[_V\text{eat}]]]$$

using the second grammar, and:

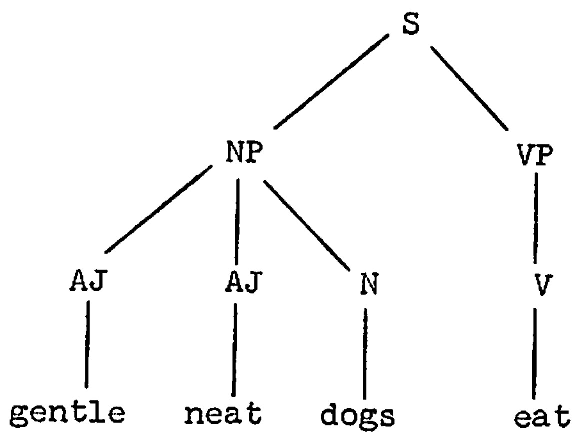


using the third grammar. Both of these will be recognized as structural descriptions. The one with brackets is described (Chomsky, 1966, p.37) as "the surface structure of a sentence" (italics mine). (More accurately, the surface structure is a bracketing). The diagram is often produced as being, in some sense, equivalent.

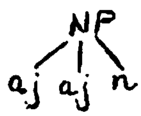
So far, we have done little that is new or exciting, save to counter Postal's claim above. However, there is a good reason for choosing noun phrases with more than one adjective, as these have been held up (Chomsky, 1965, p.196; Bach, 1966, p.68) as proof that phrase-structure grammars assign "too much structure" and therefore fail as models of English. The "proper" structure, according to these critics, is either:

$$[_S [_{NP} [_{AJ}\text{ gentle}] [_{AJ}\text{ neat}] [_N\text{ dogs}]] [_{VP} [_V\text{ eat}]]]$$

or:



If we were to allow the rules NP → AJ AJ N,

np → [$_{NP}$aj aj n] and np → 

in addition to the other rules (changing the second rule to NP → AJ N, *mutatis mutandis*, to avoid ambiguity) we then cannot account for a string of three adjectives. In fact an infinite number of rules are needed to generate the "proper" structural descriptions.

Thus, Chomsky and Bach implicate phrase-structure grammars, in particular, the first grammar in our system. This is ridiculous; the objects deserving criticism are the grammars of the structural description languages, if these have been made explicit. While they are not explicit, there can be no basis for this sort of witch-hunt. If there is some systematic way of producing structural description grammars, then this system deserves criticism, but not the original grammar itself.

Redirecting criticism to better places, we suggest the following structural description grammars, without indicating any preference for them over the others beyond the fact that their sentences are easier to read:

$s \rightarrow [_S [_{NP} np] [_{VP} vp]]$

$np \rightarrow aj\ np$

$np \rightarrow n$

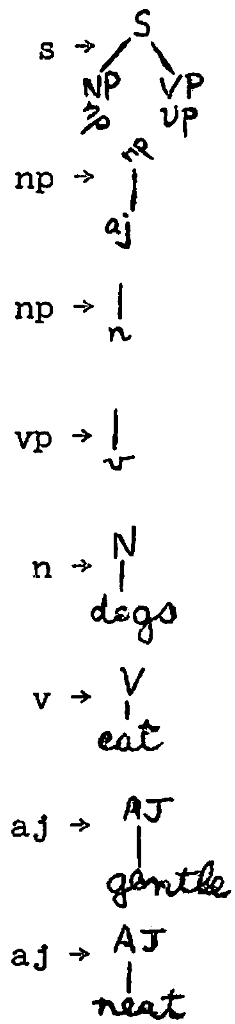
$vp \rightarrow v$

$n \rightarrow [_N dogs]$

$v \rightarrow [_V eat]$

$aj \rightarrow [_{AJ} gentle]$

$aj \rightarrow [_{AJ} neat]$



Applying the "same" process to these grammars as for the others, we produce the desired results. Quite clearly, the orientation question becomes important, in the first two rules of the picture grammar, as only a few adjectives will produce an unreadable picture. We have the choice of saying "performance", and leaving the decisions about length of line, and extent of rotation of "np" at each step to the user of the grammar, or we can say "competence", and therefore find fault with the notation because it neglects the fact that there are only 360° in a circle (just as PS grammars can be criticised for neglecting page width).

One solution to this problem is to consider who or what the grammar is intended for. If a human, humans are smart enough to extend the grammar appropriately. If a computer, some mechanism must be postulated, to enable it to function appropriately. To claim that this mechanism should have nothing to do with the grammar is to set up a spurious dichotomy (cf. Narasimhan, 1969, p.3) between the processing involving grammatical features and that involving so-called performance problems. Processing in a computer is homogeneous, at least to within these sorts of distinctions, and any reluctance to call a spade by its usual name is going to lead critics to say "weak model", with justification, when the performance problems become non-trivial. A weak model is one which does not reasonably preclude the possibility of another model which retains an equivalent, or smaller, degree of complexity to that of the weak one, and which models, or describes, the situation better. An example of a weak model is a transformational grammar for modelling the MITRE program (Zwicky, 1965). We shall see later that a translation system is a better model for this program.

To the extent that a human uses some internalized grammar in the same way as he copes with non-grammatical problems, it is relatively uninteresting to invoke the dichotomy, beyond using it for temporary purposes, like a movable lamp, to focus attention on interesting features of behavior.

3.2 Generalities

Perceptive automata (and I do not exclude animals) deal, not with reality, but with representations of reality. The question of how and why such representations are brought about <u>may</u> be dealt with by translation theory, but as this approach eventually leads us into an infinite regress, for the moment we say rather that the representations are brought about by perception mechanisms. That this is a good attitude for a programmer is supported by the observation that it is the engineer's job to produce efficient readers, cameras and microphones. At the other end are effective output mechanisms: printers, plotters, punches, displays and loudspeakers. Therefore we delimit our attention to a programmer's theory of translation.

The <u>primitives</u> for this theory are representations, and significant features of representations. The <u>syntactic problem</u> for representations is to find sets of criteria, or rules, for recognizing possible significant features of a representation. A grammar is any set of such rules. The semantic problem is to find correspondences between rules in different grammars, to facilitate translation of representations into other representations.

It will suffice for the moment that we embed our perceptive automaton in a one-dimensional universe. The general notions of recognition, combination, association, and generative identity will all be exhibited in the context of the Turing Machine model. If we are to extend our interest to higher dimensionality, we need not abandon the general notions, only the Turing Machine with a one-dimensional tape. Certainly it is possible to map the plane onto the line, or for that matter so to map any hyperspace. But the well-known (to topologists) absence of a continuous such mapping (that is, the images under no such mapping, of points arbitrarily close on the plane, can be guaranteed to be arbitrarily close on the line) suggests the inelegance, if not the inefficiency of such a mapping. And elegance and efficiency are the best criteria of good models, for practical users of models: elegance for ease of understanding; and efficiency, that the model may survive in competition with other models, in an environment where only results count.

3.3 Grammars

With Chomsky (Chomsky, 1959), we stress that we are concerned with different classes of grammars. In fact, we use exactly three, and doubt whether, for the immediate future of translation theory, any more are needed.

The reasons for having systematic grammars are that they afford a means of storing information economically, and also (more importantly) they display criteria common to different rules, remembering that we called sets of criteria rules. For example, phrase-structure rules have in common the notion of juxtaposition, and phrase-structure analysis algorithms make effective use of this feature, making no distinction between the way in which a preposition next to a noun phrase is recognized as a prepositional phrase, and that in which a noun phrase next to a verb phrase is recognized as a sentence. Less obvious is a similar relation between the problem of sentences that use the word "respectively" and the intractable nature of agreement in number; we shall show how the one grammar readily describes (and gives the mechanism for solving) these problems.

3.4 Finite-State Grammars

We count three phenomena as important to the translation process in a one-dimensional universe. The first is the ability of a machine to recognize an object, for our purposes a string of symbols. The corresponding grammar for this process is called a finite-state (FS) one, where the rules simply specify, given the input symbol and the state the automaton is in, what state the machine will enter. When the machine is in state S, it has recognized an object.

As this machine is often described as

operating in reverse, we shall consider this too. Starting in state S, the machine emits symbols as it changes states. The same grammar used for the recognition machine will serve for the generating machine.

A rule for a FS grammar, or an instruction for either of the above two machines, consists of a pair of states and a symbol. We shall, for uniformity, use Chomsky's notation, which corresponds to instructions for the second machine above, e.g.,

$$S \to aM$$
$$M \to bM$$
$$M \to c \quad \text{etc.}$$

In the last rule, the terminal state for the second machine (and the starting state for the first) is written as the null symbol. This asymmetric choice of terminology strongly reflects Chomsky's asymmetric approach to grammars. He sees them, not as recognition automata programs, but solely as generative mechanisms. In translation theory, the emphasis is on the essential symmetry of a formal communication process since all practical computer programs must be able to speak <u>and</u> hear. And we do not necessarily require automata to hear <u>by</u> speaking, as is suggested by the analysis-by-synthesis school (Matthews, 1961; Petrick, 1965) and by advocates of top-to-bottom parsing.

3.5 Context-Free Grammars

The second phenomenon is the ability to use the results of recognition recursively, that is, having recognized each component of a string of strings, to recognize the whole string in those terms; equivalently, the ability to use several recognition states in the same way as input or output symbols.

The appropriate grammar is a context-free one. The rules must, therefore, allow for the input or output symbols to be recognition states. In addition to rules of the form $A \rightarrow bC$, we must allow $A \rightarrow DC$. Again we are assuming, with Chomsky, a generative automaton, rather than a cognitive one.

The more general form of the rule is

$A \rightarrow BC...EF$, that is, any number of symbols may replace one. It is easy to reduce such a rule to a set of equivalent rules producing only two symbols each, by mentioning explicitly each state an automaton goes into when generating or recognizing strings of non-terminals. For example, $A \rightarrow BCD$ becomes $A \rightarrow BX$ $X \rightarrow CD$. For the computer program described later, we adopt the two-symbol form explicitly. Most practical parsers achieve this implicitly; in looking for A, using say a rule $A \rightarrow BCDE$, it is sufficient to start the search by looking for B, and then BC, and so on, without simultaneously looking for, say, DE.

The necessary mechanism for an automaton whose program is a CF grammar is, in addition to that for the FS automaton (FSA), a place where a fact (that the string just read has been recognized or that a string, for which this is a starting state, must be generated later) can be put to one side while the machine proceeds to recognize or generate more symbols. In the process of trying to enter a state (a goal) for which this fact (non-terminal symbol, or recognition-state symbol) is a meaningful input, as determined by the grammar, other facts may need to be stored and retrieved, as necessary for various subgoals of the above goal. It is very convenient, from both a designer's and a user's point of view, if all facts can be stored in the same place, in the same way, and likewise retrieved from the same place. The simplest mechanism for achieving this is a push-down store, analogous to the spring-loaded stacks of plates or trays found in cafeterias where only the topmost plate is accessible. The spring is inessential to the analogy - the topmost dish of any stack of dishes is more accessible than the rest.

Provided the facts required for subgoals can be used or otherwise disposed of before the fact required for the goal (above) is required, they will not be in the way when the latter fact is needed. This is the case for the problem stated, that is, recognition of a string in terms of the recognition of its substrings.

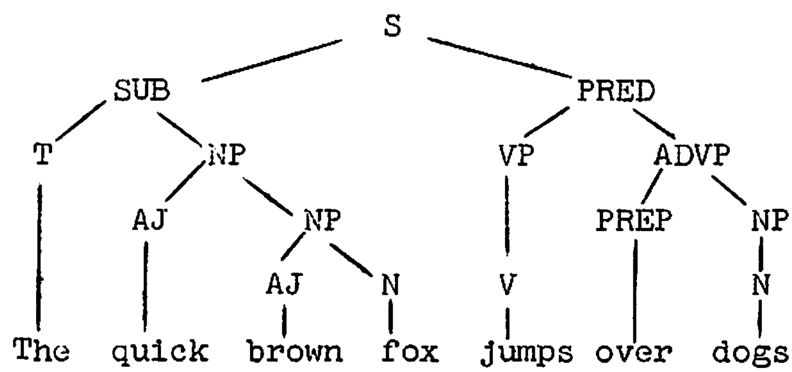
A different view of the same subject is to consider the communication channels available to the sort of machinery we are considering. This is a very good view, as it makes many hard-to-prove theorems about automata beautifully transparent.

The only channels that are obvious are: between the current input or output symbol (more accurately, the medium in which it is embedded) and the machine; and between the machine and the top element of the push-down stack. Any correspondence between, say, two symbols in a derivation, must be accounted for in terms of a set of signals sent through these channels. Equivalently, given a structure diagram as a representation of the operation of the machine, such a correspondence must appear as a path through the nodes of the diagram, along the connecting lines.

In this light, symbols put on the pushdown stack are no more than bearers of information, for one or more potential paths through nodes bearing that symbol in a structure diagram. The greater the number of independent paths that may pass through a node, the greater the variety of non-terminal symbols required in the vocabulary of the grammar.

3.6 Problems with CF Grammars

Consider the following diagram:

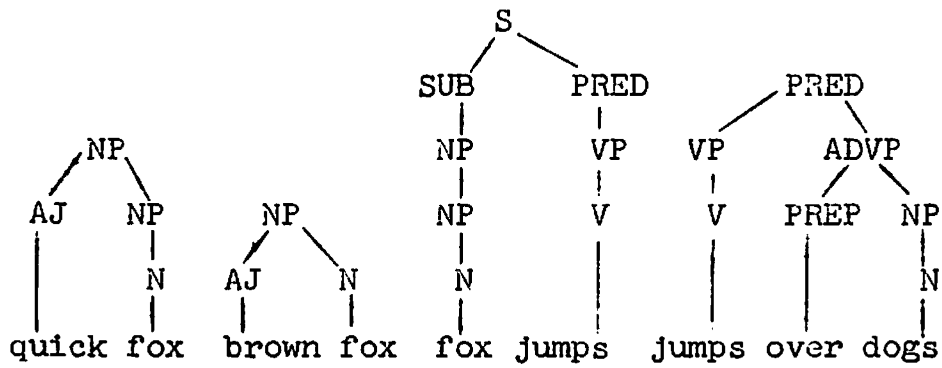


Among many interesting paths is the agreement-in-number path. This concerns "fox" (plural: foxes) and "jumps" (plural: jump), in this example. The shortest path between the two words involves ~~seven~~ eight STET different non-terminal symbols. A more elaborate diagram, corresponding to a more elaborate sentence and/or grammar, might involve many more. To enable each node to bear this information, we must label them all singular. To allow for plural sentences we must add at least another seven nodes, labelled plural, to the vocabulary.

