36

THE AUTOLING SYSTEM[1, 2]

by

Sheldon Klein
William Fabens
Robert G. Herriot
William J. Katke
Michael A. Kuppin
Alicia E. Towster

Technical Report #43

September 1968

[1]A portion of this paper was presented at the 42nd Annual Meeting of the Linguistic Society of America, December 1967, Chicago, under the title "Testing an Automated Linguistic Fieldworker".

[2]This research has been supported in part by the Wisconsin Alumni Research Foundation and the National Science Foundation.

ABSTRACT

The AUTOLING system represents an attempt to replace the human linguist with a machine in the process of linguistic fieldwork with an informant. To the extent that the attempt succeeds, the analytic and heuristic methodology of live linguists can be considered formalized.

The current system consists of three as yet unjoined components: a morphological analyzer, a program for learning context-free phrase structure grammar, and a program for learning monolingual and bilingual transformations. All programs are written in ALGOL and operational on the Burroughs B-5500 computer. The capabilities of the system are illustrated with examples of its treatment of selected problems in English, Latin, Roglai, Indonesian, Thai, Chinese and German.

TABLE OF CONTENTS

1.0 Introduction.

The AUTOLING system* is, in conception, an on-line computer program that replaces a human linguist in analytic interaction with a live informant. The discovery procedures are heuristic rather than algorithmic; an algorithm is here defined as a method that guarantees success; a heuristic as an analytic approach that may work, but does not necessarily guarantee success in all cases.

1.1 On Discovery Procedures, Algorithmic and Otherwise.

Zellig Harris probably deserves the most credit for expositing in 1946 [5] and 1951 [6] the method of using massive distributional tabulations of units in texts to determine the structure of a language. Rulon Wells soon followed in 1947 [21] with an approach to syntactic analysis that also rested ultimately on distributional analysis.

In 1959 [18], Solomonoff suggested an automatic method for phrase structure language grammar discovery that, while adding little to the methodology of linguistic analysis, introduced the problem to a computing audience and suggested the use of an informant.

Distributributional analysis grammar discovery procedures alone will not work for two reasons. The first is practical: the combinatorial computations required in the analysis of say a million words of running text would yield intermediate data exceeding the memory capacity of any known computer--and the computation time of course would be commensurately prohibitive. More formally, E. Shamir demonstrated in 1962 [16] that it is impossible to obtain a discovery method for context free phrase structure grammars using natural language text alone as input. (He also suggested that the use of an informant would not alter the problem.)

---

*Written in ALGOL for the Burroughs B-5500 Computer.

Paul Garvin, in 1961 [2] suggested the use of heuristics in guiding the choice of distributional tests. His methodology still used only edited texts as a data base.

E. Mark Gold, in 1964 [4], indicates a formal demonstration that a discovery algorithm for learning context free grammar is possible if interaction with an informant is permitted.

Garvin discussed the use of heuristics in automatic informant work in 1965 [2].*

Work on the AUTOLING system has been described in 1967 [10, 11].

Other programs which involve some sort of language learning include those of Knowlton 1962 [12], Uhr 1964 [20] McConlogue & Simmons 1965 [14], and Siklossy 1968 [17].

1.2 The Philosophy Behind AUTOLING

The basic goal of AUTOLING research is to replace the linguist rather than aid him. To the extent that this goal is attained (in the form of a computer program system), the analytic methodology of live linguists can be considered formalized.

Human linguists obtain grammars without performing massive distributional analysis through the use of heuristics which permit them to make testable hypotheses that, when verified, eliminate the need for great masses of distributional analyses.

The ideal AUTOLING system would incorporate all the heuristics that a good linguist uses in fieldwork, and be capable of undertaking the analytic process via interaction with a live informant.

---

*At a RAND Corporation Linguistics Colloquium in December 1964, S. Klein suggested to Paul Garvin, the invited speaker, that he might use the concept of heuristic problem solving and game playing in application to automated informant work.

However, we have not attempted to deal with phonology in the project. The intended informant for the system would be bilingual in English and some other language, and be capable transcribing his non-English language in phonemic notation on a teletype keyboard.

The ideal AUTOLING system would, initially, ask the informant a battery of prestored questions ('How do you say. . . ?) designed to elicit material for morphalogical analysis. At a somewhat later stage of interaction a phrase structure learning component would attempt to learn a phrase structure grammar, accepting as inputs the informant's responses rewritten morphophonemically. A continuous interaction between the phrase structure learning component and the morphological program would take place. Ideally, the program would learn monolingual transformations, and also bilingual ones so that the program might generate its own query list in English. That is, a sentence is generated to test a hypothesized rule, then translated into English; the system would then output a 'How do you say' message followed by the English translation of the test sentence; the informant's reply is then matched against this prediction

This paper, however, deals with the current reality of the AUTOLING system. At the moment it consists of three disconnected components: a morphological analyzer that is not informant interactive; an informant-interactive, context-free phrase structure learning program (the most developed component) which does test rules, but with a 'Can you say' message followed by the test production in the language under analysis rather than its English translation; and an informant-interactive program that learns monolingual and bilingual transformations. Also, in the actual systems, no query lists are used. The informant implicitly provides his own at certain times by feeding in pertinent data.

The reader will please note that all approaches that might involve a human linguist giving the machine advice have been avoided because of the primary goal of the research -- the replacement of the linguist and the concomitant formalization of fieldwork methodology.

There is of course a key secondary goal that demands the same kind of approach: the AUTOLING system logic will be incorporated in the learning component of Klein's computer simulation of Historical Change in Language system [8, 9]. The exact function of AUTOLING in this system is described in a paper presented at the 10th International Congress of Linguists in 1967 [10].

2.0 Morphological Analyzer

This program is essentially the work of Alicia E. Towster. Although a great deal of work has gone into it, it is relatively undeveloped from the point of view of integration into the rest of the AUTOLING system. A number of early versions have been moderately successful in analyzing problems taken from Nida [15], Koutsoudas [13], but have revealed basic problems that have led to major revisions.

One of the key problems is that of control of the gloss metalanguage. Inherent in the analytic procedure is a parallel comparison of strings in the language and their glosses. If the glosses are left in English, the program is limited to making morphological cuts that match the English gloss units. Human analysts draw upon a larger semantic data base and reinterpret the glosses as the problem demands. To incorporate this reinterpretation in a program demands a formalization of the gloss reinterpretation process. This in turn makes mandatory the incorporation of a universal semantic list (not necessarily the semantic distinctive features of Katz and Fodor [7] or Chomsky [1]) with rules for rewriting English glosses in terms of those features, even though many might not be semantic distinctive features in English.

If the claim for 'universiality' of such semantic features disturbs some linguists, they might view them as simple an inventory of all (hopefully) possible semantic units any language in the world might assign to a given English gloss.

The determination of such a list of semantic features is, of course, a momentous task. Accordingly, only a small listing of such features suitable for handling perhaps five or ten assorted text book problems have been determined.

The decision to rewrite glosses in terms of semantic features creates another major analytic problem for a program: the determination of which such features are actually distinctive; or in other terms, what bundles of semantic primitives are to be treated as units in a given language, and how they are distributed. The problem can be made clear from a 'trivial' English example:

Form	Gloss
I eat	1st person, singular, human, animate, eat, indicative, present tense.
you eat	2nd person, singular, human, animate, eat, indicative, present tense.

Clearly, 'I' and 'you' are uniquely associated with the features 'lst person' and '2nd person'. But what of the remainders?

In an effort to handle the problem a distinction between primary and secondary glosses is noted. 'I' and 'you' are assigned 'lst person' and '2nd person', respectively, as primary glosses. The remaining features with each input are assigned to both members of each cut but labelled as secondary glosses.

Later analysis determines the final assignment of the features, and their relabelling as primary.

Other problems that arise in analysis are those of over-cutting. Occasionally what should emerge as a single morpheme is split into two components. Proper analysis of the semantic features associated with these over-cut units at a later stage should provide enough data for recombination, but the pertinent heuristics have not yet been programmed.

Additional analytic heuristics which are in a state of flux pertain to input-sequence sensitivity. Currently, the inputs are analyzed in block units containing a fixed number of forms and glosses. The selection of new analytic blocks for intermediate stages of analysis as a function of what cuts have occurred in the preliminary stages involve heuristics that are also yet to be programmed. Heuristics for grouping allomorphs, and determining morphophonemic rules also await implementation.

3.0 Phrase Structure Heuristic Learning Program

This system, which learns unordered, context-free phrase structure rules is not yet connected to the Morphological Analysis program. It accepts as inputs sentences written with spaces between morphological units. A multi-path parser yields all possible parses of each input. If no complete parses are obtained, the top nodes of the incomplete parses are ordered according to the number of uncombined nodes remaining in each parse.

These incomplete parse top nodes serve as inputs to the basic heuristic 1.

Heuristic 1

$$X \quad Y \quad Z \,] \Longrightarrow {}^{*}S_i \rightarrow X \; Y \; S$$

[where S_i stands for rule: ${}^{*}S_i$ indicates sentence rule]

That is, coin as a rule the closure of the parse. In the case that no rules of the provisional grammar apply to the input string, the string itself is treated as a parse, and heuristic 1 yields:

$${}^{*}S_i \rightarrow \text{morpheme}_1 \;\; \text{morpheme}_2 \ldots, \text{morpheme}_n \;.$$

This of course is what happens to the first input to the system. The remaining learning heuristics cover the various cases in which units in identical environments are assigned to classes.

Heuristic 2 states that two single morphemes in identical environments are assigned to the same class.

Heuristic 2

$$\left.\begin{array}{l} S_1 \rightarrow X \;\; m_1 \;\; Y \\ S_2 \rightarrow X \;\; m_2 \;\; Y \end{array}\right] \Longrightarrow \begin{array}{l} S_3 \rightarrow m_1 \\ S_3 \rightarrow m_2 \end{array}$$

where X and Y are strings consisting of terminal, non-terminal or a mixed combination of units, and m_1 and m_2 are <u>single morphemes</u>, and either

X or Y may be empty but not both.

Heuristic 2 also applies in the case that m_1 and m_2 are strings of one or more non-terminals.

Heuristic 3 states that if a terminal element (morpheme) and a single non-terminal element occur in identical environments, the morpheme is added to the non-terminal class.

Heuristic 3.

$$\left.\begin{array}{l} S_1 \rightarrow X \; S_1 \; Y \\ S_2 \rightarrow X \; m_j \; Y \\ S_i \rightarrow m_i \end{array}\right] \Longrightarrow \begin{array}{l} S_i \rightarrow m_i \\ S_i \rightarrow m_j \\ S_1 \rightarrow X \; S_i \; Y \end{array}$$

where X and Y are strings of terminal, non-terminal or mixed elements, and where X or Z may be empty but not both, and where m_i and m_j are morphemes.

There are also what might be termed negative heuristics, designed to prevent premature ad hoc rule coining. Namely, if one of the strings occurring in an environment identical with some other string should contain one morpheme plus anything else, no new rules are coined (other than to add the tree top to the rule list via heuristic 1). Also, no new rules are coined in the case of frame overlap, e.g.

$$\begin{array}{l} S_1 \rightarrow X \; Y \\ S_2 \rightarrow X \; A \; Y \end{array} \quad \text{or} \quad \begin{array}{l} S_1 \rightarrow X \; A \; B \; Y \\ S_2 \rightarrow X \; A \; A \; B \; Y \end{array}$$

where X and Y are defined as above, and A and B are strings of non-terminals.

The last pattern matching, phrase structure learning heuristic coins recursive rules;

Heuristic 4.

$$\left.\begin{array}{l} S_1 \rightarrow X\ A\ Y \\ S_2 \rightarrow X\ A\ A\ Y \\ \end{array}\right] \Longrightarrow \begin{array}{l} S_3 \rightarrow A \\ S_3 \rightarrow S_3\ A \\ S_1 \rightarrow X\ S_3\ Y \end{array}$$

where X and Y are defined as before and A is a string of one or more nonterminals. Actually, this heuristic is not really needed for the coining of recursive rules. The other heuristics are capable of learning indirect recursion in a variety of ways.

3.1 Rule Testing Heuristics

The remaining heuristics pertain to the testing of the validity of the rules coined by heuristics 2, 3, & 4. Except for heuristic 5, they are not numbered, and are best described in terms of the flow of the program.

3.1.1 Substitution and Informant Queries

The validity of each rule coined by heuristics 2-4 are tested via substitution, and test sentence generation. Each time a new class is coined, the system inspects each rule in the grammar for tokens of elements of the new class. When such a rule is found, the system tentatively makes the substitution of the new class name for the token of its element, and performs a test generation in the following manner:

Only full sentences are offered to the informant for acceptance or rejection. Accordingly, if the substitution is made in an unstarred rule (non-sentence), the system climbs upwards through the heirarchy of rules to the first starred rule it can find that might yield the test rule in a generation path. The program then generates randomly downwards but with a forced choice of the rule in which the substitution was made.

The test sentence is outputted on the teletype with the query:

CAN YOU SAY:

If the informant types:

YES

the substitution is made.

If the reply is NO, no substitution is made, and the test sentence is added to the illegal list. If a given rule contains two strings in the domain of the substitution, each is tried individually and serially, and if any succeeds, the next substitution is in the revised string.

If substitution should yield duplicate rules, one of the duplicates is deleted and all reference to the deleted rule are replaced by references to the remaining one.