If, in addition, we wished to verify that it is reasonable to expect foxes to jump, we invoke independent paths. The variety here is enormous; all sorts of features of foxes and jumping might be relevant. If five independent paths are involved, say, each representing a simple yes-no lexical feature (Chomsky, 1965, p.82), we must allow for 32 (= 2^5) varieties of each of the original 14 symbols, a total of 448 symbols for a simple noun-verb comparison, not to mention at least that many rules, if not two or three times more (since many non-terminals in a grammar appear at least twice on the left of a rule).

Chomsky encountered problems with context-free grammars which essentially can be viewed in this way. Chomsky's answer was to change the structure diagram (a reverse transformation) so that every interesting path could be shortened. Which paths are interesting is difficult to say; a fairly complicated path would be needed to deal at the grammatical level with "the quick brown fox jumps over skyscrapers", if it is felt that foxes cannot jump over skyscrapers.

The reverse application of transformation rules would reveal something along the lines of the following:



This shows clearly how our previous structure diagram has simply been exploded, to reveal which might be the interesting paths. This sort of demonstration, despite the obvious departures from rigid structural requirements (e.g., the "excess" structure in the first noun phrase), is more revealing of the spirit of transformational grammars than misleading arguments about, e.g., passive sentences and word permutation with context-sensitive grammars.

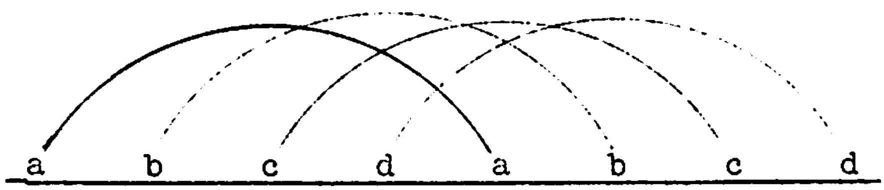
In each substructure, the structures are no longer of interest, only the relationships between the terminals. It would seem from Chomsky's account that base phrase-markers more or less correspond to these features and relationships. Since the notion of structure does not seem relevant to base phrase-markers, we are inclined to agree with Thorne (1967) that base phrase-markers should be accounted for with a finite-state grammar.

The relationship of base phrase-markers to kernel sentences and to noun phrases, say, should not, ideally, favour either. A base phrase-marker should not favour "The fox is quick" over "The quick fox", since both seem equally dependent on it. Unfortunately, Chomsky's absolute dependence on structure forces him to adopt one or the other (the former) when generating the base phrase-marker with a CF grammar. Had Chomsky been aware of the structure fallacy, he might have abandoned a context-free base component, and simply generated compact unstructured base elements with a FS grammar, which could be mapped into structures, if there were a need for surface structures as well as for sentences. But there is no *a priori* need to rely entirely on the notion of structure.

So far we have had occasion to criticize CF grammars, without condemning them entirely as inadequate. Assuming that English was not a growing language, and that we had found a context-free grammar with an astronomical vocabulary, that generated English sentences, it would still be possible to recognize sentences very efficiently. With fast table-look-up features, e.g., hash addressing (Peterson, 1957), many recognition algorithms are unaffected by the kind of grammar extensions implied by multi-path considerations. The only hardware extension would be the use of random-access mass storage; while this is expensive and marginally slower than small memories, a typical CF algorithm would still be much faster than techniques that use analysis by synthesis (Matthews, 1961).

Unfortunately, this is not the case; the vocabulary required is not astronomical, it is infinite. This is very easily and beautifully demonstrated with the path-oriented approach. The demonstration is, approximately, the graph-theoretic version of the proof suggested by Chomsky (Chomsky, 1959, p.151) that the language $\{ xx | x \text{ is any string} \}$ is not context-free. This language is of interest, as it reflects the essence of sentences of the form: Tom, Dick and Harry like Peter, Paul and Mary respectively.

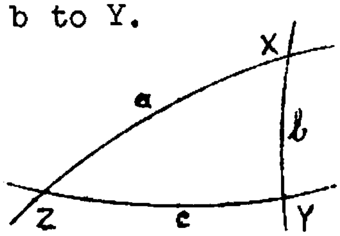
In any string xx in this language, there must be a path between the ith symbols in each half of the matched pair of strings, to account for the fact that they are matched.



We impose the reasonable restriction on the paths, that they do not go below the line of the sentence. It is clear that every pair of paths must have at least one node common to both paths.

We assert that there is a node common

to all paths. For if not, let path b cross path a at node X, and path b cross path c at Y, such that $Y \neq X$. Let path a cross path c at Z. Then there is a loop, from Y via c to Z, via a to X, via b to Y.



But a property of a tree is that it has no loops. A structure diagram is a tree, hence we have a contradiction, since each path must be part of the structure diagram.

Each path is clearly independent. If m terminals are involved, each path must be of variety m, that is, it must bear enough information to allow for m possibilities. If there are n paths through the common node, there must be at least m^n symbols in the vocabulary. We have imposed no bound on the length of xx, hence none on the number of paths. Thus the vocabulary must be infinite, as must the number of production rules for the vocabulary.

Thus, by graphic means, we may agree with

Chomsky's observation (Chomsky, 1959, p.151), that "it can be shown".

3.7 Indexed Grammars

The third phenomenon concerns the association of objects which may be quite remote. We considered in the previous section how this phenomenon could not be accounted for adequately with a CF grammar.

So far, we have endeavoured to deal with grammars that readily lend themselves to possible translation algorithms. Like Chomsky, we are concerned at the inadequacy of CF grammars in producing surface structures bearing much information, or equivalently, at the cost of automata with unboundedly many states.

Unlike Chomsky, we wish to disturb the status quo as little as possible in proposing mechanisms for solving these problems, since in all other respects the status quo is very satisfactory, both from a recognition and a translation viewpoint.

Therefore we shall not abandon CF grammars, but simply extend them, in a way reminiscent of extended phrase-structure grammar (Harman, 1963). Since Harman's suggestion, and its vigorous criticism (Chomsky, 1966, p.40), the theoretical situation has improved.

It is shown (Aho, 1968) that the use of indices, as a means of increasing node capacity in a structure diagram, or equivalently, of increasing the variety of non-terminals without bound, produces a grammar that is more powerful than a context-free grammar, in that the class of indexed languages properly contains that of CF languages. Moreover, the class of context-sensitive languages properly contains that of indexed languages, suggesting that recognition may be less painful with an indexed grammar than with a context-sensitive one. This is discussed in the chapter on recognition.

We shall attempt to give the reader an intuitive feel for indexed grammars. Our notation differs slightly from Aho's, but is nevertheless equivalent.

As with any phrase-structure language, we have a finite vocabulary V of symbols, a finite number of rewriting rules, and a starting symbol S. We impose a partitioning on V, into terminals and non-terminals, according as the symbols of V cannot or can be rewritten, again as for any PS grammar. We denote the semi-group concatenation operator by the non-vocabulary symbol + . (This will be seen to be needed as a delimiter, for the sake of clarity if not the prevention of ambiguity because of the other semi-group operation below. It is not entirely unrelated to the arithmetic addition operator, which it resembles.)

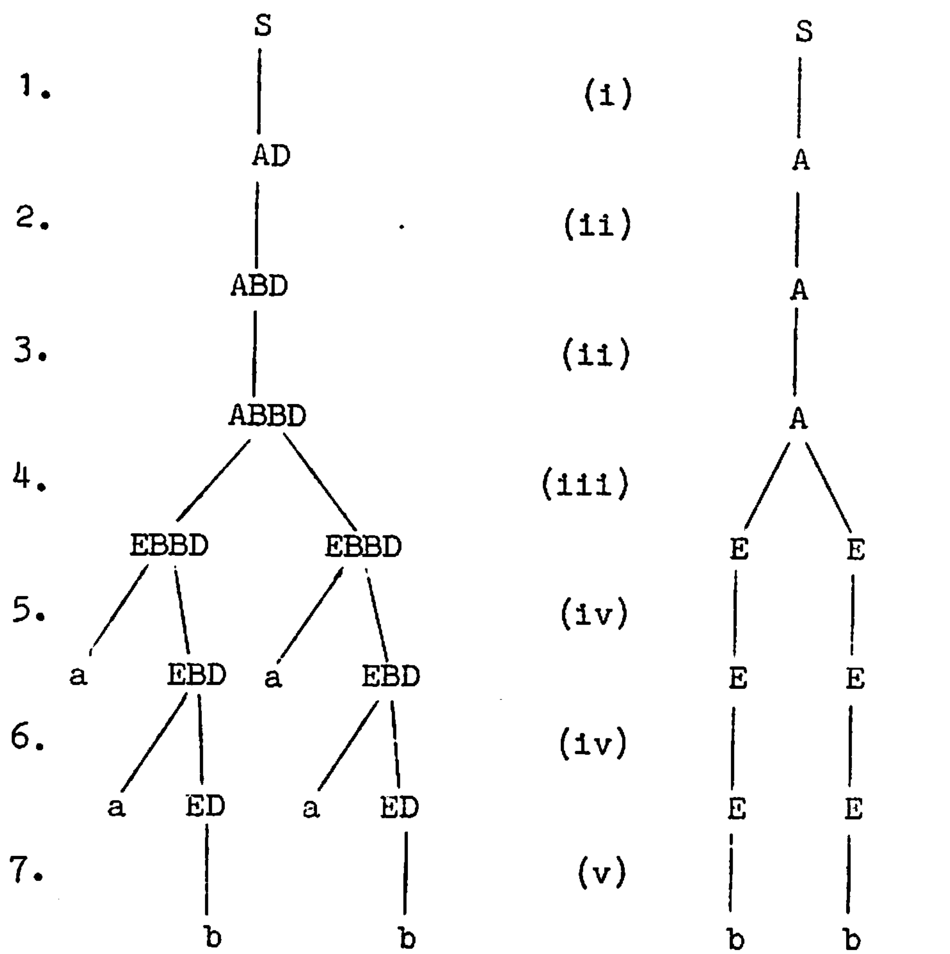
The crucial difference between conventional phrase-structure grammars and indexed grammars is that, for each symbol appearing in a PS derivation, a set of symbols (technically, an element of a semi-group with a semi-group operator distinguished from the above one - we distinguish it by omitting it, i.e., using the null symbol, and so it resembles the multiplication operator) is found in the equivalent indexed-grammar derivation. In a structure diagram for a PS analysis, each node is characterised by a single symbol, which amounts to the only description there is of a node. In that of an indexed-grammar structure diagram, a node is characterised by an unbounded string of symbols, thus allowing unbounded variety in the description of a node.

The mechanism is best exhibited by demonstrating an example of a simple indexed grammar, and a structural description of a sentence in the corresponding language. To make it easier to see the connection with context-free languages, we exhibit simultaneously a rather trivial CF grammar derived in an obvious way from the indexed grammar.

Indexed Grammar		A Corresponding CF Grammar
S → AD	(i)	S → A
A → AB	(ii)	A → A
A → E + E	(iii)	A → EE
EB → a + E	(iv)	E → E
ED → b	(v)	E → b

Derivation of aabaab — Indexed Grammar

Derivation of bb — Context-Free Grammar



To achieve the correspondence, we have had to let the CF grammar idle while the other worked.

There are four features worth noting here.

(a) The ability to generate variety, for nodes. This is achieved using the first two rules. The mechanism should be obvious, as it is the same mechanism, essentially, as for a pushdown memory whose point of access is on the left. Thus, the rewriting rule, both here and for all other rules, is that the leftmost symbol must be included in the rewrite process. (This point is easier to make in Aho's characterisation.) Symbols not rewritten remain untouched.

(b) The ability to distribute index symbols not rewritten. An analogy for this is to say, if pets consist of dogs and cats, then big red pets consist obviously of big red dogs and big red cats. At step 4 of the derivation, we see precisely this situation where BBD is the description of the rewritten node. Another analogy is the distributive axiom in algebra, where $(a + b)c = ac + bc$. Thus the symbol + is not entirely unmotivated.

(c) The ability to consume (Aho's terminology) indices, as demonstrated in rules (iv) and (v).

(d) The convention that abandons indices attached to terminals. For those whose mathematical upbringing causes them to shudder at this convention, the alternative is to regard them as all still there, but lined up behind the terminal vertically to the page. (Naturally, this must then be done also with the other nodes.)

Formally we define the set of rules to be a finite subset of $(V_N^+ \times (V_T \cup V_N^+)^\cdot)$. That is, a rule is an ordered pair (a,b), whose interpretation is a $\rightarrow$ b, such that a is a non-null string of non-terminals and b is the non-null concatenation of objects, each of which may be either a terminal or a non-null string of non-terminals. The distinction between string and concatenation, and between $^+$ and $^\cdot$, is exactly the same as that pointed out earlier between the two concatenation operators. We adopt all this terminology purely for convenience.

It is worth noting that Aho distinguishes between symbols that always appear as the leftmost element of a node (non-terminals) and those that always appear to the right of non-terminals (indices) by writing the latter in lower-case letters, e.g., Affgfh. This has the advantage that one can distinguish the start of each node in a production.

On the other hand, if we now remove the + from rules, we may confuse terminals with indices, unless we impose a partitioning on the alphabet to distinguish them. In practice, we shall be using somewhat verbose grammatical terminology for non-terminals and indices, and the particular use of + that we have adopted seems to make the situation clearest. Furthermore, whether the leftmost element of a node is called a non-terminal or an index is purely a matter of taste.

The automaton that Aho prefers for accepting exactly the class of indexed languages is a nested stack automaton. Since we assume some familiarity of the reader with programming (this being a programmer's theory of translation), we offer a list-processor equivalent.

A LISP-type list element is a pair of, say, computer words. The first word may contain a symbol (atom) or a pointer to another list element (list). The second contains a pointer to another list element. A special end-of-list element (nil) is always available for terminating lists, but for no other use.

A push-down stack in this context is simply a sequence of list elements, each pointing to its successor, and the last pointing to nil, such that every element contains a symbol in its first word. Thus, the memory used by a computer that, say, generated random sentences using a CF grammar, would be what is called a single-level list.

The necessary change to this structure is to permit two-level lists, if sentences randomly generated by an indexed grammar are required. That is, the first word of each element in the original push-down stack is no longer a symbol, but a pointer to a single-level list, or conventional push-down stack.

With single-level lists, the notion of shared lists is not meaningful, since, with only one list, there is no need to share. With a two-level list, there are arbitrarily many one-level lists, and sharing becomes meaningful. Consider the rule AE → B + CD. It means, take the first list off the stack S, call it X (compare this with

A → BC for a pushdown stack automaton, which starts, take the first symbol off the stack...); take the first two elements off list X, checking that they are A and E respectively; form a list Y, which is (C, D, X); put list Y on stack S; form a list Z, which is (B, X); put list Z on stack S. In this case, lists Y and Z share list X.

It is of course possible in a computer to have lists embedded in lists to any level. However, there is a penalty. In the above example, we had to have space in the FS automaton to keep track of S and Y (and Z, but we could without confusion have used Y for Z, since the stack itself, not Y and Z, ultimately is responsible for keeping track of these lists). If we had a 3-level list, we would need 3 variables, and so on. Thus, the size of the particular automaton in question sets an upperbound on the number of levels we can use, without appealing for another source of unbounded memory, such as another push-down stack. And once a machine has two independent push-down stacks, it becomes a Turing Machine, since it can use them as if they were a single tape.

For dealing with English, we are content to use a conceptual automaton that has enough memory to cope with a two-level list structure. Most of the post-Chomsky phrase-structure discussion of languages has dealt with single-level lists as the memory attached to a finite-state automaton. In extending our attention to the next level, some problems related to structure suddenly became solvable. Possibly all the structure-related problems for one-dimensional languages can be shown to be readily solved with indexed grammars, though this conjecture is based on nothing stronger than intuition. However, not all grammarians confine their attention to one-dimensional languages; there are various syntactically oriented picture-processing schools of thought, involving media of higher dimensionality. A question worthy of their attention is, must they make use of more powerful languages than the linguists, or do indexed grammars also solve their problems. Again, intuitively, probably the former; a three-level list for two-dimensional pictures, and so on.

3.8 Generative Identity

In a transformation rule, Chomsky is careful to identify each element participating in a structural change, using positive integers for tags. (The "X" is irrelevant.)

e.g., Structural Change:

$$X_1 - X_2 - X_3 - X_4 \rightarrow X_4 - X_2 - \text{be} - \text{en} - X_3 - \text{by} - X_1 .$$

Note that not every object gets, or needs, a number, even if we were to reverse the direction of the arrow (assuming we could generate the right hand side) and transform the other way.

In most of the correspondences set up between grammatical rules, in a translation system, such a notation is adequate. However, without a clear understanding of the exact theory underlying this practice, it is difficult to set up a translation mechanism to handle the "respectively" problem, despite the obvious fact that indexed grammars would have to be the minimal generative mechanism at the syntactic level.

There are two essential points. The first deals with the rephrasing of a transformation rule as a translation system correspondence. Not all of Chomsky's rules can be thus rephrased (at least not readily), but the active-passive example works well.

S →	NP	Aux	VP	NP	S →	NP	Aux	be	en	VP	by	NP
A	A.1	A.2	A.3	A.4	A	A.4	A.2	0	0	A.3	0	A.1

The exhibited correspondence is between a rule in a grammar of active sentences, and a rule in one of passive.

The second point deals with the explication, and source, of the unfamiliar notation beneath the rules given above. The integer tags are obviously related to Chomsky's notation. In full, however, the line reads:

"The left hand side (S) of the first rule is acknowledged to have its own identity, which we tag A. In replacing S, each component assumes the identity of S, and in addition a tag of its own to distinguish it from its siblings."

This process should not be confused with the notion of family name. Given the rule

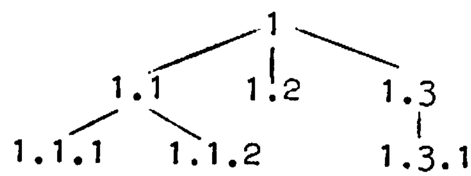
NP → T ADJ NP

A A.1 A.2 A.3

the ADJ "big" in the derivation of "The big boy can eat a horse" has the identity 1.1.2, where the noun phrase NP ("the big boy") has the identity 1.1 and the sentence S has the identity 1. Thus, surnames "grow" as the derivation proceeds.

The "O" indicates that this object needs no identity. In the description of the universal translation algorithm, this will become more apparent.

The source of this notation is Brainerd (Brainerd, 1969), although he notes earlier users. Brainerd needs to identify nodes in tree structures in order to rewrite them. Since the distinction between trees and generative phrase-structure processes is somewhat fine (mainly one of a choice of either time or space coordinates), it takes little imagination to see the obvious application of Brainerd's notation to a grammar. The above account should make it unnecessary to say anything about Brainerd's notation, except to reproduce approximately a diagram from his paper which illustrates a tree with identifiers attached.



(We have departed slightly from Brainerd's notation, in assigning a specific identity to the root, rather than the null element. This simply makes it possible to refer in print to the identity of the root without confusion. The root may be any positive integer.)

Not all rules simply rewrite one element. Consider

X	Y	→	F	G	H
A	B		A.1	A.B	A.2.1

This is an example of two objects with separate identities combining to produce a mixed batch of offspring . Two of them (F and H) acknowledge only one source. H assumes two integer tags. The middle element acknowledges the identity of both the

X and the Y, and moreover feels no need for further tagging with integers.

Not all transferences of identity imply an increase in the length of the identifier, e.g.,

NP	→	ADJ	NP
A		A.1	A

Here, the adjective acknowledges its inferiority to the rewritten noun phrase, but the residual noun phrase maintains it is as equal as its predecessor. This is not idle animism, but in fact a powerful tool available to the translation process. And for those who set store by "proper" structural descriptions, this rule should be compared with the corresponding rule in the final picture grammar given in the section on the structure fallacy. The similarity should be striking. (If not, consider

NP	→	ADJ	NP
A		A.1	A.2

and compare it with the first "fallacious" picture grammar.)

Though it is possible to do without arithmetic in this theory, we shall not hesitate to use it in order to keep down the length of identifiers, noting that most computers can perform addition readily. The meaning of

$$\begin{array}{llll} X & \rightarrow & Y & X \\ A & & A & A+1 \end{array}$$

should be transparent. We are here generating a string of Y's, with increasing numerical identifiers all of the same length. For instance, the above example for noun phrases would be better expressed as

$$\begin{array}{llll} NP & \rightarrow & ADJ & NP \\ A & & A & A+1 \end{array}$$

to enable each adjective to be distinguished. Rephrasing this in structural terms, we have a mechanism for generating <u>immediate</u> constituents without bound, if we rephrase the notion of immediate constituency in identification terms. We may say, if x is an identifier of an object, then x.i is an identifier of an immediate constituent of that object if and only if i is a positive integer (i.e., an identifier of length 1).

We have suggested that, for purposes beyond simple generative or cognitive ones, the usual notion of a phrase-structure rule is inadequate. We follow Chomsky's theory in invoking identity-markers, and we depart from it in embedding them in the phrase-structure component of our translation system. Our rules now deal both with syntactic markers and identity markers. A fringe benefit is the possibility of a relation between Chomsky's notion of "correct" surface structure, and the identity component of these extended rules.

3.9 A Universal Translation Algorithm

Much is either available in the literature, or is intuitively obvious, about the functioning of automata that recognize, and automata that generate, strings. There is very little about the more formal aspects of connecting one of each kind together so that, given a string in the language accepted by one automaton, the other automaton can be constrained to generate a corresponding string in its language. On the other hand, there is no end to the amount of informal literature on the subject, in the fields of program-compiling, so-called mechanical translation (of natural languages) and even transformational grammars, which in certain respects resemble our formal translation systems. The most formal paper to date on this problem would appear to be Lewis and Stearns (1968). However, it deals with the theoretical aspects of problems that practical compiler writers had to solve informally years ago. Our concern is not only with formalizing informal solutions, but with finding any sort of solution to some problems not even solvable with transformational grammars. If our solutions approach some degree of formality, then it becomes easier to describe, evaluate, compare, use and change the solutions.

For the algorithm, we distinguish the source grammar and the target grammar, and likewise the source and target automata and strings respectively. The source automaton's role is to set up a theory of how it might have generated the source string had it been operating in its generative mode. The target automaton starts anytime, even before the source automaton if it wishes, and proceeds to generate a string of its language until a decision has to be made. It then consults the source automaton's theory to see what it would have done at the corresponding stage. There are two independent considerations. Th. first deals with whether the source automaton says that any more theories are likely about what it would have done at this stage. The second deals with the number of theories about that stage. We tabulate the corresponding responses of the target automaton:

No. of theories	No more theories	More theories to come
0	Takes evasive action.	Waits.
1	Takes this theory.	Takes this theory and notes place.
>1	Takes best theory and notes place.	Takes best theory and notes place.

Consider the first column, no more theories. If there are no theories about this stage, something has gone wrong. What form evasive action takes is a matter for a particular implementation. The simplest action is to terminate generation of the current string, generate information about the current stage, and about the most recent stage which appealed to the source automaton's theory, and then continue as if it had finished generating the current string.

If there is exactly one theory, the course is evident. If several theories, the best should be selected (or the first if there is no difference). When other theories also seem promising, this should be noted.

The second column is included for the case where the target automaton wishes to proceed as fast as possible. Only the second line should need comment: A theory about a stage need not be unique, if more theories about this stage are possible.

For a compiler, where the source language is presumed unambiguous (regardless of whether it actually is), only the first two lines of the table need be used. If, in addition to being unambiguous, the language is LR(k) (Knuth, 1965), that is, the source automaton need only stay k symbols ahead of a point in the source string to be sure that there are no more theories relevant to that point, for some fixed k, then only the first column need be used. In practice, when k is finite, k is rarely very large, for programming languages.

For natural languages, ambiguity is a non-trivial problem. We shall consider it further in the chapter on recognition; here we note that it corresponds to the contingency for which the third line of the table is provided.

In the particular implementation of this algorithm used for translating syllogisms on a PDP-8, we used Younger's algorithm to construct hypotheses. Some of these were then confirmed, thus becoming theories, simultaneously with the operation of the target automaton. The program caters for all contingencies in the above table, although this observation will receive qualification in Chapter 5.

The translation algorithm given so far is very general, and assumes little about the nature of grammar, beyond the fact that it be generative. This is not a particularly onerous restriction, since many mathematical theories of language to date have been phrased generatively.

We now restrict our attention to translation between languages with phrase-structure grammars, since recognition algorithms for these are quite efficient, in comparison to recognition of deep-structure features of English sentences using existing transformational theories.

So far we have not explained how to locate stages in the operation of the source automaton, so that we may consider theories about that stage. We now define a stage in phrase-structure terms; it is simply the application of a rewriting rule. If we restrict our attention even further, to context-sensitive grammars, in which a rule rewrites just one symbol, we may now identify a stage using the identifier of the symbol rewritten. The only decision the target automaton has to make is which rule to use to rewrite the symbol it is currently contemplating, it chooses the rule that corresponds to the rule chosen ("theory") by the source automaton at the stage corresponding to the contemplated symbol. Thus, where a grammatical rule might be involved in a choice, it must if possible be put into correspondence with one or more rules of any potential source grammars in the translation system. The system at the start of this chapter demonstrates a complete one-to-one correspondence; in practice we need not expect this, except for parsing and compiling systems.

3.10 Applications

In this section, we demonstrate a simple translation system, to give the reader an intuitive feel for the translation algorithm, and also to show how elegantly it can solve problems not even catered for by transformational theory.

The Respectively Problem

we have set up, in the previous three sections, enough mechanisms to translate between, say, "John and Bill like Mary and Joan respectively" and "John likes Mary and Bill likes Joan". We could equally well have chosen, in place of the second sentence, the deep structure of the first sentence, but since translation theory makes no distinction between structural-description grammars (e.g., in section 3.1) and any other kind, we invent a kernel-sentence grammar instead, noting that it would not be very hard to change this grammar to produce deep structures.

We set up a simple system sufficient for a demonstration:

```
Sentence → Kernel Next Ult   (A "2-path" sentence)
   A          A.1   2   1

Kernel Next → Kernel Next Next (Generates more "paths")
   A      B      A+1    B+1  B

Kernel → Noungen + Verb Plural + Noungen + respectively
   A        A.1      A.2   0       A.3         0
                               (Shape of kernel sentences)
Noungen Next → Nounphrase + Noungen
   A       B       A.B          A
                                 (Generates Nounphrases)
```

Noungen Ult → and + Nounphrase (Last Nounphrase)

A B 0 A.B

Nounphrase → John, Bill, Mary, Joan

0 0 0 0

Verb Plural → likes

0

Verb Sing → likes

0

Let us generate, step by step, the first sample sentence. For brevity we shall write Ng for Noungen, etc..

Sentence

1

Kernel Nx Ul

1.1 2 1

Ng Nx Ul + Vb Pl Nx Ul + Ng Nx Ul + resp

omit space

1.1.1 2 1 1.1.2 0 2 1 1.1.3 2 1 0

Np Ul + Ng Ul + like + Np Ul + Ng Ul + resp

1.1.1.2 1 1.1.1 1 0 1.1.3.2 1 1.1.3 1 0

Np Ul + and + Np + like + Np Ul + and

1.1.1.2 1 0 1.1.1.1 0 1.1.3.2 1 0

										+	Np	+	resp	
											1.1.3.1		0	
John	+	and	+	Bill	+	like	+	Mary	+	and	+	Joan	+	resp
0		0		0		0		0		0		0		0

This can now be taken as a theory of how the first sentence might have been generated. We now attempt to generate its translation. Only the third, fourth and fifth rules in the previous grammar need be changed to produce the target grammar.

Kernel	→	Kernelgen
A		A

Kernelgen	Next	→					
A	B						
Nounphrase	+	Verb	Sing	+	Nounphrase	+	Kernelgen
A.1.B		A.2	0		A.3.B		A

Kernelgen	Ult	→					
A	B						
and	+	Nounphrase	+	Verb	Sing	+	Nounphrase
0		A.1.B		A.2	0		A.3.B

In setting up a correspondence between the rules of the source and target grammars, it should be noted that no correspondence is needed between the fourth rule of each, nor between the fifth, nor would any correspondence have significance, beyond the fact that they both consume indices similarly. For the other rules, the correspondence should be obvious. We now generate the second sample sentence, using the above theory.

Sentence

1

Kernel Nx Ul

1.1 2 1

At this point, a decision (whether to apply the second or third rule) is necessary. For object 1.1 in the theory, we applied the third rule, so we do likewise here, and then we perform several steps for which no decisions are needed, but for which there are no steps in the theory that correspond sufficiently for us to keep track of identities in the conventional way.