The system also avoids the creation by substitution of unexitable node loops involving rules of the type:

$$S_i \rightarrow S_j$$
$$\vdots$$
$$S_j \rightarrow S_i$$

In the case that all substitutions of a newly posited rule fail, the heuristic is assumed to have failed, and the next one is tried. If all heuristics 2-4 fail for a given top node, then the remaining top nodes are each subjected to the same heuristics and testing. In the event of all failures, the original top node is added as a rule via heuristic 1. There remains one special rule coining heuristics governing class splitting applicable to morphemes. It is treated in this section because its application also involves rule testing.

Suppose the grammar contains:

Heuristic 5

$$S_1 \rightarrow m_1$$
$$S_1 \rightarrow m_2$$
$$S_2 \rightarrow X \ S_1 \ Y$$
$$S_3 \rightarrow W \ S_1 \ Z$$

where X, Y, W, & Z are strings of terminals or non-terminals or both.

Suppose that, via heuristic 2, the rule

$$S_1 \rightarrow m_3$$

is posited, and that the substitution test succeeds in S_2, but fails in S_3. In that case the following rules are coined

$$S_1 \rightarrow m_1$$
$$S_1 \rightarrow m_2$$
$$S_4 \rightarrow S_1$$
$$S_4 \rightarrow m_3$$
$$S_2 \rightarrow X \ S_4 \ B$$
$$S_3 \rightarrow W \ S_1 \ Z$$

and the rule $S_1 \rightarrow m_3$ is thrown out.

3.1.2 Parsing Illegal Sentences and Recycling.

Parsing

Each test sentence rejected by the informant is added to a list of "illegals". All illegals determined during a single testing cycle are treated as belonging to the same frame. A frame is the interaction and processing that takes place after an input sentence from the informant. All the rule substitutions, test sentences and informant 'yes' and 'no' responses arising from the informant input sentence are treated as belonging to the same testing frame.

As each rule is coined or updated, the parsing component of the program attempts to parse each illegal sentence of the current frame. If an illegal sentence parses, then the postulated new rule or substitution is disregarded.

After each five informant inputs, the system attempts to parse all of its illegals, from all previous processing frames. Because the testing is inherently incomplete, bad rules may have slipped into the grammar. If at this time any of the illegal sentences is successfully parsed, the system enters a recycle mode, which is described below. (At one stage of development the program parsed all illegals from all frames before coining a new rule. This procedure proved too costly in processing time and the current methodology was used instead.) It also happens that some illegals are the product of posited spurious recursive rules, and may accordingly be very long as well as illegal. Such sentences if parsed to completion, required as much as 5 or even 10 minutes of computer time. After determining that 99% of parsable sentences were parsed in 1 minute or less, we made the system assume a sentence is unparsable if the completed parse is not found in less than one minute. This limitation is for the parsing of illegals only; no limitation exists on parsing time for new inputs from the informant.

Recycling

As indicated earlier, an attempt is made to parse all illegal sentences from all previous frames after every 5 informant sentence inputs. If an illegal is parsed, then some bad rule has slipped past the tests. Accordingly the system destroys its entire grammar, but saves the list of past input sentences and the list of illegal sentences. The analysis then begins anew, but with the following differences. The system first processes the recorded input list instead of immediately soliciting new inputs from the informant. The input list is reordered slightly as a rough attempt to overcome the sequence sensitivity of the learning process. Specifically, the last 5 informant sentence inputs before the recycle are put at the head of the list and processed first. Also, the illegal sentence that caused the recycle is made a permanent member of the current frame illegal set, and an attempt is made to parse it during each rule coining step in the recycle mode. If there have been other recycles, the responsible illegals are also made permanent members of the "current frame illegal set".

There may be recycles within recycles with a limit of depth 3. If this limit is reached the system gives up on the whole analysis. No limit exists for unnested recycles.

3.2 Test Problems

In the following copies of actual teletype output, the only human inputs are:

1. An input sentence following a program generated NEXT.
2. *TYPE in response to a computer generated NEXT which is a request by the human for a listing of the current grammar.
3. YES and NO in response to program requests, CAN YOU SAY THIS followed by a program generated test sentence.

Other messages the system can accept are:

4. A *RESTART message permits restarting with an empty grammar.
5. A *SAVE followed by a single digit makes the system periodically save the total state of the learning process on disc file at periodic intervals. At each saving point the system outputs a message, indicated the name of the saved file in terms of the initial digit after the first *SAVE message, plus a sequence number.
6. A *RESTART message followed by a file number reinitializes the program to the state stored under that file name. This feature permits the continued analysis, over a long period of time, of up to 9 different languages. It also permits partial backup restarts in case of accidents or erroneous inputs.

Some of the problems were taken from Koutsoudas' test, Writing Transformational Grammars: An Introduction [13]. The sentences were entered as inputs in the order they occurred in each problem. Often the program generated as a test case a problem sentence not yet entered. In this case the informant usually omitted the repitition. Each time the computer grammar was adequate for parsing an input, it outputed the message:

PARSED OK

Often, the informant's responses to the machine's test utterances did not come from a full knowledge of the language. The informant behaved consistently, but may have said YES and NO to hypothetical constructions that may have deserved different replies.

3.2.1 Embedding Illustration: Artificial Language.

```
BEGIN AUTOLING
NEXT: (X  Y←)

NEXT: (*TYPE ←)
*S1  :=  X  Y

(X  X  Y  Y←)

NEXT: (*TYPE ←)
*S1  :=  X  Y
*S2  :=  X  S1  Y

(X  X  X  Y  Y  Y←)
CAN  YOU  SAY:
     X  X  X  X  X  X  Y  Y  Y  Y  Y  Y
(YES ←)

NEXT: (*TYPE ←)
  *S1  :=  X  Y
  *S2  :=  X  S3  Y
   S3  :=  S2
   S3  :=  S1
(X  X  X  X  Y  Y  Y  Y←)
-PARSED OK-

NEXT:
(X X X X X X X X X X X X Y Y Y Y Y Y Y Y Y Y Y Y←)
-PARSED  OK-
```

[circled items from human]

3.2.2 English I

```
BEGIN AUTOLING
NEXT:
I WANT HIM TO GO ←

NEXT:  I NEED HIM TO GO←

NEXT:  :*TYPE
  *S1  :=  I  S2  HIM  TO  GO
   S2  :=  NEED
   S2  :=  WANT


I WANT HER TO GO ←
CAN YOU SAY:
    I NEED HER TO GO
YES ←

NEXT:  *TYPE ←
  *S1  :=  I  S2  S3  TO GO
   S2  :=  NEED
   S2  :=  WANT
   S3  :=  HER
   S3  :=  HIM

YOU WANT HER TO GO ←
CAN YOU SAY:
    YOU NEED HER TO GO
YES ←

NEXT:  *TYPE ←
  *S1  :=  S4  S2  S3  TO GO
   S2  :=  NEED
   S2  :=  WANT
   S3  :=  HER
   S3  :=  HIM
   S4  :=  YOU
   S4  :=  I
YOU WANT HIM TO RUN ←
CAN YOU SAY:
    YOU NEED HIM TO RUN
YES ←
*PARSING ILLEGALS*
```

```
NEXT:  HE WANT  S  HER  TO  GO←
CAN YOU SAY:
    HE WANT  S  HIM TO RUN
YES←

NEXT:  *TYPE←
  *S1  :=  S4  S2  S3  TO  S5
   S2  :=  NEED
   S2  :=  WANT
   S3  :=  HER
   S3  :=  HIM
   S4  :=  YOU
   S4  :=  I
   S5  :=  RUN
   S5  :=  GO
  *S6  :=  HE  S2  S  S3  TO  S5
SHE WANT  S  HIM  TO  RUN←
CAN YOU SAY:
    SHE WANT  S  HER TO GO
YES←

NEXT:  *TYPE←
  *S1  :=  S4  S2  S3  TO  S5
   S2  :=  NEED
   S2  :=  WANT
   S3  :=  HER
   S3  :=  HIM
   S4  :=  YOU
   S4  :=  I
   S5  :=  RUN
   S5  :=  GO
  *S6  :=  S7  S2  S  S3  TO  S5
   S7  :=  SHE
   S7  :=  HE
SHE WANT  S  HER TO RUN←
-PARSED OK-
```

3.2.3 English II.

```
BEGIN AUTOLING
NEXT:

      EAT THE CAT ←

NEXT:  EAT A CAT ←

NEXT:  EAT THE CAT  S ←
CAN YOU SAY:
    EAT A CAT  S
NO ←

NEXT:  *TYPE ←
  *S1  :=  EAT  S2  CAT
   S2  :=  A
   S2  :=  THE
  *S3  :=  EAT THE CAT  S

EAT A DOG ←
CAN YOU SAY:
    EAT THE DOG
YES ←
CAN YOU SAY:
    EAT THE DOG S
YES ←

NEXT:  *TYPE ←
  *S1  :=  EAT  S2  S4
   S2  :=  A
   S2  :=  THE
  *S3  :=  EAT THE S4  S
   S4  :=  DOG
   S4  :=  CAT

KNOW THE DOG ←
CAN YOU SAY:
    KNOW A DOG
YES ←
CAN YOU SAY:
    KNOW THE CAT  S
YES ←
*PARSING ILLEGALS*
```

```
NEXT: *TYPE←
  *S1  := S5  S2  S4
   S2  := A
   S2  := THE
  *S3  := S5  THE  S4  S
   S4  := DOG
   S4  := CAT
   S5  := KNOW
   S5  := EAT

EAT THESE CAT  S←
CAN YOU SAY:
    EAT THESE DOG  S
YES←

NEXT: *TYPE←
  *S1  := S5  S2  S4
   S2  := A
   S2  := THE
  *S3  := S5  S6  S4  S
   S4  := DOG
   S4  := CAT
   S5  := KNOW
   S5  := EAT
   S6  := THESE
   S6  := THE

EAT SOME CAT  S←

NEXT: EAT SOME CAT←

NEXT: *TYPE←
  *S1  := S5  S2  S4
   S2  := SOME
   S2  := A
   S2  := THE
  *S3  := S5  S6  S4  S
   S4  := DOG
   S4  := CAT
   S5  := KNOW
   S5  := EAT
   S6  := SOME
   S6  := THESE
   S6  := THE

EAT THOSE CAT  S←
```

```
NEXT: I EAT A CAT←

CAN YOU SAY:
   I KNOW A CAT

YES ←
*PARSING ILLEGALS*

NEXT: *TYPE ←
 *S1  :=  S5  S2  S4
  S2  :=  SOME
  S2  :=  A
  S2  :=  THE
 *S3  :=  S5  S6  S4  S
  S4  :=  DOG
  S4  :=  CAT
  S5  :=  KNOW
  S5  :=  EAT
  S6  :=  THOSE
  S6  :=  SOME
  S6  :=  THESE
  S6  :=  THE
 *S7  :=  I  S1

YOU EAT A CAT ←

CAN YOU SAY:
    YOU KNOW A DOG
YES ←

NEXT: *TYPE ←
 *S1  :=  S5  S2  S4
  S2  :=  SOME
  S2  :=  A
  S2  :=  THE
 *S3  :=  S5  S6  S4  S
  S4  :=  DOG
  S4  :=  CAT
  S5  :=  KNOW
  S5  :=  EAT
  S6  :=  THOSE
  S6  :=  SOME
  S6  :=  THESE
  S6  :=  THE
 *S7  :=  S8  S1
  S8  :=  YOU
  S8  :=  I
```

```
THE DOG S KNOW A CAT←

CAN YOU SAY:
    THOSE CAT S KNOW SOME DOG
YES ←

NEXT: *TYPE ←
 *S1 := S5 S2 S4
  S2 := SOME
  S2 := A
  S2 := THE
 *S3 := S5 S6 S4 S
  S4 := DOG
  S4 := CAT
  S5 := KNOW
  S5 := EAT
  S6 := THOSE
  S6 := SOME
  S6 := THESE
  S6 := THE
 *S7 := S8 S1
  S8 := YOU
  S8 := I
 *S9 := S6 S4 S S1
```

3.2.4 Latin: Koutsoudas #26

PROBLEM 26: LATIN

1. puer virum videt	*The boy sees the man.*
2. vir puerum videt	*The man sees the boy.*
3. puer virum defendit	*The boy defends the man.*
4. vir puero nocet	*The man harms the boy.*
5. puer viro nocet	*The boy harms the man.*
6. puer viro subvenit	*The boy helps the man.*
7. puer viri meminit	*The boy remembers the man.*
8. puer viro meminit	*The boy remembers the man.*

```
BEGIN AUTOLING
NEXT:
PUER VIR UM VIDE T←

NEXT: VIR PUER UM VIDE T←

NEXT:
PUER VIR UM DEFENDI T←
CAN YOU SAY:
    VIR PUER  UM DEFENDI T
YES ←

NEXT: *TYPE ←
 *S1  :=  PUER VIR UM S3 T
 *S2  :=  VIR PUER UM S3 T
  S3  :=  DEFENDI
  S3  :=  VIDE

VIR PUER O NOCE T←

NEXT: PUER VIR O NOCE T←
*PARSING ILLEGALS*

NEXT: PUER VIR O SUBVENI T←
CAN YOU SAY:
    VIR PUER O SUBVENI T
YES ←

NEXT: *TYPE ←
 *S1  :=  PUER VIR UM S3  T
 *S2  :=  VIR PUER UM S3  T
  S3  :=  DEFENDI
  S3  :=  VIDE
 *S4  :=  VIR PUER O S6 T
 *S5  :=  PUER VIR O S6 T
  S6  :=  SUBVENI
  S6  :=  NOCE

PUER VIR I MEMINI T←

NEXT: PUER VIR O MEMINI T←
CAN YOU SAY:
    VIR PUER O MEMINI T
YES ←
CAN YOU SAY:
    PUER VIR I SUBVENI T
NO ←
```

```
NEXT:  *TYPE ←
  *S1  :=  PUER VIR UM S3 T
  *S2  :=  VIR PUER UM S3 T
   S3  :=  DEFENDI
   S3  :=  VIDE
  *S4  :=  VIR PUER O S6 T
  *S5  :=  PUER VIR O S6 T
   S6  :=  MEMINI
   S6  :=  SUBVENI
   S6  :=  NOCE
  *S7  :=  PUER VIR I MEMINI T
```