Kg Nx Ul
1.1 2 1

Np Ul + Verb Sing Ul + Np Ul + Kg Ul
1.1.1.2 1 1.1.2 0 1 1.1.3.2 1 1.1 1

Np Ul + Verb Sing Ul + Np Ul + and +
1.1.1.2 1 1.1.2 0 1 1.1.3.2 1 0

Np + Verb Sing + Np
1.1.1.1 1.1.2 0 1.1.3.1

Now we have a choice of rules for Nounphrases, for each Np above. Although there is no choice for singular verbs, it is clear that the identity has been preserved to enable the choice of the singular form of like to be correctly made, if there were other singular verbs. The rules might have to be extended, to Verb Sing → SG + Verb , say, where
A O A

SG is a marker to be handled by the so-called post-cyclic rules of transformational theory. The details are not, however, relevant to this demonstration. Using the source automaton's theory, we have finally:

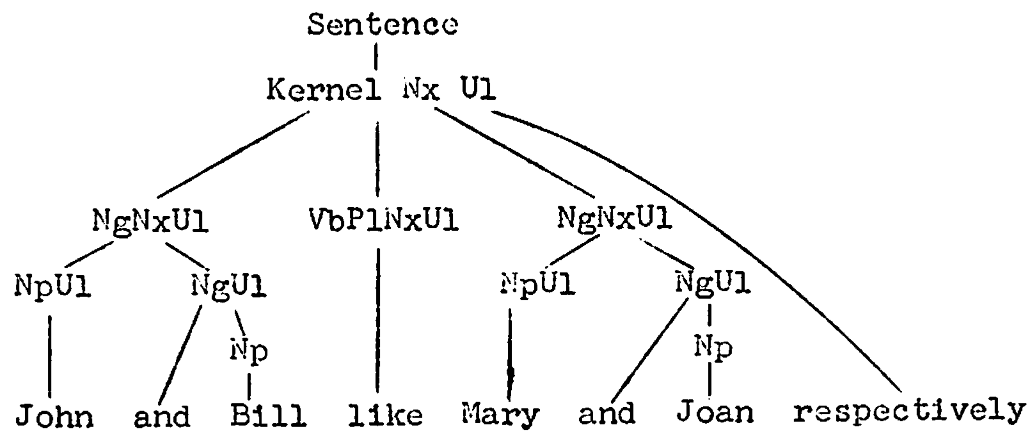
John + likes + Mary + and + Bill + likes + Joan.
0 0 0 0 0 0 0

If this demonstration seems complicated, it is because we are exhibiting each step in the translation process. It is not unreasonable to expect a computer to have to go through this many steps in performing translation. The important fact is that we have defined exactly (to within a particular implementation) what steps must be gone through in translating, once the source automaton has a theory. This particular example shows how, without the notion of generative identity, it would be difficult, if not impossible, to decide which rules to choose when rewrting the Nounphrases. We could have produced, say,

John likes Bill and Mary likes Joan

and although no appeal to complicated theories of identity are necessary to achieve this, the translation is far from plausible.

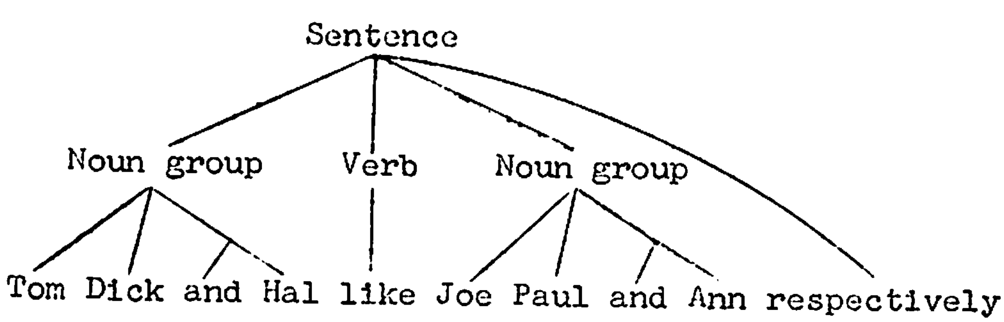
Let us turn again to our path-theoretic approach, for more insight into why translation theory handles problems not dealt with well by transformational theory. As we remarked earlier, the interesting paths cross, and uncrossing them is by no means trivial, as can be seen from an appropriate structure diagram of the first sentence:



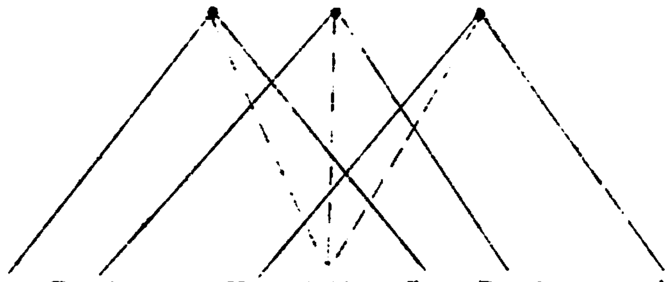
Corresponding to each path of interest is either a "Next" or an "Ult", as can be seen by tracing through the diagram and following each index. In this example, only one Next is involved, but with longer sentences, it is clear that each of the Nexts must in some way be distinguished. Identification of index symbols, in which each index in a node might have an identity, seems to provide exactly the mechanism needed for identifying paths, so that they may be successfuly untangled.

In transformational theory, as we noted, Chomsky does not attempt to identify objects except those immediately involved during the application of a single rule. We suggest that it is unlikely that transformational theory will be successful in any situation where it is clear that there are paths that cross. It is our contention that the minimum amount of machinery necessary to handle the association of remote objects generatively is some theory of identity, at least as powerful as the one used here, and some system of indices for nodes in structure-diagrams to bear identities.

One further remark on surface structure might be in order. If one were teaching a primary-school class about the use of the word "respectively", would one attempt to produce some sort of structure tree in the surface structure spirit of transformational theory, such as



or would one use



Tom Dick and Hal like Joe Paul and Ann respectively

(where the dashed line indicates sharing).

The second structure is most definitly not that of a tree. We saw in section 3.6 that if it were a tree, all the interesting structures would have a common node. Without further labelling of the diagram, a tree-like structure would hardly be of interest to any but the transformationalists, as it would not make clear the interesting structural features.

In other words, when portraying structure graphically, why must we always insist that the components of the structured object be adjacent? A radio transmitter and a receiver may display structure, in that they may form the basis of a communications system, but we would scarcely insist that they be immediately adjacent, if we were attempting to illustrate this graphically. Identity does not always imply encapsulation, and structural descriptions need not always imply trees.

Agreement

The problem of agreement in number (or for that matter, any finite number of agreements) can be handled to an extent by setting up these agreements as indices attached to a node denoting a given clause, e.g.,

Clause → Clauseno Sing

Clause → Clauseno Plural

Clauseno → Clauseatr Concrete

Clauseno → Clauseatr Abstract

are examples of rules attaching lexical features to a clause. If a clause is embedded within a clause, a delimiter and a fresh set of indices may be added. When the indices are eventually consumed, up to the delimiter, the remaining indices may be ignored, since they will be discarded when nodes carrying them are eventually rewritten as terminals. While it may seem ungainly to carry around unused indices, it will be seen in chapter 4 that recognition of sentences with such a grammar does not imply such ungainliness.

The distributive properties of indices ensure that agreement paths can be set up using indices in this way. The indices are consumed in rules of the form

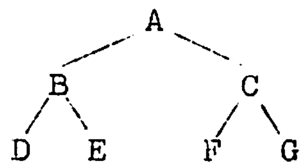
Noun Concrete Sing → horse

etc.

While it is often feasible to arrange for such agreements using indices, it is not necessarily easy, elegant, or even useful. It is felt that indices are of most benefit where they play a more obvious generative role, as in the respectively problem. Experience with the program described in Chapter 5 suggested that the only reason for checking agreement was for disambiguation, and that ambiguities that could be resolved by appeal to agreement were quite infrequent.

<u>More graphics</u>

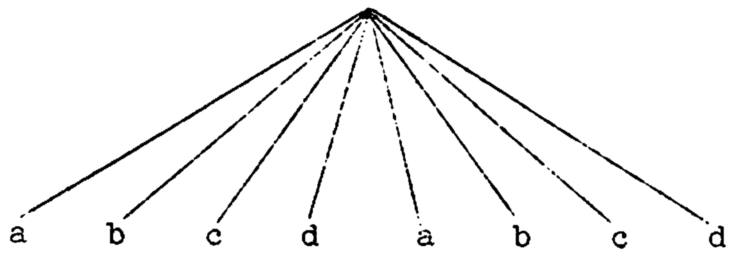
The limited access to two-level list structures does not readily permit permutations on the elements of the embedded lists, and hence on indices attached to nodes in structure diagrams. When an index is used as a communication channel, the general rule is that communication paths should be properly nested. In the diagram,



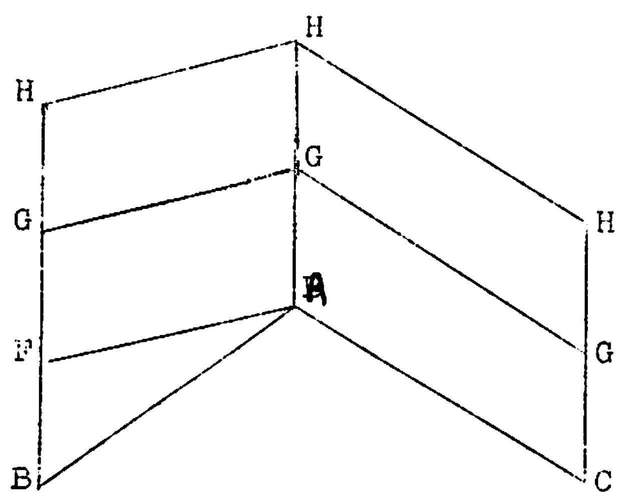
it is possible to have, simultaneously, paths from B to C, from D to G, from D to E, etc.. It is not possible to set up simultaneously independent paths from B to F and from D to C, as this implies that indices bearing identities and other information for each of these paths must be interchanged, either near B or near C. A rule that interchanges them, though quite simple to find, implies that the paths are no longer independent. There is no general rule for permuting arbitrary indices.

If difficulty is experienced in attempting to set up indexed grammars, with the intention of achieving communication between remote nodes, it may be due to attempting some improper nesting (just as FORTRAN DO loops may be improperly nested) of paths.

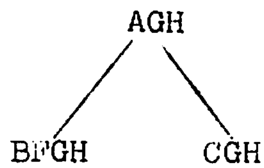
This problem does not arise in the solution to the respectively problem, since only at the common node is there any question of nesting.



To help in visualizing this, one can imagine a structure diagram as being a projection of a 3-dimensional diagram. Each line in the diagram is really the bottom edge of a plane at right angles to the paper, and each node is a line perpendicular to the paper, along which are arrayed the indices. Communication paths are straight lines drawn along the planes through the appropriate indices at each node. That indices may not be permuted corresponds to saying that these paths may not cross.



The diagram is the 3-D representation of



corresponding to the rule

$$A \rightarrow BF + C$$

with indices GH attached to A.

This conception can be helpful when deciding whether the use of indexed grammars is necessary or feasible in a given application.

Clowes, Langridge and Zatorsky (1969) point out that transformational grammars do not handle conjunctions very well. The counter-examples they give are reasonably difficult, but far more difficult are sentences such as

Mary supports John, not John, Mary. (Klima,1964,p.301)
The Chinese have short names and the Japanese long.
Jim plays guitar, Peter the drums and myself the tuba.

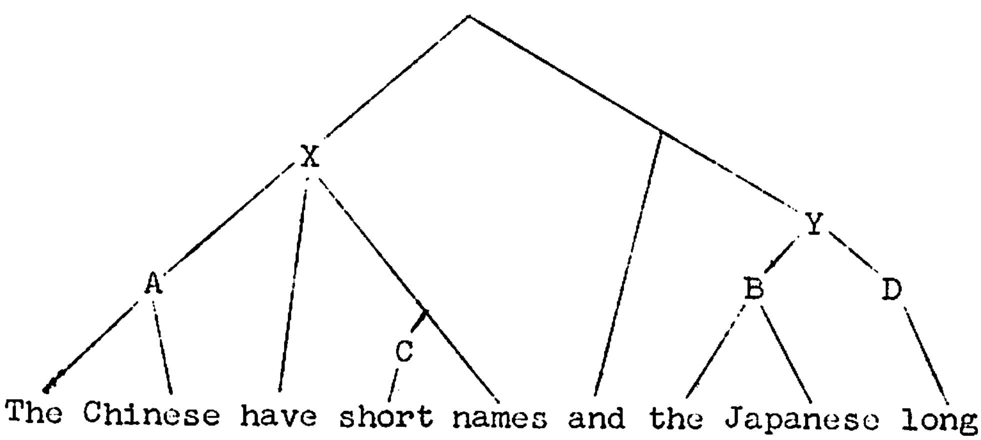
The deletion approach, that verbs and such-like have been deleted because they are repeated, is not convincing. The sentence

Peter sang "Old Man River" and John sang.

cannot be subjected to such an operation. A host of counter-examples can be found for most explanations other than that a string (in these examples, a sentence) is given, and then each subsequent string of the same syntactic class as the first is specified by supplying at least those substrings that need alteration. The specification of what syntactic classes of strings and substrings may participate in this activity appears to be manageably small.

Without demonstrating an actual solution to any of the conjunction problems, we shall show how to tell whether indexed grammars are necessary.

Consider a plausible 2-dimensional structure diagram for the second example above.



To enable a translation process to substitute "the Japanese" for "the Chinese" and simultaneously "long" for "short", a path is needed between B and A, and another between C and D. Between X and Y, these paths must lie in the same planes. However, it is clear that they need not cross within these planes. Hence it is feasible to use indices.

Since the paths cross on the 2-dimensional diagram (though not on the 3-dimensional one), it would also seem desirable, if not necessary, to use indices. The two paths are independent to a sufficient extent to make the CF treatment of their crossing very long-winded.

Not all conjunction problems need appeal to indexed grammars; notably, those accounted for by Chomsky, which is the special case of the general rule given above, where the substring that needs alteration is the whole string. In Chomsky's formulation (1957, p.35) Z + X + W are two sentences. X corresponds to some string, and Y to a string of the "same type". Chomsky insists that <u>all</u> of Y be copied, when forming Z - X + and + Y - W. Thus only a single path connecting X to Y is involved.

With no crossing paths, no appeal to indexed grammars is necessary.

This extended treatment of conjunctions still does not deal with all the problems raised by Clowes, et. al. (1969). The general problem of conjunction is quite difficult. However, problems not germane to translation theory are, for example,

John and John sold the house. (Clowes, et. al. 1969, p.10)

John is more sucessful as an artist than Bill is as an artist. (Postal, 1964, p.151)

While transformation theory appears to deal with the problem of measuring grammaticality, translation theory deals only with the problem of finding plausible translations, such as

John sold the house and John sold the house.

In transformational terms, translation theory is content to discover possible deep structures, without necessarily verifying that they satisfy lexical and other requirements. A deep structure discovered by a translation automaton does not need to be checked to see if it can be generated by the base component (again assuming transformational terminology; cf. the discussion of the MITRE program in the next section), since it had to be generated in this way to be discovered. In this case, the base component corresponds to the target grammar, in a translation system for discovering deep structures (or, equivalently, kernel sentences).

3.11 Implications

The MITRE Program

The MITRE syntactic analysis procedure for transformational grammars (Zwicky, 1965) is a program for finding deep structures of English sentences. It uses a CF "surface grammar" which generates English sentences without regard for the finer details of their grammaticality. A sentence is analyzed, and structures produced. Inverse transformations are applied to these structures to produce tentative deep structures. Then those deep structures that could not generate the original string are rejected. Petrick (1966) says that it is not known whether this technique will discover every possible deep structure for a sentence. On the other hand, Zwicky claims that no structure discovered by the MITRE Junior Grammar has failed to be discovered by this technique.

The MITRE program would appear to be an order of magnitude faster than Petrick's program. This can in part be accounted for by different machines and programming systems, but the fact that it appeals to what appears very much like a CF → CF translation theory to find deep structures seems significant. Petrick admits this, and says that his program, which includes the analysis-by-synthesis technique, is for the use of grammarians testing grammars, and hence must be guaranteed to work, whereas the MITRE grammar is relatively permanent, making it easier to ensure that it continues to work the way it does.

If it is true that some problems can be handled well by indexed grammars, and most inelegantly, if at all, by transformational theory, then the fact that some transformations have no usable inverse may be due as much to an inelegant solution to a problem better handled by indexed grammars as to any other factor. One cannot argue that transformations without inverses are a necessary evil of transformation theory, or at least of English.

This, and the fact that the MITRE program uses a translation-like approach to finding deep structures, suggests that no harm, and possibly much good, would come of recognizing and formalizing CF (and indexed) surface structure grammars as respectable components of a transformational theory, and that as much or more attention be paid them, than finding structural descriptions whose justification is in terms of descriptive or explanatory adequacy.

Psychology

Another area where transformational grammars have been considered is psycholinguistics. The Savin and Perchonock (1965) experiment is sometimes cited as evidence for a transformational explication of human processes. The experiment considers so-called immediate memory used up in memorizing simultaneously a sentence, and eight carefully chosen but unrelated words. Provided the sentence can be correctly recalled, or nearly so, the number of random words recalled is taken as a measure of the space left after memorizing the sentence. It is shown that the transformational complexity of the sentence (whether it is active or passive, affirmative or negative, declarative or interrogative,etc.) is strongly correlated with the measure of space used.

If one were to assume that, to memorize a sentence, it be recognized as a CF sentence, and then translated into an active declarative affirmative sentence for the purposes of efficient retrieval from a hypothetical data-base (which is plausible, since this is precisely how question-answering systems usually function to achieve economy and efficiency in using their data-bases for storage and retrieval), then it is possible that more immediate memory is used up if a translation is required than otherwise. Thus one is led to ask, does CF →CF (say) translation require any more memory than CF recognition.

Lewis and Stearns (1968) show that transduction (which corresponds, for LR(k) languages, to our translation) from simple infix to postfix (reverse Polish) arithmetic expressions cannot be performed using only the memory of a pushdown automaton. The process implies the ability to recognize $\{xnx \mid x \text{ some string on a finite alphabet}\}$. As we saw earlier when considering $\{xx \mid x \text{ any string}\}$ this could not be done with a pushdown automaton. If it is reasonable to deduce from this that the translation may proceed by using more immediate memory, then in fact the experiment is quite good support for a translation-oriented theory.

While this speculation on its own is not very valuable, it does mean that the results of the experiment cannot be regarded as favouring a transformational account of sentence memorizing, since a phrase-structure account by no means implies merely recognition. In fact, if only the sentence, and the fact that it had been recognized as a sentence, were memorized, it is hard to imagine how this fact could be used. One may as well memorize the string without attempting its recognition.

Clearly, some experiment that can distinguish between transformational and translational processing in humans is required. This might be no more than attempting to determine if analysis-by-synthesis is used, which seems to be the vital difference between Petrick's program and the MITRE one. If no such experiment is forthcoming, this could indicate that the results of the experiment are not particularly relevant to applied psychology, since a need for a particular fact is often sufficient in itself to suggest an experiment.

3.12 Summary

Firstly, we exhibited three classes of grammars, in increasing order of power, without reaching the power of context-sensitive grammars. The first, finite-state grammars, we saw could be used for recognition of a string in terms of its terminal symbols alone. The second, context-free grammars, could be used for recognition of a string in terms of more than one previously recognized substring. The third, indexed grammars, could be used for recognition of similarities between objects not closely related by context-free standards.

We considered the sort of memory required by the automata associated with each grammar, to show how they resembled each other. The first had a 0-level list, that is, no list at all (or at best, one symbol, corresponding to the finite-state automaton itself). The second had a 1-level list, or pushdown stack, to store, temporarily, symbols denoting recognized substrings. The third had a 2-level list, to store, temporarily, arbitrarily many features of each recognized substring, in a way that made them readily available at remote stages in the recognition (or generation) process.

Each of these automata provides exactly the sort of properties one would want a perceptive automaton to have, if it lived in a one-dimensional world. The first recognized a finite number of primitive objects, without appealing to any significant internal structure. The second can, in addition perceive structure, in the sense that it can take previously recognized objects and their relationships (in one dimension, that of adjacency) and see that together they form a familiar object, that is, one for which there is a corresponding recognition state, or symbol, or description. We referred to this as combination - we could equally well associate the notion of articulation with this automaton.

The third can, in addition, perceive associations between remote objects; we considered the association of noun phrases in "respectively" sentences, and of various components in sentences with conjunctions.

Secondly, we formalized hitherto informal notions of identity, and showed how to combine these with indexed grammars to provide a sufficiently firm foundation to set up a translation algorithm which could be demonstrated to work with grammars at least as complex as indexed ones.

Thirdly, we claimed that the primitives of any theory of communication were representations, and that the fundamental use of grammars, in practice, was to enable corresponding representations in different languages to be derived from given representations. We demonstrated at the outset (3.1) that this was true of structural descriptions, which were simply translations, in a structural description language, of representations in some source language. In doing so, we assigned a weaker role to the notion of structural description than that currently popular with some linguists; we used it as no more than an aid to visualizing generative processes. In addition, we suggested how appropriate structural description grammars that might generate the sort of surface structures sought by transformationalists could be readily derived from phrase-structure rules that included the generative identity component appropriate to their use in some practical translation system (3.8, on immediate constituents).

Fourthly, we showed how translation theory dealt with problems inadequately catered for by transformational theory, although we acknowledged the extent to which transformational theory was, for generating the sentences of a language, a good improvement over a simple context-free approach, in that it invoked an asymmetric translation-like theory to isolate and reduce paths in surface structures to manageable lengths. An inherent fault was its lack of a mechanism to disentangle crossed paths, and a practical fault was its asymmetry, or failure to provide explicitly the surface-structure grammar which was used successfully in the MITRE program (Zwicky, 1965).

Chapter 4. Practical Recognition

4.1 Introduction

In the previous chapter, we assumed that, given an automaton programmed with a grammar, and a string in the language of that grammar, we could either make that automaton generate that string or reverse the direction of time, i.e., run the machine backwards, and recognize that string. This was a convenient assumption to make, since it enabled us to examine the problem of translation independently of that of practical recognition. In doing so, we were able to make translation a more exact science than before, although perhaps not so exact that its mathematical properties, and time and memory considerations, could be immediately determined.

We now consider practical recognition, that is, the art of making a deterministic automaton construct theories about the operations of a conceptual non-deterministic automaton. That it is still an art is suggested by the uncertainty (Aho, Hopcroft, Ullman, 1968, p.206) about the least upper bound on time for CF recognition. Currently, the best upper bound is n^3, that is, there is no effective procedure known for performing recognition with an arbitrary CF grammar, in time T(n) for strings of length n, such that $\lim_{n \to \infty} T(n)/n^3 = 0$. The procedure for which $\lim_{n \to \infty} T(n)/n^3$ is bounded above is Younger's algorithm.

Recognition is, strictly, the process of determining membership, that is, deciding whether a given sentence belongs to a given language. A yes-no answer to the membership question for a sentence tells us nothing about why the sentence is a member of a certain set. By practical recognition, we mean more than simple recognition; we mean the determination of sufficient theories about the possible top-down generation of a given string by a given grammar (used by the source automaton) to enable the target automaton to function in the manner described for the translation process.

The form of such a theory is difficult to restrict. The conventional approach is to construct a hypothetical record of a non-deterministic generatio of a string by a pushdown automaton and call this a structural description. For example, using the gramm

$S \rightarrow AB$ (a)

$A \rightarrow x$ (b)

$B \rightarrow C$ (c)

$C \rightarrow y$ (d)

to generate ab, we might theorize

Call S object 1.

Using rule (a) replace object 1 by objects 2 and 3.

Using rule (b) replace object 2 by x.

Using rule (c) replace object 3 by object 4.

Using rule (d) replace object 4 by y.

If we now refer to each step by referring to the identity of the rewritten object (in a computer, one usually refers to something by supplying its address in memory), then this description can be abbreviated to

1: a, 2, 3

2: b, x

3: c, 4

4: d, y

Provided no two rules of the grammar are identical (and this is, trivially, always the case in practice) it is sufficient to supply only the symbol rewritten rather than the identity of the rule, since the latter can always be reconstructed. Thus:

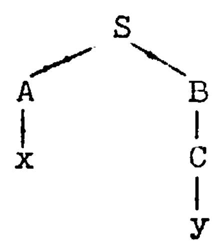
1: S, 2, 3

2: A, x

3: B, 4

4: C, y

which is as close to a computer-memory representation of the structure



as we need go here. To rediscover that step 1 is a, 2, 3, we note that 2 is A and 3 is B, whence the rule must be S → AB, which we can determine (by searching the grammar) to be rule (a).

That such a record can function as a program for a machine performing top-down generation is clear from its first formulation, above. As such, it is a very good theory of such a process, since, given the identification of an object, one simply looks at the appropriate location(s) in memory to determine which rule was chosen to rewrite that object. The target automaton, in using this theory, then chooses the corresponding rule in the target grammar.

However, this is not the only possible form for a theory. It is possible to do away altogether with this form of structural description, as we shall see in discussing the following algorithm.

4.2 Younger's Algorithm

This algorithm is described in detail by Younger (1967). We give here a very brief description.

The only rules dealt with are of the form

$$A \to BC$$

$$A \to B$$

or $$A \to a$$

As noted earlier, it is easy to restrict any CF grammar to such a form. Younger also omits the second rule, but as the resulting grammar can be quite unwieldy, this was not done here.

It should be clear from section 3.1 that such a restriction in no way prevents us from producing the same structural descriptions with this restricted form of grammar as those expected of a more elaborate grammar. Translation systems with rules of the form

$A \rightarrow BCD \qquad a \rightarrow [_A bcd]$

can be changed to

$A \rightarrow BX \qquad a \rightarrow [_A bx]$

$X \rightarrow CD \qquad x \rightarrow cd$

Given a string of terminals, it is possible to ask, Is the substring, of length j, starting with the ith terminal, an X , where X is some non-terminal category? Such a question is a boolean function of 3 variables (i, j, X); as such, a convenient data-base for answering such questions is a three-dimensional boolean matrix. This in fact is the data-base used in Younger's algorithm.

The answer to g(i, j, X) is yes if and only if one or more of the following is satisfied:

(i) (j = 1) <u>and</u> (X→a)∈P <u>and</u> the ith terminal is a.

(ii) ∃k,Y,Z, s.t. g(i, k, Y) <u>and</u> g(i+k, j-k, Z) <u>and</u> (X→YZ)∈P <u>and</u> $1 \leq k < j$

(iii) g(i, j, Y) <u>and</u> (X→Y)∈P

where P is the set of production rules for the grammar.

Each of these conditions is related to one of the three classes of rules permitted. The relationship should be transparent.

Younger's algorithm may be expressed in an ALGOL-like notation:

```
for each level j ≤ length (of input string)
    for each step k < j
        for each rule X → YZ
            for each position i ≤ length-j+1
                g(i, j, X) = g(i, j, X) or
                (g(i, k, Y) and g(i+k, j-k, Z))
    for each rule X → Y
        for each position i ≤ length-j+1
            g(i, j, X) = g(i, j, X) or g(i, j, Y)
```

(Indentation of any text implies begin end brackets around it. All for loops start from 1.)

This algorithm assumes that a rule A → B will always precede, say, C → A. For if not, and a B was discovered at position i, level j, then g(i, j, C) would be set to the value of g(i, j, A) before the latter had been set to the value of g(i, j, B), assuming the first two had previously been 0. This ordering of rules can always be arranged except when there are rules implying a cycle, e.g., A → B, B → C, C → A. This does not seem to be a serious restriction for an English grammar, although it could arise in a programming-language translation system, e.g.,

INT EXP → REAL EXP   i → FIX (r)

REAL EXP → INT EXP   r → FLOAT (i)

where we might be compiling from FORTRAN into LISP, and want to allow mixed-mode expressions. The ambiguity implied would presumably be ignored by the compiler.

In deciding the structure of the matrix with respect to the word-oriented structure of memory, it is convenient to choose i (that is, position) as the coordinate that varies within a word. In fact, if the ith bit of the word w(j, X) is g(i, j, X), then we may rewrite the algorithm

```
for each level j ≤ length (of input)
    for each step k < j
        for each rule X → YZ
            w(j, X) = w(j,X) or
            (w(k,Y) and w(j-k, Z)↑k)
    for each rule X → Y
            w(j, X) = w(j, X) or w(j, Y)
```

where $\uparrow k$ means "shifted left k bits".

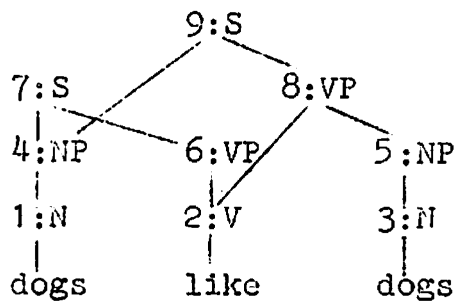
The word-length of the PDP-8 is 12 bits, and since the longest of Lewis Carroll's syllogisms is 23 words, the vector w will clearly be more than 1 word in some cases. Thus multiple-length shifting, and logical, operations are involved in both senses of the word. As the level j increases, each vector w (j, X) decreases, and an obvious economy can be, and is, effected by allowing variable-multiple-length operations.