[The sentence inputs that follow have been added to the problem]

```
VIR PUER UM VIDE T ←
-PARSED OK-

NEXT:  VIR PUER I MEMINI T ←
CAN YOU SAY:
     VIR PUER I  NOCE T
NO ←
*PARSING ILLEGALS*

NEXT:  *TYPE ←
  *S1  :=  PUER VIR UM S3 T
  *S2  :=  VIR PUER UM S3 T
   S3  :=  DEFENDI
   S3  :=  VIDE
  *S4  :=  VIR PUER O S6 T
  *S5  :=  PUER VIR O S6 T
   S6  :=  MEMINI
   S6  :=  SUBVENI
   S6  :=  NOCE
  *S7  :=  PUER VIR I MEMINI T
  *S12 :=  VIR PUER I MEMINI T
PUER VIR O NOCE T ←
-PARSED OK-

NEXT:  PUER OPPID UM VIDE T ←
CAN YOU SAY:
     PUER OPPID UM DEFENDI T
YES ←
CAN YOU SAY:
     OPPID PUER O SUBVENI T
YES ←
CAN YOU SAY:
     OPPID PUER I MEMINI T
YES ←
```

```
CAN YOU SAY:
    OPPID PUER UM VIDE T
YES ←
CAN YOU SAY:
    PUER OPPID O MEMINI T
YES ←
CAN YOU SAY:
    PUER OPPID I MEMINI T
YES ←

NEXT:  *TYPE ←
  *S1  := PUER S13 UM S3 T
  *S2  := S13  PUER UM S3 T
   S3  := DEFENDI
   S3  := VIDE
  *S4  := S13 PUER O S6 T
  *S5  := PUER S13 O  S6 T
   S6  := MEMINI
   S6  := SUBVENI
   S6  := NOCE
  *S7  := PUER S13 I MEMINI T
  *S12 := S13 PUER I MEMINI T
   S13 := OPPID
   S13 := VIR
VIR OPPID UM VIDE T ←
CAN YOU SAY:
    OPPID OPPID UM VIDE T
YES ←
CAN YOU SAY:
    PUER PUER I MEMINI T
YES ←
CAN YOU SAY:
    PUER PUER O MEMINI T
YES ←
CAN YOU SAY:
    PUER PUER UM VIDE T
YES ←
CAN YOU SAY:
    PUER PUER O NOCE T
YES ←
CAN YOU SAY:
    OPPID OPPID O SUBVENI T
YES ←
CAN YOU SAY:
    PUER PUER UM DEFENDI T
YES ←
```

```
CAN YOU SAY:
     OPPID VIR UM DEFENDI T
YES ←
CAN YOU SAY:
     VIR VIR I MEMINI T
YES ←
CAN YOU SAY:
     PUER OPPID O NOCE T
YES ←
CAN YOU SAY:
     OPPID OPPID O MEMINI T
YES ←
CAN YOU SAY:
     OPPID OPPID I MEMINI T
YES ←
CAN YOU SAY:
     VIR OPPID I MEMINI T
YES ←

NEXT:  *TYPE ←
  *S1  := S13 S13 UM S3 T
  *S2  := S1
   S3  := DEFENDI
   S3  := VIDE
  *S4  := S13 S13 O S6 T
  *S5  := S4
   S6  := MEMINI
   S6  := SUBVENI
   S6  := NOCE
  *S7  := S12
  *S12 := S13 S13 I MEMINI T
   S13 := PUER
   S13 := OPPID
   S13 := VIR
```

3.2.5 Roglai: Koutsoudas #58

PROBLEM 58: ROGLAI

1. ama naw
   Father went.
2. ama naw tubrɔy
   Father went yesterday.
3. adəy ʔbək bu
   The child ate rice.
4. ama naw judəy adəy ʔbək bu
   Father went after the child ate rice.
5. adəy ʔbək bu tubrɔy
   The child ate rice yesterday.
6. adəy ʔbək bu judəy ama naw
   The child ate rice after the father went.
7. tubrɔy ama naw
   Yesterday father went.
8. judəy adəy ʔbɔk bu ama naw
   After the child ate rice, the father went.
9. tubrəy adəy ʔbɔk bu
   Yesterday the child ate rice.
10. judəy ama naw adəy ʔbək bu
    After father went, the child ate rice.
11. adəy naw musu·p
    The child went early in the morning.
12. ama ʔbək ika·t
    Father ate fish.
13. adəy naw judəy ama ʔbək ika·t
    The child went after the father ate fish.
14. adəy naw juma ama ʔbək ika·t
    The child went before father ate fish.
15. musu·p adəy naw
    Early in the morning the child went.
16. judəy ama ʔbək ika·t adəy naw
    After the father ate fish, the child went.
17. juma ama ʔbɔk ika·t adəy naw
    Before the father ate fish, the child went.
18. ama naw juma adəy ʔbək bu
    Father went before the child ate rice.
19. juma adəy ʔbɔk bu ama naw
    Before the child ate rice, the father went.
20. adəy ʔbɔk ika·t tubrəy
    The child ate fish yesterday.
21. tubrəy adəy ʔbək ika·t
    Yesterday the child ate fish.
22. judəy ama naw adəy ʔbək ika·t
    After father went, the child ate fish.
23. juma ama naw adəy ʔbək ika·t
    Before father went, the child ate fish.

```
BEGIN AUTOLING
NEXT:  AMA NAW ←

NEXT:  ADEY NAW ←                         [Added input]

NEXT:  AMA NAW TUBREY ←
CAN YOU SAY:
    ADEY NAW TUBREY
YES ←

NEXT:  ADEY QBEK BU ←
CAN YOU SAY:
    AMA QBEK BU
YES ←

NEXT:  AMA NAW JUDEY ADEY QBEK BU ←
CAN YOU SAY:
    ADEY NAW JUDEY AMA QBEK BU
YES ←
*PARSING ILLEGALS*

NEXT:  ADEY QBEK BU TUBREY ←
CAN YOU SAY:
    AMA QBEK BU TUBREY
YES ←
CAN  YOU SAY:
    AMA QBEK BU JUDEY AMA QBEK BU
YES ←
CAN YOU SAY:
    AMA NAW JUDEY ADEY NAW
YES ←

NEXT:  *TYPE ←
  *S1  :=  S2 NAW
   S2  :=  ADEY
   S2  :=  AMA
  *S3  :=  S6  TUBREY
  *S4  :=  S2  QBEK BU
  *S5  :=  S6 JUDEY S6
   S6  :=  S4
   S6  :=  S1

ADEY QBEK BU JUDEY AMA NAW ←
--PARSED OK--
```

```
NEXT:  TUBREY AMA NAW ←
CAN YOU SAY:
    TUBREY ADEY QBEK BU
YES ←

NEXT:  JUDEY ADEY QBEK BU AMA NAW ←
CAN YOU SAY:
    JUDEY AMA QBEK BU ADEY QBEK BU
YES ←

NEXT:  JUDEY AMA NAW ADEY QBEK BU ←
-PARSED OK-
*PARSING ILLEGALS*

NEXT:  ADEY NAW MUSUUP ←
CAN YOU SAY:
    ADEY QBEK BU MUSUUP
YES ←
CAN YOU SAY:
    MUSUUP ADEY QBEK BU
YES ←

NEXT:  *TYPE ←
 *S1  :=  S2 NAW
  S2  :=  ADEY
  S2  :=  AMA
 *S3  :=  S6  S9
 *S4  :=  S2  QBEK BU
 *S5  :=  S6  JUDEY  S6
  S6  :=  S4
  S6  :=  S1
 *S7  :=  S9  S6
 *S8  :=  JUDEY  S6  S6
  S9  :=  MUSUUP
  S9  :=  TUBREY

AMA QBEK IKAAT ←
CAN YOU SAY:
    ADEY QBEK IKAAT
YES ←
```

```
NEXT:  *TYPE ←
 *S1  :=  S2  NAW
  S2  :=  ADEY
  S2  :=  AMA
 *S3  :=  S6  S9
 *S4  :=  S2  QBEK  S10
 *S5  :=  S6  JUDEY  S6
  S6  :=  S4
  S6  :=  S1
 *S7  :=  S9  S6
 *S8  :=  JUDEY  S6  S6
  S9  :=  MUSUUP
  S9  :=  TUBREY
  S10:=  IKAAT
  S10:=  BU

ADEY NAW JUDEY AMA QBEK IKAAT ←
-PARSED OK-

NEXT:  ADEY NAW JUMA AMA QBEK IKAAT ←
CAN YOU SAY:
    ADEY NAW JUMA AMA NAW
YES ←
CAN YOU SAY
    JUMA AMA NAW AMA NAW
YES ←
CAN YOU SAY:
    AMA NAW JUDEY AMA NAW
YES ←
CAN YOU SAY:
    JUDEY ADEY NAW ADEY NAW
YES ←
CAN YOU SAY:
    TUBREY AMA QBEK IKAAT
YES ←
CAN YOU SAY:
    TUBREY ADEY NAW
YES ←
CAN YOU SAY:
    AMA QBEK IKAAT TUBREY
YES ←
CAN YOU SAY:
    AMA QBEK BU JUMA AMA NAW
YES ←
CAN YOU SAY:
    MUSUUP ADEY NAW ADEY QBEK IKAAT AMA QBEK BU
NO ←
```

```
CAN YOU SAY:
    ADEY QBEK BU JUMA AMA QBEK IKAAT AMA NAW ADEY NAW AMA QBEK BU
NO ←

NEXT:  *TYPE ←
  *S1  := S2  NAW
   S2  := ADEY
   S2  := AMA
  *S3  := S5
  *S4  := S2  QBEK  S10
  *S5  := S6  S12
   S6  := S4
   S6  := S1
  *S7  := S8
  *S8  := S12  S6
   S9  := MUSUUP
   S9  := TUBREY
   S10:= IKAAT
   S10:= BU
   S11:= JUMA
   S11:= JUDEY
   S12:= S11  S6
   S12:= S9

MUSUUP ADEY NAW ←
-PARSED OK-
*PARSING ILLEGALS*

NEXT:  JUDEY AMA QBEK IKAAT ADEY NAW ←
-PARSED OK-

NEXT:  JUMA AMA QBEK IKAAT ADEY NAW ←
-PARSED OK-

NEXT:  AMA NAW JUMA ADEY QBEK BU ←
-PARSED OK-

NEXT:  JUMA ADEY QBEK BU AMA NAW ←
-PARSED OK-

NEXT:  ADEY QBEK IKAAT TUBREY←
-PARSED OK-

NEXT:  TUBREY ADEY QBEK IKAAT ←
-PARSED OK-

NEXT:  JUDEY AMA NAW ADEY QBEK IKAAT ←
-PARSED OK-
```

```
NEXT:  JUMA AMA NAW ADEY QBEK IKAAT ←
-PARSED OK-

NEXT:  *TYPE ←
  *S1  :=  S2  NAW
   S2  :=  ADEY
   S2  :=  AMA
  *S3  :=  S5
  *S4  :=  S2  QBEK  S10
  *S5  :=  S6  S12
   S6  :=  S4
   S6  :=  S1
  *S7  :=  S8
  *S8  :=  S12  S6
   S9  :=  MUSUUP
   S9  :=  TUBREY
   S10 :=  IKAAT
   S10 :=  BU
   S11 :=  JUMA
   S11 :=  JUDEY
   S12 :=  S11  S6
   S12 :=  S9
```

As indicated earlier, fewer heuristics operate on strings containing individual morphemes. Accordingly, to add more members to the verb classes, the cover morphemes VERBTRAN and VERBINTRAN were introduced into the inputs with the following result:

```
AMA VERBINTRAN ←
CAN YOU SAY:
    ADEY VERBINTRAN
YES ←

NEXT:  *TYPE ←
  *S1  :=  S2  S15
   S2  :=  ADEY
   S2  :=  AMA
  *S3  :=  S5
  *S4  :=  S2  QBEK  S10
  *S5  :=  S6  S12
   S6  :=  S4
   S6  :=  S1
  *S7  :=  S8
  *S8  :=  S12  S6
   S9  :=  MUSUPP
   S9  :=  TUBREY
```

```
   S10:= IKAAT
   S10:= BU
   S11:= JUMA
   S11:= JUDEY
   S12:= S11 S6
   S12:= S9
   S15:= VERBINTRAN
   S15:= NAW

ADEY VERBTRAN BU ←
CAN YOU SAY:
     ADEY VERBTRAN IKAAT
YES ←
CAN YOU SAY:
     AMA VERBTRAN BU
YES ←
CAN YOU SAY:
     AMA VERBTRAN IKAAT
YES ←
CAN YOU SAY:
     ADEY VERBINTRAN JUMA AMA NAW
YES ←

NEXT: *TYPE ←
  *S1  := S4
   S2  := ADEY
   S2  := AMA
  *S3  := S5
  *S4  := S2 S17
  *S5  := S6 S12
   S6  := S1
  *S7  := S8
  *S8  := S12 S6
   S9  := MUSUUP
   S9  := TUBREY
   S10:= IKAAT
   S10:= BU
   S11:= JUMA
   S11:= JUDEY
   S12:= S11 S6
   S12:= S9
   S15:= VERBINTRAN
   S15:= NAW
   S16:= VERBTRAN
   S16:= QBEK
   S17:= S16 S10
   S17:= S15
```

3.2.6 Indonesian: Koutsoudas #43 and #9 Combined

Some additional sentences were added to the corpus to link the two problems.

PROBLEM 43: INDONESIAN

1. guru itu makan səmaŋka
   teacher the eat watermelon
   The teacher is eating watermelon.
2. guru itu rupaña makan səmaŋka
   The teacher is apparently eating watermelon.
3. rupaña guru itu makan səmaŋka
   The teacher is apparently eating watermelon.
4. guru itu makan səmaŋka rupaña
   The teacher is apparently eating watermelon.
5. guru itu disini kamarin makan səmaŋka
   teacher the here yesterday eat watermelon
   The teacher ate watermelon here yesterday.
6. disini kamarin guru itu makan səmaŋka
   The teacher ate watermelon here yesterday.
7. guru itu makan səmaŋka disini kamarin
   The teacher ate watermelon here yesterday.
8. makan səmaŋka guru itu
   The teacher is eating watermelon.
9. rupaña makan səmaŋka guru itu
   Apparently the teacher is eating watermelon.
10. makan səmaŋka guru itu rupaña
    The teacher is eating watermelon apparently.
11. makan səmaŋka rupaña guru itu
    The teacher apparently is eating watermelon.

12. disini kamarin makan səmaŋka guru itu
    The teacher ate watermelon here yesterday.
13. makan səmaŋka guru itu disini kamarin
    The teacher ate watermelon here yesterday.
14. makan səmaŋka disini kamarin guru itu
    The teacher ate watermelon here yesterday.

9. INDONESIAN

1. ɔraŋ itu makan kacaŋ kamarin	*The man ate peanuts yesterday.*
2. ɔraŋ itu kamarin makan kacaŋ	*The man ate peanuts yesterday.*
3. kamarin ɔraŋ itu makan kacaŋ	*The man ate peanuts yesterday.*
4. kamarin makan kacaŋ ɔraŋ itu	*The man ate peanuts yesterday.*
5. makan kacaŋ kamarin ɔraŋ itu	*The man ate peanuts yesterday.*
6. makan kacaŋ ɔraŋ itu kamarin	*The man ate peanuts yesterday.*
7. anaʔ itu makan kacaŋ kamarin	*The child ate peanuts yesterday.*
8. anaʔ itu kamarin məmbəli kacaŋ	*The child bought peanuts yesterday.*
9. kamarin guru itu məmbəli kacaŋ	*The teacher bought peanuts yesterday.*
10. kamarin məmbəli kuwe guru itu	*The teacher bought cookies yesterday.*
11. məmbəli kuwe ɔraŋ itu kamarin	*The man bought cookies yesterday.*
12. makan kacaŋ kamarin anaʔ itu	*The child ate peanuts yesterday.*
13. kapan ɔraŋ itu makan kuwe	*When did the man eat cookies?*
14. kapan məmbəli kuwe anaʔ itu	*When did the child buy cookies?*
15. guru itu kapan makan kacaŋ	*When did the teacher eat peanuts?*
16. məmbəli kacaŋ kapan ɔraŋ itu	*When did the man buy peanuts?*
17. məmbəli kuwe anaʔ itu kapan	*When did the child buy cookies?*
18. guru itu makan kuwe kapan	*When did the teacher eat cookies?*
19. siapa makan kacaŋ kamarin	*Who ate peanuts yesterday?*
20. siapa kamarin makan kuwe	*Who ate cookies yesterday?*
21. kamarin siapa məmbəli kuwe	*Who bought cookies yesterday?*
22. yaŋ kamarin məmbəli kacaŋ siapa	*Who bought peanuts yesterday?*
23. yaŋ makan kacaŋ siapa kamarin	*Who ate peanuts yesterday?*
24. yaŋ məmbəli kuwe kamarin siapa	*Who bought cookies yesterday?*

```
BEGIN AUTOLING
NEXT:  GURU ITU MAKAN SEMANGKA ←

NEXT:  ORANG ITU MAKAN SEMANGKA ←         [added to problem]

NEXT:  ANAQ ITU MAKAN SEMANGKA ←          [added to problem]

NEXT:  GURU ITU MEMBELI SEMANGKA ←        [added to problem]
CAN YOU SAY:
    ANAQ ITU MEMBELI SEMANGKA
YES ←
NEXT:  GURU ITU MEMBELI KUWE ←            [added to problem]
CAN YOU SAY:
    ORANG ITU MAKAN KUWE
YES ←
*PARSING ILLEGALS*

NEXT:  GURU ITU MAKAN KACANG ←            [added to problem]

NEXT:  *TYPE ←
  *S1  :=  S2  ITU  S3  S4
   S2  :=  ANAQ
   S2  :=  ORANG
   S2  :=  GURU
   S3  :=  MEMBELI
   S3  :=  MAKAN
   S4  :=  KACANG
   S4  :=  KUWE
   S4  :=  SEMANGKA

GURU ITU RUPANYA MAKAN SEMANGKA ←

CAN YOU SAY:
    GURU ITU RUPANYA MEMBELI KACANG
YES ←

NEXT:  RUPANYA GURU ITU MAKAN SEMANGKA ←
CAN YOU SAY:
    RUPANYA ANAQ ITU MEMBELI SEMANGKA
YES ←

NEXT:  GURU ITU MAKAN SEMANGKA RUPANYA ←
CAN YOU SAY:
    GURU ITU MEMBELI SEMANGKA RUPANYA
YES ←
```

```
NEXT: GURU ITU DISINI KAMARIN MAKAN SEMANGKA ←
CAN YOU SAY:
    ORANG ITU DISINI KAMARIN MEMBELI KUWE
YES ←
*PARSING ILLEGALS*

NEXT: DISINI KAMARIN GURU ITU MAKAN SEMANGKA ←
CAN YOU SAY:
    DISINI KAMARIN ANAQ ITU MAKAN KUWE
YES ←

NEXT: GURU ITU MAKAN SEMANGKA DISINI KAMARIN ←
CAN YOU SAY:
    GURU ITU MAKAN KUWE DISINI KAMARIN
YES ←

NEXT: *TYPE ←
  *S1  := S2  ITU  S3  S4
   S2  := ANAQ
   S2  := ORANG
   S2  := GURU
   S3  := MEMBELI
   S3  := MAKAN
   S4  := KACANG
   S4  := SEMANGKA
  *S5  := S2  ITU  RUPANYA  S3  S4
  *S6  := RUPANYA  S1
  *S7  := S1  RUPANYA
  *S8  := S2  ITU DISINI KAMARIN  S3  S4
  *S9  := DISINI KAMARIN  S1
  *S10:= S1  DISINI KAMARIN

MAKAN SEMANGKA GURU ITU ←
CAN YOU SAY:
    MEMBELI KUWE ORANG ITU
YES ←

NEXT: RUPANYA MAKAN SEMANGKA GURU ITU ←
CAN YOU SAY:
    RUPANYA MAKAN KUWE ORANG ITU
YES ←
CAN YOU SAY:
    MAKAN KUWE ANAQ ITU DISINI KAMARIN
YES ←
CAN YOU SAY:
    MAKAN KACANG GURU ITU RUPANYA
YES ←
CAN YOU SAY:
    DISINI KAMARIN MEMBELI KUWE ORANG ITU
YES ←
```

```
NEXT:  *TYPE ←
  *S1  :=  S2  ITU  S3  S4
   S2  :=  ANAQ
   S2  :=  ORANG
   S2  :=  GURU
   S3  :=  MEMBELI
   S3  :=  MAKAN
   S4  :=  KACANG
   S4  :=  KUWE
   S4  :=  SEMANGKA
  *S5  :=  S2  ITU  RUPANYA  S3  S4
  *S6  :=  RUPANYA  S12
  *S7  :=  S12  RUPANYA
  *S8  :=  S2  ITU DISINI KAMARIN  S3  S4
  *S9  :=  DISINI KAMARIN  S12
  *S10 :=  S12  DISINI KAMARIN
  *S11 :=  S3  S4  S2  ITU
   S12 :=  S11
   S12 :=  S1

MAKAN SEMANGKA GURU ITU RUPANYA ←
-PARSED OK-
*PARSING ILLEGALS*

NEXT:  MAKAN SEMANGKA RUPANYA GURU ITU ←
CAN YOU SAY:
     MEMBELI KUWE RUPANYA GURU ITU
YES ←

NEXT:  DISINI KAMARIN MAKAN SEMANGKA GURU ITU ←
-PARSED OK-

NEXT:  MAKAN SEMANGKA GURU ITU DISINI KAMARIN ←
-PARSED OK-

NEXT:  MAKAN SEMANGKA DISINI KAMARIN GURU ITU ←
CAN YOU SAY:
     MEMBELI KUWE DISINI KAMARIN ANAQ ITU
YES ←
```

```
NEXT: *TYPE ←
  *S1  := S2 ITU S3 S4
   S2  := ANAQ
   S2  := ORANG
   S2  := GURU
   S3  := MEMBELI
   S3  := MAKAN
   S4  := KACANG
   S4  := KUWE
   S4  := SEMANGKA
  *S5  := S2 ITU RUPANYA S3 S4
  *S6  := RUPANYA S12
  *S7  := S12 RUPANYA
  *S8  := S2 ITU DISINI KAMARIN S3 S4
  *S9  := DISINI KAMARIN S12
  *S10 := S12 DISINI KAMARIN
  *S11 := S3 S4 S2 ITU
   S12 := S11
   S12 := S1
  *S13 := S3 S4 RUPANYA S2 ITU
  *S14 := S3 S4 DISINI KAMARIN S2 ITU

ORANG ITU MAKAN KACANG KAMARIN ←
CAN YOU SAY:
    MAKAN KACANG ORANG ITU KAMARIN
YES ←
CAN YOU SAY:
    KAMARIN ANAQ ITU MAKAN KUWE
YES ←
CAN YOU SAY:
    ORANG ITU KAMARIN MEMBELI KUWE
YES ←
CAN YOU SAY:
    MAKAN SEMANGKA RUPANYA ORANG ITU
YES ←
CAN YOU SAY:
    ANAQ ITU DISINI RUPANYA MEMBELI SEMANGKA
NO ←
CAN YOU SAY:
    DISINI RUPANYA ANAQ ITU MAKAN SEMANGKA
NO ←
CAN YOU SAY:
    MAKAN SEMANGKA ORANG ITU DISINI RUPANYA
NO ←
CAN YOU SAY:
    MAKAN KACANG DISINI RUPANYA ANAQ ITU
NO ←
*PARSING ILLEGALS*
```

```
NEXT:  *TYPE←
  *S1  :=  S2  ITU  S3  S4
   S2  :=  ANAQ
   S2  :=  ORANG
   S2  :=  GURU
   S3  :=  MEMBELI
   S3  :=  MAKAN
   S4  :=  KUWE
   S4  :=  SEMANGKA
  *S5  :=  S2  ITU  S15  S3  S4
  *S6  :=  S15  S12
  *S7  :=  S12  S15
  *S8  :=  S2  ITU DISINI KAMARIN  S3  S4
  *S9  :=  DISINI KAMARIN  S12
  *S10:=  S12  DISINI KAMARIN
  *S11:=  S3  S4  S2  ITU
   S12:=  S11
   S12:=  S1
  *S13:=  S3  S4  S15  S2  ITU
  *S14:=  S3  S4  DISINI KAMARIN  S2  ITU
   S15:=  KAMARIN
   S15:=  RUPANYA

ORANG ITU KAMARIN MAKAN KACANG←
-PARSED OK-

NEXT:  KAMARIN ORANG ITU MAKAN KACANG←
-PARSED OK-

NEXT:  KAMARIN MAKAN KACANG ORANG ITU←
-PARSED OK-

NEXT:  MAKAN KACANG KAMARIN ORANG ITU←
-PARSED OK-

NEXT:  MAKAN KACANG ORANG ITU KAMARIN←
-PARSED OK-

NEXT:  ANAQ ITU MAKAN KACANG KAMARIN←
-PARSED OK-

NEXT:  ANAQ ITU KAMARIN MEMBELI KACANG←
-PARSED OK-

NEXT:  KAMARIN GURU ITU MEMBELI KACANG←
-PARSED OK-
```

```
NEXT: KAMARIN MEMBELI KUWE GURU ITU ←
-PARSED OK-

NEXT: MEMBELI KUWE ORANG ITU KAMARIN ←
-PARSED OK-

NEXT: MAKAN KACANG KAMARIN ANAQ ITU ←
-PARSED OK -

NEXT: KAPAN ORANG ITU MAKAN KUWE ←
CAN YOU SAY:
    MEMBELI SEMANGKA KAPAN GURU ITU
YES ←
CAN YOU SAY:
    GURU ITU KAPAN MEMBELI KACANG
YES ←
CAN YOU SAY:
    MAKAN KACANG GURU ITU KAPAN
YES ←

NEXT: *TYPE ←
 *S1  := S2 ITU S3 S4
  S2  := ANAQ
  S2  := ORANG
  S2  := GURU
  S3  := MEMBELI
  S3  := MAKAN
  S4  := KACANG
  S4  := KUWE
  S4  := SEMANGKA
 *S5  := S2 ITU S15 S3 S4
 *S6  := S15 S12
 *S7  := S12 S15
 *S8  := S2 ITU DISINI KAMARIN S3 S4
 *S9  := DISINI KAMARIN S12
 *S10  := S12 DISINI KAMARIN
 *S11  := S3 S4 S2 ITU
  S12  := S11
  S12  := S1
 *S13  := S3 S4 S15 S2 ITU
 *S14  := S3 S4 DISINI KAMARIN S2 ITU
  S15  := KAPAN
  S15  := KAMARIN
  S15  := RUPANYA

KAPAN MEMBELI KUWE ANAQ ITU ←
-PARSED OK -
```

```
NEXT:  GURU ITU KAPAN MAKAN KACANG ←
-PARSED OK-

NEXT:  MEMBELI KACANG KAPAN ORANGE ITU ←
-PARSED OK-

NEXT:  MEMBELI KUWE ANAQ ITU KAPAN ←
-PARSED OK-
*PARSING ILLEGALS*

NEXT:  GURU ITU MAKAN KUWE KAPAN ←
-PARSED OK-

NEXT:  SIAPA MAKAN KACANG KAMARIN ←
CAN YOU SAY:
    SIAPA MAKAN KACANG KAPAN
NO ←

NEXT:  SIAPA KAMARIN MAKAN KUWE ←
CAN YOU SAY:
    SIAPA RUPANYA MAKAN SEMANGKA
NO ←

NEXT:  *TYPE ←
 *S1  :=  S2  ITU  S3  S4
  S2  :=  ANAQ
  S2  :=  ORANG
  S2  :=  GURU
  S3  :=  MEMBELI
  S3  :=  MAKAN
  S4  :=  KACANG
  S4  :=  KUWE
  S4  :=  SEMANGKA
 *S5  :=  S2  ITU  S15  S3  S4
 *S6  :=  S15  S12
 *S7  :=  S12  S15
 *S8  :=  S2  ITU  DISINI KAMARIN  S3  S4
 *S9  :=  DISINI KAMARIN  S12
 *S10 :=  S12  DISINI KAMARIN
 *S11 :=  S3  S4  S2  ITU
  S12 :=  S11
  S12 :=  S1
 *S13 :=  S3  S4  S15  S2  ITU
 *S14 :=  S3  S4  DISINI KAMARIN  S2  ITU
  S15 :=  KAPAN
  S15 :=  KAMARIN
  S15 :=  RUPANYA
 *S16 :=  SIAPA MAKAN KACANG KAMARIN
 *S17 :=  SIAPA KAMARIN MAKAN KUWE
```

```
KAMARIN SIAPA MEMBELI KUWE ←
CAN YOU SAY:
    KAPAN SIAPA MEMBELI SEMANGKA
NO ←

NEXT:  YANG KAMARIN MEMBELI KACANG SIAPA ←
CAN YOU SAY:
    YANG RUPANYA MAKAN KUWE SIAPA
NO ←

NEXT:  YANG MAKAN KACANG SIAPA KAMARIN ←
CAN YOU SAY:
    YANG MAKAN SEMANGKA SIAPA KAPAN
NO ←
*PARSIN ILLEGALS*

NEXT:  YANG MEMBELI KUWE KAMARIN SIAPA ←
CAN YOU SAY:
    YANG MEMBELI KACANG RUPANYA SIAPA
NO ←

NEXT:  *TYPE ←
 *S1  :=  S2  ITU  S3  S4
  S2  :=  ANAQ
  S2  :=  ORANG
  S2  :=  GURU
  S3  :=  MEMBELI
  S3  :=  MAKAN
  S4  :=  KACANG
  S4  :=  KUWE
  S4  :=  SEMANGKA
 *S5  :=  S2  ITU  S15  S3  S4
 *S6  :=  S15  S12
 *S7  :=  S12  S15
 *S8  :=  S2  ITU  DISINI KAMARIN  S3  S4
 *S9  :=  DISINI KAMARIN  S12
 *S10 :=  S12  DISINI KAMARIN
 *S11 :=  S3  S4  S2  ITU
  S12 :=  S11
  S12 :=  S1
 *S13 :=  S3  S4  S15  S2  ITU
 *S14 :=  S3  S4  DISINI KAMARIN  S2  ITU
  S15 :=  KAPAN
  S15 :=  KAMARIN
  S15 :=  RUPANYA
 *S16 :=  SIAPA MAKAN KACANG KAMARIN
 *S17 :=  SIAPA KAMARIN MAKAN KUWE
 *S18 :=  KAMARIN SIAPA MEMBELI KUWE
 *S19 :=  YANG KAMARIN MEMBELI KACANG SIAPA
 *S20 :=  YANG MAKAN KACANG SIAPA KAMARIN
 *S21 :=  YANG MEMBELI KUWE KAMARIN SIAPA
```

```
SIAPA MEMBELI KACANG KAMARIN ←          [added to the problem]
CAN YOU SAY:
    SIAPA MEMBELI SEMANGKA KAMARIN
YES ←

NEXT: SIAPA MAKAN KUWE KAMARIN ←        [added to the problem]
-PARSED OK-

NEXT: SIAPA MAKAN KACANG KAMARIN ←      [added to the problem]
-PARSED OK-

NEXT: *TYPE ←
 *S1  := S2  ITU  S3  S4
  S2  := ANAQ
  S2  := ORANG
  S2  := GURU
  S3  := MEMBELI
  S3  := MAKAN
  S4  := KACANG
  S4  := KUWE
  S4  := SEMANGKA
 *S5  := S2  ITU  S15  S3  S4
 *S6  := S15  S12
 *S7  := S12  S15
 *S8  := S2  ITU  DISINI KAMARIN  S3  S4
 *S9  := DISINI KAMARIN  S12
 *S10 := S12  DISINI KAMARIN
 *S11 := S3  S4  S2  ITU
  S12  := S11
  S12  := S1
 *S13  := S3  S4  S15  S2  ITU
 *S14  := S3  S4  DISINI KAMARIN  S2  ITU
  S15  := KAPAN
  S15  := KAMARIN
  S15  := RUPANYA
 *S16  := SIAPA MAKAN KACANG KAMARIN
 *S17  := SIAPA KAMARIN MAKAN KUWE
 *S18  := KAMARIN SIAPA MEMBELI KUWE
 *S19  := YANG KAMARIN MEMBELI KACANG SIAPA
 *S20  := YANG MAKAN KACANG SIAPA KAMARIN
 *S21  := YANG MEMBELI KUWE KAMARIN SIAPA
 *S22  := SIAPA  S3  S4  S15
```