The grammar

S → NP VP
NP → N
VP → V NP
VP → V
N → dogs
V → like

will generate "dogs like dogs". The corresponding matrix will be

level	1			2		3	j
position	1	2	3	1	2	1	i
S				7		9	
NP	4		5				
VP		6			8		
N	1		3				
V		2					



The structure diagram should help make the matrix clearer. An integer entry in the sketch of the matrix denotes a 1, and a blank denotes 0. The integer itself only indicates the order the bits appeared (except that 4 and 5 appeared simultaneously from 1 and 3), and is not part of the actual matrix.

We will discuss the details of the implementation further in the next chapter. Here we are mainly concerned with the principles.

It is not clear from the matrix alone in what sense we have produced theories about the operation of a non-deterministic automaton that generates strings top-down, that is, starting with the symbol S and rewriting symbols. There is certainly no structure, in the sense that there are no pointers from each bit in the matrix to those bits that were responsible for its presence. However, there is sufficient information to permit an automaton to operate deterministically to generate a copy of the input string.

Given a position i, a level j and a category X, as the coordinates of a bit in the matrix, it is not difficult to construct theories of which rules might have produced this bit. The following will suffice:

```
for each rule r:  X → YZ
    for each k < j
        if g(i, k, Y) and g(i+k, j-k, Z) then
                return r
for each rule r:  X → Y
        if g(i, j, Y) then return r
```

It does not take long to construct a theory about a bit, since usually only one or two rules starting with X are involved, and since the majority of nodes in a structural description of a string refer to short strings, j can be expected to be reasonably small.

The translation algorithm demands a rule in exchange for an identifier. Thus, the only thing needed now is a means of converting an identifier into (i, j, X) coordinates in the matrix. This is discussed in the next chapter, as are details of the actual implementation.

4.3 Recognition and Ambiguity

We mentioned earlier that appeals to the finer details of lexical and other agreements in the translation of sentences were not particularly interesting. If ignored, one can happily translate "He do not like eating hydrogen." However, sentences such as "Flying planes is dangerous" (Chomsky, 1965, p.21) are unambiguous only if an appeal to agreement in number is made; otherwise we could translate it as if it were "Flying planes are dangerous", which has translations in some other languages quite different from the correct translation of the first sentence.

Here we have conflicting plans of attack, whether to ignore or include such checks, and if include, how many.

The goal we should set is not some criterion for selecting the right number of checks, since it is clear that sometimes they are a hindrance, sometimes a help. Rather, we should aim at producing an unambiguous translation. A minimum of grammatical apparatus should be used to produce possible translations, while ensuring that the correct translation will be among them. (The grammar used in the program described in the next chapter would approximate to such a minimum.) Then the resulting translations should be compared, and the essentially different ones selected. Finally, further criteria invoking as much grammatical detail as necessary are used to eliminate candidates, until one remains, or the grammar is exhausted.

In theoretical applications of language processing, it is very convenient to be able to produce large numbers of translations where they arise, label them as ambiguities, and forget about the problem. In practice, most machines and humans function inefficiently when they attempt to process a set of messages of which one is known to be correct. For example, at one stage in the development of mechanical translation from Russian into English, if a word had several meanings, <u>all</u> were given in the translation. The unreadability of the result was seized on by critics as an indictment of mechanical translation.

An alternative approach to the problem is suggested by indexed grammars. Their use facilitates numerical estimates, for a given theory for the source automaton, of the plausibility of that theory.

In recognizing sentences with indexed grammars, it is quite easy to take over existing CF recognition algorithms, provided a few minor restrictions (analogous to the restriction on cycles A → B, B → C, C → A described for Younger's matrix) are imposed on the grammar. Let us assume some algorithm which theorizes about the CF rule A → BC by noting that, somewhere, it has a B next to a C, and hence it has an A. The extension is as follows.

Consider A, B, and C as denoting, not symbols, but lists of indices. The rule becomes, for indexed grammars, A → B + C.

Given two adjacent nodes (which are therefore lists), such that the first is of the form (B, X) and the second (C, Y), where all symbols denote lists of indices, and furthermore X and Y match (although one may be longer), then we have a node which is the list (A, Z) where Z is the longer of X and Y.

Rules of the form A → B are dealt with even more simply. If we have a (B, X) then we have an (A, X).

Rules of the form A → B + C + D can be dealt with by reducing them to two rules, just as with CF grammars.

As it stands, such an algorithm, when taking into account all sorts of lexical agreements, may repeatedly fail to find translations of mildly ungrammatical, but otherwise useful, sentences. The modification is to relax the condition given above, that the lists X and Y match, and instead to use the extent to which they do not match as a measure of implausibility of the corresponding theory. This can be made even more effective by taking into account the individual plausibilities of the existence of

B and C. If both are, independently, implausible, then their combination should be very implausible.

The most plausible way of deriving a sentence is then taken to be the best theory of how the sentence was generated. The advantage of this method over the previous one suggested is that a single analysis suffices. The disadvantage is that such an analysis may take much longer than the average analysis by the other method, in which only occasional appeals to lexicographic and similar details may be required.

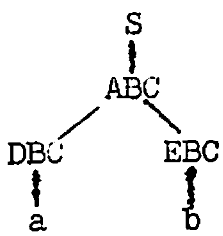
4.4 Details of Indexed Recognition

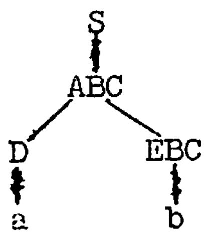
Earlier we noted that, in generating sentences with indexed grammars, many nodes might carry large numbers of indices that are never used, but are simply dropped when the nodes are rewritten as terminals. This seemed ungainly. However, when recognizing strings in the way described above, we noted that X and Y did not need to be of the same length when matching them. Thus the surplus indices need never be considered during recognition.

Consider the grammar

$S \rightarrow ABC \quad A \rightarrow D + E \quad D \rightarrow a \quad EBC \rightarrow b.$

In generating ab we have the diagram on the left,





and in recognizing, the one on the right. In comparing D and EBC, we found that the D and the E could be used with the second rule to make an A, The two remaining lists that need to be compared are the null list and the list BC. Since the latter may be considered as th null list followed by the list BC, the two lists match as far as the shorter one goes. The list BC, being the longer, is then attached to A.

The efficiency of such a recognition algorithm is difficult to estimate. There is as yet no evidence to suggest that this or any other algorithm would be usefully efficient. However, if one attempts to visualize the steps taken by such an algorithm, one can make a rough estimate.

A reasonable assumption is that the number of nodes in a typical structure diagram for an English sentence is bounded above by a linear function of the length of the sentence, say kn. The same assumption, made of the length of nodes, would also be reasonable. Since every pair of nodes need only be compared once, an upper bound on the number of comparisons is Kn^2 $(K = k^2)$. Each comparison takes a time proportional to the length of the nodes, which is therefore bounded above by, say, cn. Hence, the total time for recognition must be bounded above by Cn^3. $(C = cK)$.

But this is the upper bound on recognition with Younger's algorithm for CF recognition. So our result seems unreasonable, by comparison, since CF recognition does not involve checking a list of indices at each comparison.

In practice, as is described in the next chapter, Younger's algorithm takes a time proportional to n^2, in fact, .007 n^2 seconds in the program described. The assumption about the number of nodes seems not at all unreasonable. (It should be noted that a modification to Younger's algorithm, in which zero vectors are ignored, allows the number of nodes in the diagram to affect the timing.)

Thus the outlook for indexed grammars is reasonably bright. Whether the extra factor of n (assuming that this is the case) justifies their use is difficult to say without further experience. By comparison with CS grammars, however, they seem much easier to handle , although there seem to be no figures available on the efficiency of CS recognizers.

Chapter 5. The Program

The program itself is of interest mainly because it is well described by the translation theory, despite the fact that it came before the theory. However, it may also be of interest because it uses what is nearly the smallest cheapest general-purpose computer on the market, or because it uses a closed-class dictionary, or because it shows that Younger's algorithm is of more than theoretical interest, or because it demonstrates what to do about ambiguity, or simply because it is fun to see a computer doing useful things with English sentences. On the other hand, descriptions of list-processors and buffered I/O routines abound and presumably are of no interest. Thus we shall describe the essential features of the interesting parts of the program, as briefly as is reasonable.

5.1 Translation Theory

As one might expect from the description in chapter 1, conjunctive normal form formulae (CNF sentences) are 2-level lists, such that the list ((A, B), (C, D, E)) means (AvB).(CvDvE), and so on. It is difficult to invent a grammar for this to enable them to be produced by a single translation, so we decomposed the translation, from English to reverse Polish (CF → CF) and then to CNF, by treating the reverse Polish sentences as programs for computers with pushdown stores. While the second half of this process is a translation, it is not strictly a phrase-structure translation as described in the bulk of the theory. Since the running of such programs is easy to grasp, we give only an example of such a translation.

"Big dogs are bad" becomes

(Big)(dogs). - (bad)v

To run the second sentence as a program, we read it as

Pushdown big

Pushdown dogs

and [result is ((big), (dogs))]

not [result is ((-big, -dogs))]

Pushdown bad

or [result is ((-big, -dogs, bad))]

If we started with an empty pushdown store, we should now have a CNF formula in it.

Clearly the only programming needed for each logical operation is enough to take one or two CNF formulae and produce a single CNF formula. This is quite trivial and requires no elaboration here. It is worth noting that each step of the program can be, and is, executed and discarded as soon as it is generated by the target automaton in the first stage of translation. This saves a little space, although no time.

The interesting stage is the first, English to reverse Polish, since this is CF→ CF, and should be describable by the theory. The simplest grammar for the particular version of reverse Polish used here is

F → FFv

F → FF.

F → F-

F → T

F → string

where F is a formula and T is <u>true</u>, which is for "thing" and "one", etc., where it is obvious that they are not as interesting as "dog" and "baby".

The system of grammars simply associates CF rules of English with one of the above rules, or an extended rule, such as

F → FF-v

which, if added to the grammar, does no harm to the language, since ambiguity in target grammars is irrelevant.

The English grammar currently has 80 rules of the form A → BC, 76 of the form A → B, a closed-class dictionary of 146 words (there are 720 different words in Carroll s syllogisms) and a suffix dictionary with 19 entries. A complete description of the grammar is too much to undertake. However, certain rules are of interest, to demonstrate how it works.

A simple sentence is, Babies are illogical. Rather than describe all the rules that would in practice be applied to this sentence, we shall use a smaller translation system.

S → NP VP	F → F - F v
A A.1 A.2	A A.1 O A.2 O
NP → SS	F → string
A A.1	A O
VP → BE AJ	F → F
A A.1 A.2	A A.2

The grammar is written this way to show its similarity to the translation theoretic notation we used before. However, it is easy to abbreviate it. The identity component of the left side can always be assumed to be A A.1 (A.2), whence only the rules as they are conventionally given need appear in storing them in tables. The right-hand rules can be abbreviated to 1-2v, 0, and 2 repectively, without losing any information, since F is the only non-terminal symbol involved. For no special reason, L and R (left and right) were chosen instead of 1 and 2. Where <u>true</u> appeared, it was abbreviated to N (null). 0 was omitted.

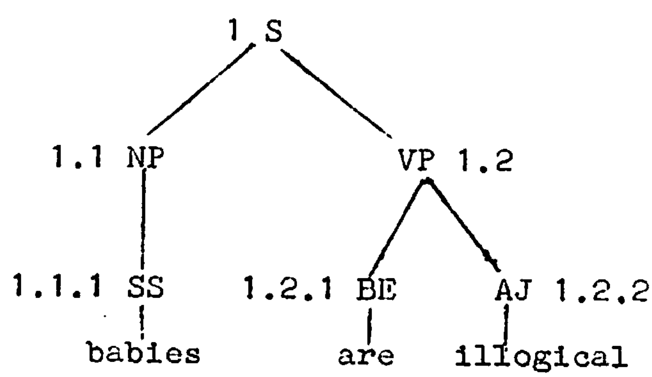
The sample grammar becomes

a   S → NP VP: L-Rv

b   NP → SS:

c   VP → BE AJ:R

The structure



should make identities and rules clearer. This can be readily used by a human for answering the translation automaton's questions.

Since every rule in the target grammar rewrites F, the translation automaton will always have to consult the source automaton's theory at each step.

Starting with F, with identity 1, we apply rules to derive a sentence:

Identity of rewritten symbol	Rule	Result
1	a	F - F v
		1.1 0 1.2 0
1.1	b	babies - F v
		0 0 1.2 0
1.2	c	babies - F v
		0 0 1.2.2 0

At this point, the theory about 1.2.2 must be that some rule of the form $A \rightarrow a$ is involved. For simplicity, all such rules were made to correspond to $F \rightarrow$ string. Thus:

1.2.2	d	babies - illogical v
		0 0 0 0

The second stage of translation yields ((-babies, illogical)), that is, a thing is either not a baby or it is illogical, which is equivalent to baby → illogical.

Identity is referred to explicitly in this discussion, but the program can keep track of identity implicitly. For example, 1.2 can be replaced by the (i, j, X) coordinates of the bit in the Younger matrix of which 1.2 is the identity. Since the target automaton never runs without consulting the source, it need not be concerned about losing track of identity in the manner described for the respectively problem.

A pushdown store is used for holding all but the leftmost symbol of the derivation. At the start, the store is empty. The coordinates (1, 3, S), corresponding to identity 1, are put on the store. Then each symbol at the top of the store is either rewritten if it is an F (that is, a coordinate), or output, until the store is empty. After the first step, the store holds

F(1, 1, NP)	(1.1)
-	(0)
F(2, 2, VP)	(1.2)
v	(0)

The remainder of the steps should be clear.

A more complex example is "No one can remember the battle of Waterloo unless he is very old". As the solution to this uses over 40 rules, and its structure diagram, in the Chomskian sense, has nearly 50 nodes, we refrain from sketching it or enumerating the rules.

Two of the rules are of interest.

S → PC AC: LR-v

AC → CN PX: R-

PC is the principal clause, up to "Waterloo", while AC is the adverbial clause. CN is a "conditional negative", in this case "unless". PX is a general symbol for the class of things that follow words like "unless", "if", "when", etc., which include past and present participial phrases, etc..

If written as one rule, this becomes

S → PC CN PX: 13v

where we have reverted momentarily to the numerical notation, since L and R only account for 1 and 2. Thus it can be seen that the sentence is interpreted as "Either no one can remember the battle of Waterloo or he is very old".

In so rendering it, we have assumed that each noun phrase functioning as a subject is quantified by the same variable. Thus "he" and "no one" refer to the same person, when the persons are enumerated, as implied by universal quantification. That is, "for all x, either x can not remember the battle of Waterloo or x is very old."