3.2.7 Thai

Problem constructed by Peter Lee of the University of Wisconsin Linguistics Department, who also acted as informant. The tones are not indicated as they are not pertinent to this particular problem. Glosses of the sentences have been added to the original teletype printout.

```
BEGIN AUTOLING

NEXT:  KHUN CHOOP PHAK←                     ("You like vegetables")

NEXT:  KHUN CHOOP BURII←                    ("You like cigarettes")

NEXT:  *TYPE←
  *S1  :=  KHUN CHOOP S2
   S2  :=  BURII
   S2  :=  PHAK

KHUN CHOOP KHAJ←                            ("You like eggs")

NEXT:  *TYPE←
  *S1  :=  KHUN CHOOP  S2
   S2  :=  KHAJ
   S2  :=  BURII
   S2  :=  PHAK
KHUN HIW KHAJ←                              ("You are hungry for eggs")
CAN YOU SAY:
     KHUN HIW BURII
NO←

NEXT:  *TYPE←
  *S1  :=  KHUN CHOOP  S2
   S2  :=  KHAJ
   S2  :=  BURII
   S2  :=  PHAK
  *S5  :=  KHUN HIW KHAJ

KHUN HIW KHAAW←                             ("You are hungry for rice")
*PARSING ILLEGALS*
```

```
NEXT: *TYPE ←
  *S1  :=  KHUN CHOOP  S2
   S2  :=  KHAJ
   S2  :=  BURII
   S2  :=  PHAK
  *S5  :=  KHUN HIW  S6
   S6  :=  KHAAW
   S6  :=  KHAJ

KHUN HIW PHAK ←                          ("You are hungry for vegetables")

NEXT: *TYPE ←
  *S1  :=  KHUN CHOOP  S2
   S2  :=  KHAJ
   S2  :=  BURII
   S2  :=  PHAK
  *S5  :=  S10
   S6  :=  PHAK
   S6  :=  KHAAW
   S6  :=  KHAJ
  *S10  :=  KHUN HIW  S6

KHUN HIW NAAM                            ("You are thirsty.")

NEXT: KHUN HIW NAAM MAJ ←                ("Are you thirsty?")
CAN YOU SAY:
    KHUN HIW KHAJ MAJ
YES ←

NEXT: *TYPE ←
  *S1  :=  KHUN CHOOP  S2
   S2  :=  KHAJ
   S2  :=  BURII
   S2  :=  PHAK
  *S5  :=  S10
   S6  :=  NAAM
   S6  :=  PHAK
   S6  :=  KHAAW
   S6  :=  KHAJ
  *S10  :=  KHUN HIW  S6
  *S11  :=  S5  MAJ

KHUN CHOOP KHAJ MAJ ←                    ("Do you like eggs?")
CAN YOU SAY:
    KHUN CHOOP BURII MAJ                 ("Do you like cigarettes?")
YES ←
```

```
NEXT: *TYPE←
  *S1  := KHUN CHOOP  S2
   S2  := KHAJ
   S2  := BURII
   S2  := PHAK
  *S5  := S10
   S6  := NAAM
   S6  := PHAK
   S6  := KHAAW
   S6  := KHAJ
  *S10  := KHUN KIW  S6
  *S11  := S12  MAJ
   S12  := S1
   S12  := S5
```

```
KHUN CHOOP PHAK MAJ -                   ("Do you like vegetables?")
-PARSED OK-
*PARSING ILLEGALS*

NEXT: KHAW MII NAAM MAJ←                ("Does he have water?")
CAN YOU SAY:
    KHAW MII PHAK MAJ                   ("Does he have vegetables?")
YES←

NEXT: PHOM MII PHAK←                    ("I have vegetables.")
CAN YOU SAY:
    PHOM MII KHAJ                       ("I have eggs.")
YES←
```

```
NEXT: *TYPE←
  *S1  := KHUN CHOOP  S2
   S2  := KHAJ
   S2  := BURII
   S2  := PHAK
  *S5  := S10
   S6  := NAAM
   S6  := PHAK
   S6  := KHAAW
   S6  := KHAJ
  *S10  := KHUN HIW  S6
  *S11  := S12  MAJ
   S12  := S1
   S12  := S5
  *S13  := KHAW MII  S6  MAJ
  *S14  := PHOM MII  S6
```

```
KHAW CHOOP KHAJ←                        ("He likes eggs.")
CAN YOU SAY:
    KHAW CHOOP PHAK                     ("He likes vegetables.")
YES←
```

```
CAN YOU SAY:
    KHAW HIW PHAK                         ("He is hungry for vegetables.")

CAN YOU SAY:
    KHUN MII KHAAW MAJ                    ("Do you have rice?")
YES ←

NEXT: *TYPE ←
  *S1  :=  S15  CHOOP S2
   S2  :=  KHAJ
   S2  :=  BURII
   S2  :=  PHAK
  *S5  :=  S10
   S6  :=  NAAM
   S6  :=  PHAK
   S6  :=  KHAAW
   S6  :=  KHAJ
  *S10  :=  S15  HIW  S6
  *S11  :=  S12  MAJ
   S12  :=  S1
   S12  :=  S5
  *S13  :=  S15  MII  S6  MAJ
  *S14  :=  PHOM MII  S6
   S15  :=  KHAW
   S15  :=  KHUN

KHUN MII KHAJ ←                           ("You have egg(s).")
CAN YOU SAY:
    KHUN MII NAAM                         ("You have water.")
YES ←
CAN YOU SAY:
    PHOM CHOOP KHAJ                       ("I like egg(s).")
YES ←
CAN YOU SAY:
    PHOM MII KHAAW MAJ                    ("Do I have rice?")
YES ←
CAN YOU SAY:
    PHOM HIW KHAJ                         ("I'm hungry for egg(s).")
YES ←
CAN YOU SAY:
    KHAW MII KHAAW MAJ                    ("Does he have rice?")
YES ←
CAN YOU SAY:
    KHAW MII PHAK                         ("He has vegetable(s).")
YES ←
CAN YOU SAY:
    PHOM HIW NAAM                         ("I am thirsty.")
YES ←
CAN YOU SAY:
    PHOM HIW KHAAW                        ("I'm hungry (for rice.)")
YES ←
```

```
CAN YOU SAY:
    PHOM MII NAAM MAJ                     ("Do I have water?")
YES ←
CAN YOU SAY:
    PHOM MII KHAJ MAJ                     ("Do I have egg(s)?")
YES ←
CAN YOU SAY:
    KHUN MII KHAJ MAJ                     ("Do you have egg(s)?")
YES ←
CAN YOU SAY:
    KHAW CHOOP KHAJ MAJ                   ("Does he like egg(s)?")
YES ←

NEXT:  *TYPE ←
  *S1  :=  S15  CHOOP  S2
   S2  :=  KHAJ
   S2  :=  BURII
   S2  :=  PHAK
  *S5  :=  S10
   S6  :=  NAAM
   S6  :=  PHAK
   S6  :=  KHAAW
   S6  :=  KHAJ
  *S10  :=  S14
  *S11  :=  S13
   S12  :=  S1
   S12  :=  S5
  *S13  :=  S17  MAJ
  *S14  :=  S15  S16  S6
   S15  :=  PHOM
   S15  :=  KHAW
   S15  :=  KHUN
   S16  :=  MII
   S16  :=  HIW
   S17  :=  S10
   S17  :=  S12

PHOM KIN KHAJ KHRAP ←                     ("I eat egg(s), sir or ma'am.")
CAN YOU SAY:
    PHOM KIN KHAAW KHRAP                  ("I eat rice, sir or ma'am.")
YES ←
*PARSING ILLEGALS*

NEXT:  DICHAN KIN KHAAW  KHA ←            ("I eat rice, sir or ma'am.")
CAN YOU SAY:
    DICHAN KIN KHAJ KHA                   ("I eat egg(s), sir or ma'am.")
YES ←
```