To achieve this, once the assumption has been accepted, rules are set up to block the copying of strings such as "one" and "he". This is done with rules such as

NP → PN: N

meaning that if a nounphrase is a pronoun, then its translation is <u>true</u> (or null).

To see that this works in practice, consider

S → NP VP: L-Rv

To make one rule out of the last two, we have

S → PN VP: N-Rv

The right side can be seen to be equivalent to R.

Another example is

NP → AJ NP: LR.

The combination rule would be

NP → AJ PN: LN.

The right side is simply L.

The final result for the above sentence is

((-remember the battle of Waterloo, very old)).

The grammar has been arranged to ignore auxiliaries, as it makes the treatment of negatives simpler when "not" occurs between an auxiliary and a verb. For Carroll's syllogisms, it makes no difference to ignore the auxiliary, but a more sophisticated system for making distinctions between "can" and "do" would insist on keeping the auxiliary. It is trivial to alter the grammar to include the auxiliary.

5.2 Relation to the Contingency Table

In the table of actions to be taken by the target automaton, given in section 3.9, we listed six contingencies. We shall discuss their relationship to the program.

If g(1, length, S) is 0, after executing Younger's algorithm, then no theory about why the whole string is a sentence is possible. Thus, evasive action is taken. In the program this amounted to proceeding to the next sentence. This is the only point in the processing where there is a possibility of no theories, which is a characteristic of bottom-up recognizers.

In using the matrix to discover theories, it is clear that further theories about the same bit can be left for later discovery. The procedure adopted was to find a theory, and then to see if any more theories were possible. If so, the bit's coordinates were put on a list of such sources of ambiguity, but no immediate attempt was made to see how many theories there were. At the completion of a translation, the list of other theories was examined and another translation was commenced if there were any more combinations of theories that might give a different translation.

With this program, it is not clear whether one can say one is using the right hand column of the contingency table. Certainly, the first line is irrelevant, except in some abstract sense. Since the possibility of further theories is considered at the time of finding the first theory, the second line is also irrelevant. On the other hand, since a complete search is not carried out for all theories about a bit, at the one time, the third row of the left hand column seems to be irrelevant. Thus in practice we may say that this particular program uses the first two rows of the left-hand column and the third of the right.

5.3 The Closed-Class Dictionary

Bobrow (1963) reports two such dictionaries used very successfuly (Klein, Simmons, 1963; Resnikoff, Dolby, 1963). A more recent example is given by Thorne (1967), also quoted as being successful.

The essence of a closed-class dictionary is that a very large proportion of different words belong to a very small number of syntactic classes, namely, nouns, verbs and adjectives. Also, as the vocabulary of a language is increased, by technological developments, for example, new words almost invariably fall into one of these classes. For the latter reason, the remaining classes are called closed, since they seem virtually immune to being increased. For the former reason, closed-class dictionaries are more economical than complete ones. As noted earlier, only 146 words were required for the successful recognition of practically all of Carroll's syllogisms, which had 720 different words. With a complete dictionary, nearly 3000 2-character words of memory would have been required, which made the other a necessity.

At first sight, it would appear to be a serious matter if one cannot tell of an unrecognized word whether it is a noun, verb or adjective. However, most nouns can be used as adjectives, and many as verbs, too. Thus, not as much is lost as one might expect. This in part could account for the glowing reports of success with such dictionaries.

Our own dictionary has been successful far beyond our expectations. There is the occasional instance of a report by the computer of something trying to cupboard something else, but ambiguities of this nature were far more rare than the structural ambiguities for which natural languages are notorious. When they did happen, the corresponding analysis was usually completely different from the correct one.

Augmenting the closed-class dictionary is a suffix dictionary of 19 entries, such as ible, ught, ing, s, ed, ould, etc. Each of these is associated with a non-terminal symbol; ible is an AJ (adjective), s is an SS (reserved especially for s), and so on. In the example earlier, babies are illogical , the rule NP → SS was used, indicating that "babies" had only a terminal s as a distinguishing feature, and hence was recognized as an SS in the first instance.

If a word submitted to the dictionary routine cannot be found in the closed-class dictionary, its ending is compared with possible suffices. If no suffix matches, the word is assigned the category U for unknown.

The grammar includes rules

$N \rightarrow U$

$AJ \rightarrow U$

$VT \rightarrow U$ (Transitive verb)

$VI \rightarrow U$ (Intransitive verb)

and it is this delightfully simple mechanism that allows the program to select the correct category with almost supernatural consistency.

5.4 Implementation of Younger's Algorithm

Sentences up to 64 words in length have been recognized with this program, during some preliminary timing tests. With 100 non-terminals, the corresponding size of the matrix is 200,000 bits. Clearly, something has been done to the matrix to reduce its size.

In the example of a matrix analysis in the last chapter, 8 out of 15 vectors were zero. Thus it is reasonable to arrange that storage space be allocated only for non-zero vectors.

This is done by maintaining a "dope vector" for each level. An element of a dope vector comprises a pointer to the block of Younger-matrix vectors for the corresponding level j, and 100 bits corresponding to the 100 non-terminals, each to indicate the presence or absence of the vector w(j, X) for category X corresponding to that bit. The dope vector has as many elements as there are levels, and hence words in the sentences. A 23-word sentence would consume nearly 200 words for the dope vector alone. But the size of the matrix is reduced drastically, to almost exactly the size of the dope vector, over a large range of sentence lengths.

A benefit from the size reduction is an increase in speed. If most vectors are zero, then the time spent in shifting and and-ing two vectors both non-zero will be negligible compared to the time manipulating pairs of vectors one or both of which are zero.

The timing of the algorithm was found to be virtually independent of any factor except the length. Sentences with no analyses were processed just as fast as ones with over 100 possible analyses. The timing was estimated to be .007 n^2 seconds; on a log-log graph, points plotted were all almost exactly on a line of gradient 2. n is the number of words in the sentence.

An improvement in timing by a factor of up to 5 can be had simply by using the fact that for over four-fifths of the rules of the form A → BC, either B or C could not be of length more than 1 or 2, and hence the main loop of the algorithm can be cut short for those rules. This was not taken advantage of in this implementation, since the idea ocurred after it had become apparent that the program was already too fast for the teletype to keep up.

5.5 Ambiguity in Practice

In a system that does more than simply parse sentences, it is possible to rely, to an extent, on successful operation as a criterion for selecting correct translations. In this case, the generation of a conclusion from several premises, such that the conclusion contained only two or three terms, would indicate that the appropriate translations might have been used.

A sentence may produce, typically, from one to three CNF translations. A suggested plan is to use each combination of translations to produce conclusions, and to select the best. In practice, this would be expected to work whenever there was a correct translation among those of each premise, since it appears unlikely that short conclusions could be drawn from bad translations.

5.6 Outline of Program

We sketch, without fine detail, the order in which translation proceeds.

1. New sentence: Perform dictionary analysis of each word in a sentence premise. Store results in an array.
2. Start the matrix routines. Set up bits to correspond to the results of step 1. Execute Younger's algorithm.
3. If g(1, length, S) = 0, go to 1 (no analysis).
4. Perform translation as described.
5. Print result.
6. If all theories (ambiguities) have been processed, go to 1, else go to 4.

5.7 Statistics

Storage

Statistics that might be of interest are:

Size of Program: 1700 words

This is broken down approximately into:

String Processor:	100
Interrupt Handler for buffered I/O	90
Dictionary routines	80
Younger s algorithm	400
List Processor	128
Theory Constructor	300
CNF logic	160
Translator	150

and miscellaneous routines and data.

Major working areas are:

I/O and other character buffers	256
Translation system grammars (156 rule pairs)	550
Dictionary	500
Younger Matrix - maximum	400
List-processing area	200

Timing

This is directly proportional to the number of rules of the form $A \rightarrow BC$. With 80 such rules, a sentence of n words takes .007 n^2 seconds to be recognized. A single translation takes from 0.2 to 0.3 seconds for sentences of from 10 to 15 words. The syllogisms are read at 10 characters per second from paper tape, or may be entered manually via the keyboard. The results are printed at 10 characters per second.

Generality

There is no need to use the English grammar. One for any other language, either natural or formal, provided it is context-free, will produce the same results. A small grammar for arbitrary logical expressions could readily be constructed, so that such expressions could be translated into CNF. A status table for punctuation symbols enables any character to have the status of a letter so that logical operators (brackets, etc.) may be treated as words, and hence may be included in the dictionary.

5.8 Envisaged Extensions

In order to extend this program to a complete syllogism solver, there are four major sections: semantics, syllogistic inference, evaluation of the best solution, and translation back into English.

The semantics section is responsible for resolving questions of synonymy. A more powerful syntactic analyser would result in requiring less semantic analysis. In this instance, semantic analysis consists of comparing strings from different premises to see if they are synonymous, contradictory or independent. The simplest such analyser would simply compare the two strings, character by character, to see whether they are identical. The next step is to compare them word by word, removing affices from each word beforehand. If an odd number of negative affices (e.g., un-, in-, etc.) are removed, and all the remaining stems match, they are contradictory. If "not" is removed entirely, and counted as a negative affix, this enables auxiliaries to be included in the string matching, thus enabling a distinction to be made between "can" and "do."

Syntactic problems that cannot be resolved by the semantic analyzer are those related to passive-active transformations, and to segments, or terms, that are the object of a verb, such as "I dislike coloured flowers". Most of the difficult premises are affected by the latter consideration.

The syllogistic inference section is quite trivial. The discussion in sections 1.3 and 1.4 should make this clear.

Evaluation of the best solution can be done as a function of either the number of different terms, or the number of disjuncts, in the conclusions derived from the inference section. In the latter case, a normal conclusion has one disjunct, for a well-formed sorites, and in the former it has two terms, and possibly one or two universe terms as well. The difficulty in distinguishing universe terms from any others suggests that the number of disjuncts be used as a criterion.

Translation back into English is best done by appealing to style. CNF is not a CF language, and does not lend itself to the same translation process used for going from English to CNF. In appealing to style, it is essential to know the original syntactic roles of each term. The construction of an English sentence must be on an appeal to style, since there are so many ways of expressing a CNF formula in English.

5.9 Conclusion

The program was quite successful, considering that its grammatical capabilities were relatively unsophisticated. The theory evolved was also successful, possibly as a result of a happy combination of insights arising out of the program and ideas from the literature.

Computational linguistics would appear to be at a stage where it will benefit equally from doing and thinking. Without the doing, there will be no examples or counter-examples of what can or can't be done. Without the thinking, the doers may not know when they are attempting the impossible or the inefficient. It does not cost much money to think, but a prevailing attitude amongst the doers is that it can't be done for less than a hundred thousand dollars, and will probably cost a million. In demonstrating that equipment that can be bought for eight thousand dollars can be used in a non-trivial natural language application, it is hoped that this attitude has been, at least, challenged.

Appendix

Listed below are the results of translation of the first twelve syllogisms. The notation should be transparent; all formulae are in CNF.

All but the second and the seventh syllogisms can be seen to be solvable, in the sense that for each premise, there is at least one correct translation.

The third premise of the seventh syllogism suffers at the hands of the grammar; "consists" was incorrectly given as behaving like "be".

The second syllogism is atypically pathological. Although it is quite easy to arrange the grammar to process sentences starting "x find(s)", it was not done for this demonstration. The "only" problem was not attempted; sentences of the form "x are the only y" mean "all y are x". An extension to the grammar of the order of ten rules would be required to effect repairs.

1

BABIES ARE ILLOGICAL

A BABIES
B ILLOGICAL

(-A, B)

NOBODY IS DESPISED WHO CAN MANAGE A CROCODILE

A DESPISED
B MANAGE A CROCODILE
(-A,-B)

ILLOGICAL PERSONS ARE DESPISED

A ILLOGICAL
B PERSONS
C DESPISED

(-A,-B, C)

2

MY SAUCEPANS ARE THE ONLY THINGS I HAVE THAT ARE MADE OF TIN

A MY
B SAUCEPANS
C ONLY
D I HAVE
E MADE OF TIN

(-A,-B, C,-E)&(-A,-B, D,-E)

A MY
B SAUCEPANS
C ONLY
D I HAVE

E MADE OF TIN

(-A,-B, C)&(-A,-B, D)&(-A,-B, E)

I FIND ALL YOUR PRESENTS VERY USEFUL

NONE OF MY SAUCEPANS ARE OF THE SLIGHTEST USE

A MY
B SAUCEPANS
C OF THE SLIGHTEST USE

(-A,-B,-C)

A MY
B SAUCEPANS
C OF THE SLIGHTEST USE

(-A,-B,-C)

3

NO POTATOES OF MINE, THAT ARE NEW,
HAVE BEEN BOILED

A POTATOES
B MINE,
C NEW,
D BOILED

(-A,-B,-C,-D)

A POTATOES
B MINE,
C NEW,
D BOILED

(-A,-B,-C,-D)

ALL MY POTATOES IN THIS DISH ARE FIT
TO EAT

A MY
B POTATOES
C IN THIS DISH
D ARE FIT TO EAT

(-A,-B,-C, D)

A MY
B POTATOES
C IN THIS DISH
D FIT TO EAT

(-A,-B,-C, D)

NO UNBOILED POTATOES OF MINE ARE FIT
TO EAT

A UNBOILED
B POTATOES
C MINE
D ARE FIT TO EAT

(-A,-B,-C,-D)

A UNBOILED
B POTATOES
C MINE
D FIT TO EAT

(-A,-B,-C,-D)

4

THERE ARE NO JEWS IN THE KITCHEN

A JEWS

B IN THE KITCHEN
(-A,-B)

NO GENTILES SAY "SHPOONJ"

A GENTILES
B SAY "SHPOONJ"

(-A,-B)

MY SERVANTS ARE ALL IN THE KITCHEN

A MY
B SERVANTS
C ARE ALL IN THE KITCHEN
(-A,-B, C)

A MY
B SERVANTS
C IN THE KITCHEN
(-A,-B, C)

A MY
B SERVANTS
C IN THE KITCHEN
(-A,-B, C)

5

NO DUCKS WALTZ

A DUCKS
B WALTZ
(-A,-B)

NO OFFICERS EVER DECLINE TO WALTZ

A OFFICERS
B WALTZ
(-A, B)

ALL MY POULTRY ARE DUCKS

A MY
B POULTRY
C DUCKS

(-A,-B, C)

6

EVERY ONE WHO IS SANE CAN DO LOGIC

A SANE
B DO LOGIC
(-A, B)

A SANE
B DO LOGIC
(-A, B)

NO LUNATICS ARE FIT TO SERVE ON A JURY

A LUNATICS
B ARE FIT TO SERVE ON A JURY
(-A,-B)