Morphological Note:

PHOM = 'I' (masc. speaker)
DICHAN = 'I' (fem. speaker)
KHRAP = 'sir or ma'am' (masc. speaker)
KHA = 'sir or ma'am' (fem. speaker)

```
NEXT: *TYPE ←
 *S1  := S15  CHOOP  S2
  S2  := KHAJ
  S2  := BURII
  S2  := PHAK
 *S5  := S10
  S6  := NAAM
  S6  := PHAK
  S6  := KHAAW
  S6  := KHAJ
 *S10 := S14
 *S11 := S13
  S12 := S1
  S12 := S5
 *S13 := S17  MAJ
 *S14 := S15  S16  S6
  S15 := PHOM
  S15 := KHAW
  S15 := KHUN
  S16 := MII
  S16 := HIW
  S17 := S10
  S17 := S12
 *S18 := S15  KIN  S6  KHRAP
 *S19 := DICHAN KIN  S6  KHA

DICHAN KAMLANG KIN KHAJ KHA ←            ("I am eating egg(s) sir or ma'am.")
CAN YOU SAY:
    DICHAN KAMLANG KIN KHAAW KHA         ("I am eating rice  sir or ma'am.")
YES ←

NEXT: *TYPE ←
 *S1  := S15  CHOOP  S2
  S2  := KHAJ
  S2  := BURII
  S2  := PHAK
 *S5  := S10
  S6  := NAAM
  S6  := PHAK
  S6  := KHAAW
  S6  := KHAJ
 *S10 := S14
 *S11 := S13
  S12 := S1
  S12 := S5
 *S13 := S17  MAJ
 *S14 := S15  S16  S6
  S15 := PHOM
  S15 := KHAW
  S15 := KHUN
```

```
  S16  :=  MII
  S16  :=  HIW
  S17  :=  S10
  S17  :=  S12
 *S18  :=  S15  KIN  S6  KHRAP
 *S19  :=  DICHAN KIN  S6  KHA
 *S20  :=  DICHAN KAMLANG KIN  S6  KHA

KHAW KIN PHAK ←                          ("He eats vegetable(s).")
CAN YOU SAY:
    KHAW HIW PHAK KHRAP                  ("He is hungry for vegetable(s)
YES ←                                               sir or ma'am.")
CAN YOU SAY:
    DICHAN KIN NAAM KHA                  ("I am drinking, sir or ma'am.")
YES ←
CAN YOU SAY:
    DICHAN KAMLANG HIW NAAM KHA          ("I am (and have been for some
YES ←                                    time) thirsty, sir or ma'am.")

NEXT:  *TYPE ←
 *S1  :=  S15 CHOOP S2
  S2  :=  KHAJ
  S2  :=  BURII
  S2  :=  PHAK
 *S5  :=  S10
  S6  :=  NAAM
  S6  :=  PHAK
  S6  :=  KHAAW
  S6  :=  KHAJ
 *S10  :=  S14
 *S11  :=  S13
  S12  :=  S1
  S12  :=  S5
 *S13  :=  S17  MAJ
 *S14  :=  S15  S16  S6
  S15  :=  PHOM
  S15  :=  KHAW
  S16  :=  KHUN
  S16  :=  KIN
  S16  :=  MII
  S16  :=  HIW
  S17  :=  S10
  S17  :=  S12
 *S18  :=  S15  S16  S6  KHRAP
 *S19  :=  DICHAN  S16  S6  KHA
 *S20  :=  DICHAN KAMLANG  S16  S6  KHA
```

```
DICHAN CA HIW NAAM KHA ←                   ("I (will / am about to) be thirsty, sir.")
CAN YOU SAY:
    DICHAN CA KIN PHAK KHA                 ("I will eat vegetable(s), sir.")
YES ←

NEXT: NAKRIAN CA KIN KLUAJ ←               ("Students will eat banana(s).")
CAN YOU SAY:
    NAKRIAN KAMLANG HIW KLAUJ              ("The students are (and have been / for some time)
YES ←                                      hungry for bananas.
*PARSING ILLEGALS*

NEXT: NAKRIAN KAMLANG KIN KLUAJ ←          ((The) students are eating bananas.)
-PARSED OK-

NEXT: NAKRIAN KAMLANG RIAN NANGSYY ←       ((The) students are studying (books).)
CAN YOU SAY:
    NAKRIAN CA RIAN NANGSYY                ((The) students will study (books).)
YES ←

NEXT: *TYPE ←
 *S1  := S15 CHOOP S2
  S2  := KHAJ
  S2  := BURII
  S2  := PHAK
 *S5  := S10
  S6  := NAAM
  S6  := PHAK
  S6  := KHAAW
  S6  := KHAJ
 *S10 := S14
 *S11 := S13
  S12 := S1
  S12 := S5
 *S13 := S17 MAJ
 *S14 := S15 S16 S6
  S15 := PHOM
  S15 := KHAW
  S15 := KHUN
  S16 := KIN
  S16 := MII
  S16 := HIW
  S17 := S10
  S17 := S12
 *S18 := S15 S16 S6 KHRAP
 *S19 := DICHAN S16 S6 KHA
 *S20 := DICHAN S21 S16 S6 KHA
  S21 := CA
  S21 := KAMLANG
 *S22 := NAKRIAN S21 S16 KLUAJ
 *S23 := NAKRIAN S21 RIAN NANGSYY
```

```
KHRUU KAMLANG SOON NAKRIAN ←          ("Teacher is teaching the students.")
CAN YOU SAY:
    KHRUU CA SOON NAKRIAN             ("The teacher will teach students.")
YES ←

NEXT: KHRUU KAMLANG SOON NANGSYY ←    ((The) teacher is teaching (from
CAN YOU SAY:                            the book).)
    KHRUU CA SOON NANGSYY             ((The) teacher will teach (from the book).)
YES ←
CAN YOU SAY:
    NANGSYY CA RIAN NANGSYY           (The book(s) will study book(s).)
NO ←
CAN YOU SAY:
    NANGSYY KAMLANG KIN KLUAJ         (The book is eating bananas.)
NO ←
CAN YOU SAY:
    NAKRIAN KAMLANG RIAN NAKRIAN      (The student is studying students.)
NO ←

NEXT: *TYPE ←
 *S1  := S15  CHOOP  S2
  S2  := KHAJ
  S2  := BURII
  S2  := PHAK
 *S5  := S10
  S6  := NAAM
  S6  := PHAK
  S6  := KHAAW
  S6  := KHAJ
 *S10  := S14
 *S11  := S13
  S12  := S1
  S12  := S5
 *S13  := S17  MAJ
 *S14  := S15  S16  S6
  S15  := PHOM
  S15  := KHAW
  S15  := KHUN
  S16  := KIN
  S16  := MII
  S16  := HIW
  S17  := S10
  S17  := S12
 *S18  := S15  S16  S6  KHRAP
 *S19  := DICHAN  S16  S6  KHA
 *S20  := DICHAN  S21  S16  S6  KHA
  S21  := CA
  S21  := KAMLANG
```

```
*S22 := NAKRIAN S21 S16 KLUAJ
*S23 := NAKRIAN S21 RIAN NANGSYY
*S24 := KHRUU S21 SOON S25
 S25 := NANGSYY
 S25 := NAKRIAN


KHRUU JAJ MAAK←                    (" The teacher is very big.")

NEXT. KHRUU KHAW KAW MAAK←         ("His teacher is very old.")
CAN YOU SAY:
    KHRUU PHOM KAW MAAK            ("My teacher is very old.")
YES ←
*PARSING ILLEGALS*

NEXT: *TYPE←
 *S1 := S15 CHOOP S2
  S2 := KHAJ
  S2 := BURII
  S2 := PHAK
 *S5 := S10
  S6 := NAAM
  S6 := PHAK
  S6 := KHAAW
  S6 := KHAJ
 *S10 := S14
 *S11 := S13
  S12 := S1
  S12 := S5
 *S13 := S17 MAJ
 *S14 := S15 S16 S6
  S15 := PHOM
  S15 := KHAW
  S15 := KHUN
  S16 := KIN
  S16 := MII
  S16 := HIW
  S17 := S10
  S17 := S12
 *S18 := S15 S16 S6 KHRAP
 *S19 := DICHAN S16 S6 KHA
 *S20 := DICHAN S21 S16 S6 KHA
  S21 := CA
  S21 := KAMLANG
 *S22 := NAKRIAN S21 S16 KLUAJ
 *S23 := NAKRIAN S21 RIAN NANGSYY
 *S24 := KHRUU S21 SOON S25
  S25 := NANGSYY
  S25 := NAKRIAN
 *S26 := KHRUU JAJ MAAK
 *S27 := KHRUU S15 KAW MAAK
```

3.2.8 Mandarian Chinese

Problem constructed by Margaret A. Naeser of the University of Wisconsin Linguistic Department. Ai Chen Ting of the University of Wisconsin East Asian Languages and Literature Department served as informant. Transcription is in the Yale romanization system. Tones are not indicated as they were not pertinent to this particular problem (no minimal pairs used). Glosses have been added to the original output.

```
*SYSTEM REINITIALIZED*

NEXT:  WO SHWO HWA ←                    ("I speak words.")

NEXT:  TA SHWO HWA ←                    ("He speaks words.")

NEXT:  NI SHWO HWA ←                    ("You speak words.")

NEXT:  WO MEN SHWO HWA ←                ("I plural speak words.")
CAN YOU SAY:                               (I plural = we)
    NI MEN SHWO HWA                     ("You plural speak words.")
YES ←

NEXT:  WO YOU SHU ←                     ("I have books.")
CAN YOU SAY:
    TA YOU SHU                          ("He has books.")
YES ←
*PARSING ILLEGALS*

NEXT:  *TYPE ←
  *S1  :=  S2  SHWO HWA
   S2  :=  NI
   S2  :=  TA
   S2  :=  WO
  *S3  :=  S2  MEN  SHWO  HWA
  *S4  :=  S2  YOU  SHU
WO YOU JUNG GWO SHU ←                   ("I have China Land books.")
CAN YOU SAY:                               (China Land = Chinese)
    TA YOU JUNG GWO SHU                 ("He has China Land books.")
YES ←
```

```
NEXT:  WO YOU FA GWO SHU ←            ("I have France Land books.")
CAN YOU SAY:                                    French
    TA YOU FA GWO SHU                  ("He has France Land books.")
YES ←                                           French

NEXT:  WO  SHWO FA GWO HWA ←          ("I speak France Land words.")
CAN YOU SAY:                                    French
    NI SHWO FA GWO HWA                 ("You speak French.")
YES ←

NEXT:  WO YOU YING GWO SHU ←          ("I have England Land books.")
CAN YOU SAY:                                    English
    NI SHWO YING GWO HWA               ("You speak England Land words.")
YES ←                                                English

NEXT:  WO YOU DE GWO SHU ←            ("I have Germany Land books.")
CAN YOU SAY:                                    German
    NI SHWO DE GWO HWA                 ("You speak Germany Land words.")
YES ←                                                German
*PARSING ILLEGALS*


NEXT:  WO KAN SHU ←                   ("I read books.")
CAN YOU SAY:
    TA KAN SHU                         ("He reads books.")
YES ←
CAN YOU SAY:
    TA KAN YING GWO SHU                ("He reads England Land books.")
YES ←                                                English

NEXT:  TA CHANG GER ←                 ("He sings songs.")
CAN YOU SAY:
    NI CHANG GER                       ("You sing songs.")
YES ←

NEXT:  *TYPE ←
 *S1  :=  S2  SHWO HWA
  S2  :=  NI
  S2  :=  TA
  S2  :=  WO
 *S3  :=  S2  MEN SHWO HWA
 *S4  :=  S2  S8  SHU
 *S5  :=  S2  S8  S6  GWO SHU
  S6  :=  DE
  S6  :=  YING
  S6  :=  FA
  S6  :=  JUNG
 *S7  :=  S2  SHWO  S6  GWO HWA
  S8  :=  KAN
  S8  :=  YOU
 *S9  :=  S2  CHANG GER
```

```
TA CHANGE JUNG GWO GER ←          ("He sings China Land songs.")
CAN YOU SAY:                                  Chinese
     NI CHANG JUNG GWO GER        ("You sing Chinese songs.")
YES ←

NEXT:  WO CHR FAN ←               ("I eat food.")
CAN YOU SAY:
     TA CHR FAN                   ("He eats food.")
YES ←

NEXT:  TA CHR JUNG GWO FAN ←      ("He eats China Land food.")
CAN YOU SAY:                                Chinese
     TA CHR YING GWO FAN          ("He eats England Land food.")
YES ←                                       English
*PARSING ILLEGALS*

NEXT:  TA DZWO FAN ←              ("He prepares food.")
CAN YOU SAY:
     WO DZWO FAN                  ("I prepare food.")
YES ←
CAN YOU SAY:
     WO DZWO YING GWO FAN         ("I prepare England Land food.")
YES ←                                         English

NEXT:  TA CHR TSAI ←              ("He eats vegetables.")
CAN YOU SAY:
     TA DZWO TSAI                 ("He prepares vegetables.")
YES ←
CAN YOU SAY:
     NI CHR JUNG GWO TSAI         ("You eat Chinese vegetables.")
YES ←

NEXT:  *TYPE ←
 *S1  :=  S2  SHWO HWA
  S2  :=  NI
  S2  :=  TA
  S2  :=  WO
 *S3  :=  S2  MEN SHWO HWA
 *S4  :=  S2  S8  SHU
 *S5  :=  S2  S8  S6  GWO  SHU
  S6  :=  DE
  S6  :=  YING
  S6  :=  FA
  S6  :=  JUNG
 *S7  :=  S2  SHWO  S6  GWO HWA
  S8  :=  KAN
  S8  :=  YOU
 *S9  :=  S2  CHANG GER
```

```
*S10 := S2 CHANG S6 GWO GER
*S11 := S2 S13 S14
*S12 := S2 S13 S6 GWO S14
 S13 := DZWO
 S13 := CHR
 S14 := TSAI
 S14 := FAN

WO HE CHA ←                           ("I drink tea.")
CAN YOU SAY:
    TA HE CHA                         ("He drinks tea.")
YES ←

NEXT: WO HE JYOU ←                    ("I drink wine.")
CAN YOU SAY:
    NI HE JYOU                        ("You drink wine.")
YES ←

NEXT: WO HE PIJYOU ←                  ("I drink beer.")
*PARSING ILLEGALS*

NEXT: WO HE FA GWO PIJYOU ←           ("I drink French beer.")
CAN YOU SAY:
    NI HE FA GWO PIJYOU               ("You drink French beer.")
YES ←

NEXT: SHEI HE FA GWO PIJYOU ←         ("Who drinks French beer?")
CAN YOU SAY:
    SHEI SHWO HWA                     ("Who speaks words?")
YES ←
CAN YOU SAY:
    SHEI HE JYOU                      ("Who drinks wine?")

*TYPE ←
 *S1 := S2 SHWO HWA
  S2 := SHEI
  S2 := NI
  S2 := TA
  S2 := WO
 *S3 := S2 MEN SHWO HWA
 *S4 := S2 S8 SHU
 *S5 := S2 S8 S6 GWO SHU
  S6 := DE
  S6 := YING
  S6 := FA
  S6 := FA
  S6 := JUNG
 *S7 := S2 SHWO S6 GWO HWA
  S8 := KAN
  S8 := YOU
```

```
 *S9  := S2  CHANG GER
 *S10 := S2  CHANG S6  GWO GER
 *S11 := S2  S13  S14
 *S12 := S2  S13  S6  GWO  S14
  S13 := DZWO
  S13 := CHR
  S14 := TSAI
  S14 := FAN
 *S15 := S2  HE  S16
  S16 := PIJYOU
  S16 := JYOU
  S16 := CHA
 *S17 := S2  HE  S6  GWO  S16
YES ←
CAN YOU SAY:
    SHEI DZWO FAN                    ("Who prepares food?")
YES ←
CAN YOU SAY:
    SHEI DZWO JUNG GWO FAN           ("Who prepares Chinese food?")
YES ←
CAN YOU SAY:
    SHEI CHANG DE GWO GER            ("Who sings German songs?")
YES ←
CAN YOU SAY:
    SHEI CHANG GER                   ("Who sings songs?")
YES ←
CAN YOU SAY
    SHEI YOU SHU                     ("Who has books?")
YES ←
CAN YOU SAY:
    SHEI SHWO FA GWO HWA             ("Who speaks French words?")
YES ←                                               French
CAN YOU SAY:
    SHEI KAN DE GWO SHU              ("Who reads German books?")
YES ←
CAN YOU SAY:
    SHEI MEN SHWO HWA
NO ←
```

```
NEXT: *TYPE ←
  *S1  := S18  SHWO HWA
   S2  := NI
   S2  := TA
   S2  := WO
  *S3  := S2  MEN SHWO HWA
  *S4  := S18  S8  SHU
  *S5  := S18  S8  S6  GWO SHU
   S6  := DE
   S6  := YING
   S6  := FA
   S6  := JUNG
  *S7  := S18  SHWO  S6  GWO HWA
   S8  := KAN
   S8  := YOU
  *S9  := S18  CHANG GER
  *S10  := S18  CHANG  S6  GWO GER
  *S11  := S18  S13  S14
  *S12  := S18  S13  S6  GWO  S14
   S13  := DZWO
   S13  := CHR
   S14  := TSAI
   S14  := FAN
  *S15  := S18  HE  S16
   S16  := PIJYOU
   S16  := JYOU
   S16  := CHA
  *S17  := S18  HE  S6  GWO  S16
   S18  := S2
   S18  := SHEI

WO YAU SHWO HWA ←                       ("I want to speak words.")
                                                  (speak words = speak)
CAN YOU SAY:
    TA YAU SHWO HWA                     ("He wants to speak words.")
                                                  (speak words = speak)
YES ←

NEXT:  WO SYIHWAN SHWO HWA ←            ("I desire to speak words.")
                                                  (speak words = speak)
CAN YOU SAY:
    SHEI SYIHWAN SHWO HWA               ("Who desires to speak words.")
                                                  (speak words = speak)
YES ←

NEXT:  WO KEYI SHWO HWA ←               ("I can speak.")
*PARSING ILLEGALS*

NEXT:  WO DEI SHWO HWA ←                ("I must speak words.")
                                                  (speak words = speak)
```

NEXT: WO DEI SHWO JUNG GWO HWA ← ("I must speak China Land words.")
Chinese
CAN YOU SAY:
WO SYIHWAN SHWO FA GWO HWA ("I desire to speak French.")
YES ←
CAN YOU SAY:
WO SYIHWAN KAN SHU ("I desire to read books.")
YES ←
CAN YOU SAY:
WO YAU CHANG GER ("I want to sing songs.")
YES ←
CAN YOU SAY:
TA SYIHWAN CHANG DE GWO GER ("He desires to sing German songs.")
YES ←
CAN YOU SAY:
TA KEYI DZWO FA GWO FAN ("He can prepare French food.")
YES ←
CAN YOU SAY:
NI DZWO TSAI ("You prepare vegetables.")
YES ←
CAN YOU SAY:
NI SYIHWAN HE CHA ("You desire to drink tea.")
YES ←
CAN YOU SAY:
NI HE FA GWO CHA ("You drink French tea.")
YES ←
CAN YOU SAY:
NI DEI SHWO HWA ("You must speak.")
YES ←
CAN YOU SAY:
TA DEI KEYI SHWO HWA
NO ←
CAN YOU SAY:
NI YAU YOU JUNG GWO SHU ("You want to have Chinese books.")
YES ←
CAN YOU SAY:
NI SYIHWAN SHWO HWA ("You desire to speak.")
YES ←
CAN YOU SAY:
NI SYIHWAN KAN SHU ("You desire to read books.")
YES ←
CAN YOU SAY:
SHEI KEYI CHANG GER ("Who can sing songs?")
YES ←
CAN YOU SAY:
SHEI CHANG JUNG GWO GER ("Who can sing Chinese songs?")
YES ←
CAN YOU SAY:
TA CHR FA GWO FAN ("He eats French food.")
YES ←

CAN YOU SAY:
    WO KEYI DZWO TSAI ("I can prepare vegetables.")
YES ←
CAN YOU SAY:
    SHEI HE CHA ("Who drinks tea ?")
YES ←
CAN YOU SAY:
    TA DEI HE YING GWO CHA ("He must drink English tea.")
YES ←
CAN YOU SAY:
    TA DEI SHWO HWA ("He must speak.")
YES ←
CAN YOU SAY:
    NI YAU SHWO HWA ("You want to speak.")
YES ←
CAN YOU SAY:
    TA YAU YOU FA GWO SHU ("He wants to have French books.")
YES ←
CAN YOU SAY:
    SHEI SHWO YING GWO HWA ("Who speaks English?")
YES ←

NEXT: WO MEN DEI SHWO HWA ← ("We must speak.")
CAN YOU SAY:
    SHEI MEN SYIHWAN SHWO HWA
NO ←

NEXT:
WO MEN DEI DAU JER LAI SHWO HWA ← ("We must come here to speak.")
CAN YOU SAY:
    WO KEYI MEN DEI DAU JER LAI SHWO HWA
NO ←

NEXT: SHEI DEI DAU JER LAI SHWO HWA ← ("Who must come here to speak?")
CAN YOU SAY:
    WO YAU DAU JER LAI SHWO HWA ("I want to come here to speak.")
YES ←
PARSING ILLEGALS

NEXT: WO YAU DAU JYA LAI SHWO HWA ← ("I want to come home to speak.")
CAN YOU SAY:
    SHEI YAU DAU JYA LAI SHWO HWA ("Who wants to come home to speak.")
YES ←

NEXT: WO YAU DAU SYWESYAU LAI SHWO HWA ← ("I want to come to school to speak.")
CAN YOU SAY:
    NI SYIHWAN YAU DAU SYWESYAU LAI SHWO HWA
NO ←
CAN YOU SAY:
    NI KEYI YAU DAU SYWESYAU LAI SHWO HWA
NO ←

```
CAN YOU SAY:
    TA YAU DAU SYWESYAU LAI SHWO HWA        ("He wants to come to school
YES ←                                         to speak.")

NEXT: WO YAU DAU FANGDZ LAI SHWO HWA ←  ("I want to come to the house
                                          to speak.")
NEXT: *TYPE ←
 *S1  := S22 SHWO HWA
  S2  := NI
  S2  := TA
  S2  := WO
 *S3  := S2 MEN SHWO HWA
 *S4  := S22 S8 SHU
 *S5  := S22 S8 S6 GWO SHU
  S6  := DE
  S6  := YING
  S6  := FA
  S6  := JUNG
 *S7  := S22 SHWO S6 GWO HWA
  S8  := KAN
  S8  := YOU
 *S9  := S22 CHANG GER
 *S10  := S22 CHANG S6 GWO GER
 *S11  := S22 S13 S14
 *S12  := S22 S13 S6 GWO S14
  S13  := DZWO
  S13  := CHR
  S14  := TSAI
  S14  := FAN
 *S15  := S22 HE S16
  S16  := PIJYOU
  S16  := JYOU
  S16  := CHA
 *S17  := S22 HE S6 GWO S16
  S18  := S2
  S18  := SHEI
 *S19  := S1
  S20  := DEI
  S20  := KEYI
  S20  := SYIHWAN
  S20  := YAU
  S21  := S2 S20
  S21  := S18
  S22  := S21
  S22  := S18 S20
 *S23  := WO MEN DEI SHWO HWA
 *S24  := WO MEN DEI DAU JER LAI SHWO HWA
 *S25  := S22 DAU JER LAI SHWO HWA
 *S26  := S22 YAU DAU S29 LAI SHWO HWA
  S29  := FANGDZ
  S29  := SYWESYAU
  S29  := JYA
```

ADDENDUM to page 65

The sensitivity problem described on page 65 is now corrected via random checking. The error might still occur on a random basis if S_2 has more than two members.

The improvement is reflected in the examples of Section 3, e.g. page 19.

3.3 Planned Improvements in the Phrase Structure Learning Program.

The system is very sensitive to the order in which inputs are presented. At the moment the class splitting heuristic 5 does not apply under certain input sequence circumstances. For example, given the inputs:

the girl is tall

a girl is tall,

yielding the grammar:

$*S_1 \rightarrow S_2$ girl is tall

$S_2 \rightarrow$ the

$S_2 \rightarrow$ a

If the next input is:

the girl s are tall

The system will add the rule:

$*S_3 \rightarrow S_2$ girl s are tall

<u>without</u> checking to see if

a girl s are tall

is legal. This test would have been made if the last input had been the first input. As a result the system maintains an illegal rule which may not be corrected for a very long time, if ever. If an "a" should occur with a plural noun in a later test for another rule and be rejected by the informant, the system will merely reject the rule currently under test. In such circumstance 3 different things may happen. The system may recycle and correct the error in a later learning run; the system may recycle recursively to a depth 3 and quit; or, more frequently, learn a very complicated grammar which is capable of parsing all the inputs from the informant, but which, from a generative point of view, still contains illegal rules.

This flaw can probably be corrected by the following heuristic procedure which will be added to the system: if a given top node string derived from an informant input sentence contains any nonterminals that are classes of morphemes, generate test sentences through the top node string selecting the other members of each morpheme class, and apply the class splitting heuristic 5, each time the informant rejects a test case.

Another method for obtaining a cleaner grammar would be to treat the right half of each rule in which a valid substitution has been made as an informant input, and subject it to the input rule coining heuristics. This improvement will be attempted, but the refining may come only at periodic intervals rather than after evey rule change because of computation time problems.

The major improvement of the system will come from converting it to a context sensitive phrase structure learning system. The data structures already have appropriate links for associating context with individuals rules. Such an improvement will also require the construction of a context sensitive multi-path parser.

The heuristics for context sensitive learning will be supplemental to, and on the pattern of those for context free learning. Basically, if a context free rule is to be rejected on the basis of an informant's rejection of a text sentence, the system will attempt to reformulate the rule with a context restriction.

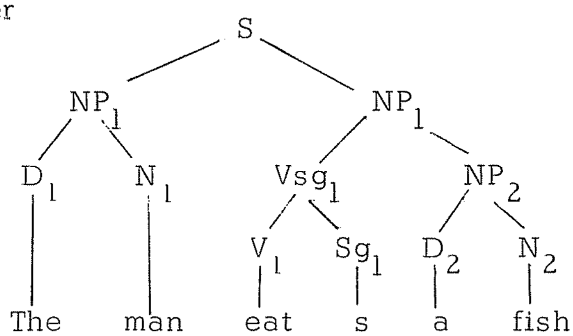
The formulation of the context restrictions themselves will initially be rather specific, but may grow in generality of statement via application of the heuristics already used in learning the context free rules.