A LUNATICS
B ARE FIT TO SERVE ON A JURY
(-A,-B)

A LUNATICS
B FIT TO SERVE ON A JURY
(-A,-B)

NONE OF YOUR SONS CAN DO LOGIC

A YOUR
B SONS
C DO LOGIC
(-A,-B,-C)

A YOUR
B SONS
C DO LOGIC
(-A,-B,-C)

7

THERE ARE NO PENCILS OF MINE IN THIS BOX

A PENCILS
B MINE
C IN THIS BOX

(-A,-B,-C)

NO SUGAR-PLUMS OF MINE ARE CIGARS

A SUGAR-PLUMS
B MINE
C CIGARS

(-A,-B,-C)

THE WHOLE OF MY PROPERTY, THAT IS NOT IN THIS BOX, CONSISTS OF CIGARS

```
A MY
B PROPERTY,
C IN THIS BOX,
D OF CIGARS

(-A,-B, C, D)
```


```
A MY
B PROPERTY,
C IN THIS BOX,
D OF CIGARS

(-A,-B, C, D)
```


8

```
NO EXPERIENCED PERSON IS INCOMPETENT

A EXPERIENCED
B PERSON
C INCOMPETENT
(-A,-B,-C)
```


```
A EXPERIENCED
B PERSON
C INCOMPETENT
(-A,-B,-C)
```


```
JENKINS IS ALWAYS BLUNDERING

A JENKINS
B BLUNDERING

(-A, B)
```


```
A JENKINS
B BLUNDERING
```

(-A, B)

NO COMPETENT PERSON IS ALWAYS BLUNDERING

A COMPETENT
B PERSON
C BLUNDERING

(-A,-B,-C)

A COMPETENT
B PERSON
C BLUNDERING

(-A,-B,-C)

9

NO TERRIERS WANDER AMONG THE SIGNS
OF THE ZODIAC

A TERRIERS
B WANDER AMONG THE SIGNS OF THE ZODIAC
(-A,-B)

NOTHING, THAT DOES NOT WANDER AMONG
THE SIGNS OF THE ZODIAC, IS A COMET

A WANDER AMONG THE SIGNS OF THE ZODIAC,

B COMET
(A,-B)

A WANDER
B AMONG THE SIGNS OF THE ZODIAC,
C COMET
(A,-B,-C)

A ZODIAC,
B COMET
(-A,-B)

NOTHING BUT A TERRIER HAS A CURLY TAIL

A TERRIER
B HAS A CURLY TAIL
(A,-B)

NO ONE TAKES IN THE TIMES, UNLESS HE
IS WELL-EDUCATED

A TAKES IN THE TIMES,
B WELL-EDUCATED

(-A, B)

A TAKES IN THE TIMES, UNLESS HE IS
WELL-EDUCATED

(-A)

A TAKES IN THE TIMES, UNLESS HE IS
WELL-EDUCATED

(-A)

NO HEDGE-HOGS CAN READ

A HEDGE-HOGS
B READ
(-A,-B)

MY GARDENER IS WELL WORTH LISTENING
TO ON MILITARY SUBJECTS

A MY
B GARDENER
C IS WELL WORTH LISTENING TO ON MILITARY
SUBJECTS

(-A,-B, C)

A MY
B GARDENER
C WORTH LISTENING TO ON MILITARY SUBJECTS

(-A,-B, C)

NO ONE CAN REMEMBER THE BATTLE OF WATERLOO,
UNLESS HE IS VERY OLD

A REMEMBER THE BATTLE OF WATERLOO,

B VERY OLD

(-A, B)

A REMEMBER THE BATTLE OF WATERLOO,
UNLESS HE IS VERY OLD

(-A)

A CAN REMEMBER THE BATTLE OF WATERLOO,
UNLESS HE IS VERY OLD

(-A)

NOBODY IS REALLY WORTH LISTENING TO
ON MILITARY SUBJECTS, UNLESS HE CAN

GRAMMAR CURRENTLY
IN USE:

```
 S TA NG:R-
 S PC AC:LR-V
 S PC RC:LR-V
PC NQ PD:L-RV
PC NG PD:LR-
NR NR PJ:LR
NR NP OP:
NR NP PO:LR
NR TG JP:R
NR NP PS:LR
NR WL OP:R
NF NF PJ:LR
NF NF OP:R
WT SU AS:
NQ WT PS:R
NQ WT PD:R
NQ JM QN:
NQ AT NR:LR
PS NR VT:
NG AG NR:R
NG NH OP:R
AT AL AA:R
NP JP NP:LR
NP ED NP:LR
NP GR NP:LR
RC EX NQ:R-
RC WH PS:R
RC WH PD:R
AC CV PX:R
AC CN PX:R-
CV SO LA:
LA LG AS:
AP PR NQ:
OP OF NQ:R
PO OF PP:R
PD AN VP:R
PD AI VP:R-
PD PD AC:
PE AF PE:R
PE AK PE:R-
VP LY VP:
VP MV VX:
VP MG VX:R-
VF VF AP:
```

VF VI OP:
VF VF AB:
VF VF LY:
VF VT NQ:
VF HN PU:
VF DV OB:
VX TO VP:R
VB BN OB:R
VB BG OB:R-
VT GV NQ:
PT LY PT:
PT ED OP:
PT DD NQ:
PU PU FF:
PU WO IM:
IM GR NQ:
IM IM FF:
FF AS PU:
GR GG NQ:
OB TO VB:
OB JP OP:
HN HV AF:
AI AX NT:
AN AX AF:
BN BE AF:
BN HN BD:
BN BN BI:
BG BE NT:
MV MW NQ:
DV TN AB:
LY JM LY:
LY AD OC:
JP JM AJ:R
JP MJ VX:
MJ AF MJ:
TA TH BN:;
BN BE:
MV HV:
MV RE:
MW RE:
AF AL:
AF RY:
AP HR:
AP TH:
AB PV:
WH TT:
JM RY:
AJ MJ:
AJ U:

```
JP AJ:
QA AL:N
SP TE:N
SP PA:
AA SP:L
AA DM:
AT AA:L
AT IA:N
AT QA:L
PR TO:
PR AD:
PR AS:
PR PV:
DV CM:
VT HV:
VT RE:
VT  U:
VT DO:
VT SS:
VI SS:
VI CM:
VI  U:
 N SS:
 N  U:
AX DO:
AN AX:
HN HV:
FF AB:
FF PR:
FF AP:
PT ED:
PU PT:
IM GR:
PZ IM:
PZ AP:
PZ PU:
PX PZ:L
PX PC:L
VF VI:
VF VB:L
VP VF:L
PE VP:L
PD PE:L
PJ AC:L
PJ PZ:
PJ RC:L
NP NI:N
NP IM:
NP DM:N
```

```
NP  N:
NP ON:
NR NP:L
NR PN:N
NR QA:N
NR TG:N
NQ NR:L
NF NH:N
NG NF:L
OB PZ:
OB OP:
OB NQ:L
OB JP:
 S PC:L;!
PN I
IA A
BE AM
IA AN
AS AS
AD AT
BE BE
PV BY
PN HE
CV IF
BE IS
PN IT
PN ME
PA MY
PV ON
OF OF
AG NO
PV IN
SO SO
TO TO
DO DO
CJ AND
QA ANY
BE ARE
EX BUT
PR FOR
ED GOT
ED HAD
HV HAS
PA HIS
AX MAY
ED MET
WH WHO
AX CAN
PV OFF
```

```
PA OUR
AB OUT
TE THE
AL ALL
MJ FIT
NI ONE
NT NOT
JM TOO
PN YOU
AB AWAY
AB BACK
KN CALL
AX CANT
MV CARE
CM COME
AX DARE
DO DOES
PV DOWN
MJ EASY
MG FAIL
KN FIND
PR FROM
DV GETS
HR HERE
PR INTO
ED KEPT
GV LEND
LG LONG
RE LOVE
PR LIKE
AJ ONLY
ED MADE
PA YOUR
JM VERY
NH NONE
PP MINE
QN MUCH
PR NEAR
OC ONCE
PR OVER
TT THAT
HV HAVE
BD BEEN
ED PAID
ED SOLD
MJ SURE
SU SUCH
PN THEY
PN THEM
```

SP THIS
ED TOLD
TN TURN
AF WELL
AX WILL
PR WITH
WT WHAT
CV WHEN
AF EVER
PV ABOUT
PR AMONG
BI BEING
AJ CURLY
MG FAILS
ED GROWN
DV LOOKS
RE LOVES
AK NEVER
JM QUITE
AB STILL
N TABLE
SP THESE
DM THOSE
TH THERE
TN TURNS
PR UNDER
WH WHICH
CV WHILE
WL WHOLE
WO WORTH
IA EVERY
GV ALLOWS
AI CANNOT
U CIRCUS
VT DETEST
EX EXCEPT
GG GIVING
MV HAPPEN
MJ LIKELY
DD MARKED
RY REALLY
NI THINGS
CN UNLESS
NH NOBODY
AF ALWAYS
DD BRANDED
DD LABELED
NH NOTHING
MV OFFERED

```
MG DECLINE
AB UPWARDS
MJ WILLING
PR WITHOUT
ED WRITTEN
TG ANYTHING
BE CONSISTS
NI ARTICLES
WT WHATEVER
NR EVERYBODY
RE RECOMMEND
JM ABSOLUTELY
TG EVERYTHING
JM HOPELESSLY!
AJ ABLE
AJ IBLE
AJ ICAL
AJ LESS
 N NESS
AX OULD
AJ SOME
 N TION
ED UGHT
AJ FUL
AJ EST
GR ING
 N OUR
AJ OUS
ED ED
LY LY
PA 'S
SS S
 N "
```

Bibliography

Abbreviations:

CACM - Communications of the ACM

JACM - Journal of the ACM

IC - Information and Control

FJCC - Fall Joint Computing Conference

Aho, Alfred V., 1968, Indexed grammars, JACM 15, 4, 647, Oct.

Aho, Hopcroft, Ullman, 1968, Time and tape complexity of pushdown automaton languages, IC 13, 3, 186.

Bach, Emmon, 1966, An Introduction to Transformational Grammars, Holt, Rinehart and Winston, Inc., N.Y.

* Bobrow: D.G., 1969, Quoted from personal communication.

Brainerd, Walter, 1969, Tree generating regular systems, IC 14, 2, 217.

Chartres, B.A., and J.J. Florentin, 1968, A universal syntax-directed top-down analyzer, JACM 15, 3, 447, July.

Chomsky, Noam, 1957, Syntactic Structures, Mouton and Co., The Hague (6th pr. 1966).

______, 1959, On certain formal properties of grammar, IC 2, 2.

_____, 1964, A Transformational Approach to Syntax, in Fodor and Katz, p.211.

_____, 1965, Aspects of the Theory of Syntax, M.I.T. Press, Cambridge, Mass.

_____, 1966, Topics in the Theory of Generative Grammar, Mouton and Co., The Hague.

Clowes, M., D. Langridge, and R. Zatorski, 1969, "Linguistic Descriptions", Presented paper, Conference on Picture Language Machines, Canberra, Australia, Feb. 1969 (Draft copy).

Fodor and Katz, 1964, The Structure of Language, Prentice-Hall Inc., Englewood Cliffs, New Jersey.

Friedman, Joyce, 1969, A computer system for transformational grammar, CACM 12, 6, 341, June.

Harman, G.H., 1963, Generative grammars without transformation rules - a defence of phrase-structure, Language 39, 597-616.

Ginsburg, S., S. Greibach, and M.A. Harrison, 1967, Stack automata and compiling, JACM 14, 1, 172-201, Jan.

** Klima, Edward S., 1964, Negation in English, p.246 in Fodor and Katz.

Knuth, Donald E., 1965, On the translation of languages from left to right, IC 8, 6, 607, Dec.

Lewis, P.M., and R.E. Stearns, 1968, Syntax-directed transduction, JACM 15, 3, 465-488, July.

Matthews, G.H., 1961, Analysis by synthesis of sentences of natural languages , Proc. 1961 Internat. Cong. on Machine Translation of Languages and Applied Language Analysis, Nat. Phys. Lab., Teddington, England.

Narasimhan, R., 1969, "Computer Simulation of Natural Language Behavior", Invited paper, Conference on Picture Language Machines, Canberra, Aust., Feb. 1969 (Draft copy).

Peterson, W.W., 1957, Addressing for random-access storage, IBM Journ. R. and D., 1, 130.

Petrick, Stanley R., 1965, A Recognition Procedure for Transformational Grammars, Ph.D. Thesis, M.I.T., quoted in Woods, 1967.

______, 1966, A program for transformational syntactic analysis, Air Force Cambridge Res. Lab., U.S.A.F, PSRP 278, AFCRL 66-698.

Postal, Paul M., 1964, Limitations of Phrase Structure Grammars, p.137 in Fodor and Katz.

***

Rosenbaum, P., 1966, The Core Grammar, Contract No. AF 19(628)-5127, AFCRL-66-270, quoted in Petrick, 1966.

Savin, H.B., and E. Perchonock, 1965, Grammatical structure and immediate recall of English sentences, Journ. Verbal Learning and Verbal Behaviour 4, 348-353.

Thorne, 1967 (should read, Bratley, P., H. Dewar, and J.P. Thorne), Recognition of syntactic structure by computer, Nature 216, 5119, 969-973, Dec. 9.

Woods, William, 1967, Semantics for a Question-Answer System, Ph.D. Thesis, Harvard.

Younger, Daniel, 1967, Recognition and parsing of context-free languages in time n^3, IC 10, 2, 189.

Zwicky, A., J. Friedman, B. Hall, and D. Walker, 1965, The MITRE Syntactic Analysis Procedure for Transformational Grammars, Proc. FJCC, Spartan Books, Wash., D.C., quoted in Petrick, 1965.

*Bobrow, D.G., 1963, Syntactic analysis of English by computer - a survey, AFIPS Conference Proceedings 24 (1963 FJCC), 365-387.

**Klein, S., and R.F. Simmons, 1963, A computational approach to grammatical coding of English words, quoted in Bobrow, 1963.

***Resnikoff, H., and J.L. Dolby, 1963, Automatic determination of parts of speech of English words, quoted in Bobrow, 1963.

THOSE WHO CANNOT READ ARE NOT WELL-EDUCATED

A READ
B WELL-EDUCATED

(A,-B)

11

ALL PUDDINGS ARE NICE

A PUDDINGS
B NICE
(-A, B)

A PUDDINGS
B NICE
(-A, B)

THIS DISH IS A PUDDING

A THIS
B DISH
C PUDDING

(-A,-B, C)

NO NICE THINGS ARE WHOLESOME

A NICE
B WHOLESOME

(-A,-B)