4.0 Transformation Learning Program.

Our work on a transformation learning program yielded two learning methods, bottom-to-top and top-to-bottom. The top-to-bottom method, which is entirely the work of William Fabens, was the one actually implemented. The bottom-to-top method, however, lends itself more readily to rule modification via informant interaction, and will be implemented and used in futuer versions of the system. Both methods require as input first a P-marker, (that is, an input sentence with a tree structure derived either from parsing or generation); and second, the sentence (without a tree) into which the first sentence is to be transformed. In each method the sentences may be in different languages. Both methods yield learning of bilingual transformations. The learning of monolingual transformations is a special case.

4.1 Bottom-to-Top Learning.

This method yields learning of the least general case first, and gradually increases the level of generality acceptable to an informant. Consider an input P-Marker



and a "target" sentence (the desired transform)

a fish is eat en by the man

Increasingly complex transformations are coined by climbing up the trees of both sentences: in the case of the first, the given P-marker; in the case of the second, the implicit local tree structures existing in common with the first.

Accordingly, the lowest level transformation that could be coined is:

T_1: the man eat s a fish $\Longrightarrow$ a fish is eat en by a man .

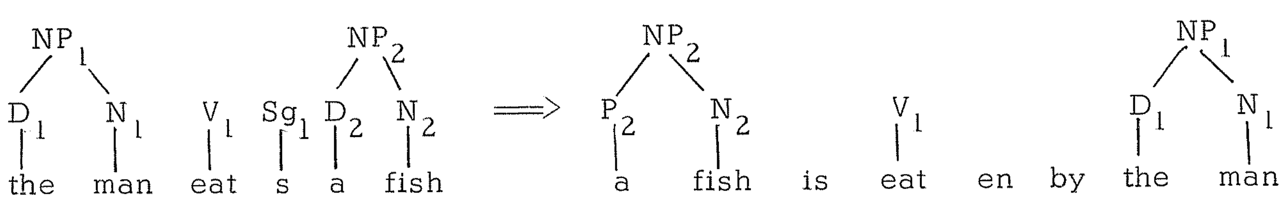
A somewhat higher level transformation would incorporate what is common to both inputs one level above the terminal string level. e.g.

T_2: D_1 N_1 V_1 Sg_1 D_2 N_2 $\Longrightarrow$ D_2 N_2 is V_1 en by D_1 N_1

The assignment of higher level nodes to the elements on the right was determined by their existence in the P-marker of the first sentence. Test sentences generated via this transformation are offered an informant. Should he reject any, the level of generality of the transformation is decreased. For example, should this transformation not work if a different member of class V_1 is used, the transformation would be reformulated:

T_3: D_1 N_1 eat Sg_1 D_2 N_2 $\Longrightarrow$ D_2 N_2 is eat en by D_1 N_1

And a special verb class containing "eat" might be formulated at a later stage. Assuming, however that T_2 is accepted, the program would search for additional higher level common units, e.g.



This suggests the transformation:

T_4: $NP_1 \quad V_1 \quad Sg_1 \quad NP_2 \implies NP_2 \text{ is } V_1 \text{ en by } NP_1$

There would, of course, be intermediate stages before T_4 is obtained, e.g. the coining of:

$$NP_1 \quad V_1 \quad Sg_1 \quad D_2 \quad N_2 \implies D_2 \quad N_2 \text{ is } V_1 \text{ en by } NP_1$$

If the informant accepts the test cases for T_4, it is accepted as the most general transformation to be learned, as no more common elements can be found between the explicit tree of the source and the implicit tree of the transform string. Should the informant reject test sentences derived from T_4 the system would again retreat in level of generality.

The transformation might also be subject to change and updating because of changes in the nature of the phrase structure grammar.

4.2 Top-to-Bottom Transformation Learning.

As indicated, this program is implemented and working. (Some bugs still exist in it, but these did not seriously interfere with the test examples presented here.) As indicated the logic and program are due to William Fabens.

4.2.1 Program Logic.

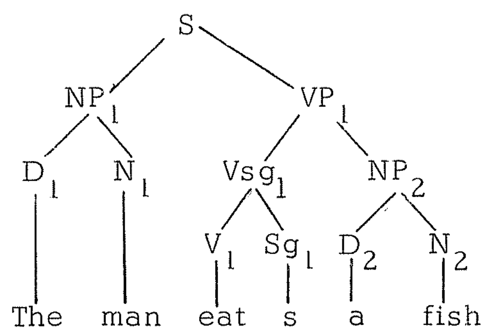
The output of the learning process is a list of ordered transformations, which operate from the top of the tree downwards. This block of transformations is also restated by the program (through substitution of terms) as a single transformation identical with the type derived by the bottom-to-top learning method.

Let us consider the learning process with the same example as in section 4.1. The input P-marker is "the man eat s a fish" plus its attendant tree structure, and again the transform: "a fish is eat en by the man". The system asks the translation of each morpheme in the input sequence. The correct reply is the equivalent morpheme in the transform string. (This is a substitute for dictionary lookup in the general case of bilingual transformation learning.) If there are no equivalents, the informant replies 'NONE'. If two morphemes are identical in the transform, the informant subscripts his replies to identify the relative positions of each like morpheme. (In the case of bilingual learning, this is in no sense word for word translation. Rather it is a method for locating equivalent phrase units).

Each morpheme in the transform sentence is placed at the bottom of its own push down stack. Above each morpheme in its stack is a list, in sequence, of the tree nodes on the path leading from it to the S node in the original input P-marker. Thus

a	D_2	NP_2	VP_1
fish	N_2	NP_2	VP_1
is			
eat	V_1	Vsg_1	VP_1
en			
by			
the	D_1	NP_1	
man	N_1	NP_1	

from



Next the tops of the stacks are scanned. If the same node occurs discontinuously on the top level, it is added to the tops of the intervening stacks. Accordingly, VP_1 is added to the top of the stack above 'is' :

a	D_2	NP_2	VP_1
fish	N_2	NP_2	VP_1
is	VP_1		
eat	V	Vsg_1	VP_1

.
.
.

If one or more morphemes are at the top of an adjacent stack (immediate adjacency in a forward linear scan) the node at the top of the stack of the last sequence of like top nodes is added to the tops of the morpheme stacks. In this case the result is that VP_1 is now added to the stacks containing 'en' and 'by' . (The ordering is admittedly arbitrary: one may ask why NP_1 was not added instead of VP_1). The result is:

a	D_2	NP_2	VP_1
fish	N_2	NP_2	VP_1
is	VP_1		
eat	V_1	Vsg_1	VP_1
en	VP_1		
by	VP_1		
the	D_1	NP_1	
man	N_1	NP_1	

The system then tabulates all possible tree branch transformations starting at the top. The source tree yields the left hand formulation of each transformation. The left half of the first of the series of ordered transformations is:

$$S(NP_1 \quad VP_1) \Longrightarrow$$

which may be interpreted as 'NP_1 and VP_1' dominated by an S node. An S node was implicity at the top of the transform pushdown stacks. Accordingly, the right half of the transformation is formulated as S dominating whatever is currently at the top of the pushdown stacks, where adjacent strings of like nodes are treated as a single node:

T_a: $S(NP_1 \quad VP_1) \Longrightarrow \quad S(VP_1 \quad NP_1)$

For the next step, the top nodes of the pushdown stacks above the transform tree are deleted and the node redistribution process described above is repeated except that morphemes at the top of the stacks no longer receive adjacent nodes. In this case no new nodes are added to any of the stacks.

a	D_2	NP_2
fish	N_2	NP_2
is		
eat	V_1	Vsg_1
en		
by		
the	D_1	
man	N_1	

The strings of VP_1's and NP_1's have been removed from the tops of the pushdown stacks. The right half of the next transformation is formed from what was under the VP_1 nodes:

$$\Longrightarrow \quad VP_1 \ (NP_2 \quad is \quad Vsg_1 \quad en \quad by)$$

The left half is derived from what is under the VP_1 node in the input tree; yielding:

T_b: $\quad VP_1 \ (Vsg_1 \quad NP_2) \quad \Longrightarrow \quad VP_1 \ (NP_2 \quad is \quad Vsg_1 \quad en \quad by)$

Similarly, what is beneath the deleted NP_1 in the source tree and the stack form the next transformation:

$$NP_1 \ (D_1 \quad N_1) \Longrightarrow \quad NP_1 \ (D_1 \quad N_1)$$

which is an identity transformation (a consequence of the method). Identity transformations, although calculated, are surpressed in the teletype output. Again the program strips the top nodes from the push down stacks:

a $\quad D_2$

fish $\quad N_2$

(is) (removed from the stack)

eat $\quad V_1$

(en) (removed from the stack)

(by) (removed from the stack)

the

man

The removed NP_2 yields an identity transformation which is disregarded:

$$NP_2\ (D_2\quad N_2) \implies NP_2\ (D_2\quad N_2)$$

However the removal of the Vsg_1 unit yields a non-trivial transformation:

$$T_c:\quad Vsg_1\ (V\quad Sg) \implies Vsg_1(V)$$

The deleted D_1 and N_1 nodes again yield trivial identity transformations:

$$D_1\ (\text{the}) \implies D_1\ (\text{the})$$

$$N_1\ (\text{man}) \implies N_1\ (\text{man})$$

Repeating the process, the identity transformations

$$D_2(\text{a}) \implies D_2\ (\text{a})$$

$$N_2\ (\text{fish}) \implies N_2\ (\text{fish})$$

are also coined.

Repeated substitutions in the battery of ordered transformations yield a rule identical to that derived by the bottom-to-top learning method:

$$T_a:\quad S(NP_1\quad VP_1) \implies S(VP_1\quad NP_1)$$

yields via substitution of the terms in T_b for VP_1 :

$$S(NP_1\quad Vsg_1\quad NP_2) \implies S(NP_2\ \text{is}\ Vsg_1\ \text{en by}\ NP_1)$$

followed by substitution for Vsg_1 from transformation T_c :

$$S(NP_1\quad V_1\quad Sg_1\quad NP_2) \implies S(NP_2\ \text{is}\ V_1\ \text{en by}\ NP_1)$$

which is identical to the transformation T_4 derived in section 4.1.

4.2.2 Features of the Program.

The system begins its input of the starting P-marker by outputting a message:

SENT :=

The human replies with the nodes deriving from SENT. The nodes must be bracketed by quotes and separated by commas. For example the human might reply:

"NP" , "VP"

followed by an arrow. The system will then query the expansion of each node, descending the left most branch of the tree first. E.g. the next query would then be:

. NP :=

The dot indicates the relative depth in the tree. If the last element was a terminal, the arrow key alone is punched. At the end of the input of a tree the system asks:

DO YOU WANT ANOTHER PARSE ?

If the answer is YES, (in case of error) the program disregards the last inputted tree and offers one the chance of another tree input. If the answer is NO , the system outputs the message:

PLEASE TRANSLATE

followed by the terminal elements of the tree in sequential order. The human must then input the desired transform string, with each morpheme in quotes and separated by commas. The system will then ask for morpheme equivalents. If two or more morphemes are identical in either string the system assigns or interprets subscripts where appropriate.

After the transformation is learned, the system prints the rules, both in the ordered form and in the expanded, single expression notation. The program also uses '+' to indicate two units descended from the same node, and '-' to indicate more remote ancestry. The program then outputs a message:

WANT TO REAPPLY LAST TRANSFORMATION?

A YES answer means that the next time a tree exhibiting nodes in the domain of the transformation is inputted, the program will apply the transformation.

If the new P-marker is more complex then that of the learning input (but with nodes still in the domain of the transformation), the system will still correctly apply the rule (except in case of extreme complexity because of an uncorrected bug in the program).

Section 4.2.2 lists a copy of the teletype output with keypunch errors and extraneous system messages excluded) for learning of an active-passive transformation, and its application to a more complex sentence with different vocabulary. Section 4.2.3 lists a similar teletype transcript for the learning of an English to German bilingual transformation. (The print out of the dominating node on the right half of each transformation is surpressed on the teletype output.)

4.2.3 Learning and Application of an Active-Passive Transformation.

```
SENT := "NP", "VP" ←
. NP := "D", "NP" ←
. . D := ←
. . NP := ←
. VP := ←
DO YOU WANT ANOTHER PARSE? YES ←
SENT := "D" ←
. D := ←
DO YOU WANT ANOTHER PARSE? YES ←
SENT := "NP", "VP" ←
. NP := "JOHN" ←
. . JOHN := ←
. VP := "VP", "NP" ←
. . VP := "VSTMA", "SGA" ←
. . . VSTMA := "WANT" ←
. . . . WANT := ←
. . . SGA := "S" ←
. . . . S := ←
. . NP := "MARY" ←
. . . MARY := ←
DO YOU WANT ANOTHER PARSE? NO ←
PLEASE TRANSLATE
JOHN WANT S MARY
"MARY", "IS", "WANT", "ED", "BY", "JOHN" ←
```

```
C( JOHN ) := "JOHN" ←

C( WANT ) := "WANT" ←

C( S ) := "NONE" ←

C( MARY ) := "MARY" ←

SENT ( NP , VP ) ::= VP NP

VP ( VP(1) , NP(1) )::= NP(1) IS VP(1) ED BY

VP(1) ( VSTMA , SGA )::= VSTMA

OR: NP - VSTMA + SGA - NP(1) - := NP(1) + IS - VSTMA - ED + BY - NP

TRANSLATION IS...
MARY IS WANT ED BY JOHN

WANT TO REAPPLY LAST TRANSFORMATION ? YES ←

SENT := "NP", "VP" ←

. NP := "D", "NP" ←

. . D := "THE" ←

. . . THE := ←

. . NP := "NP", "MOD" ←

. . . NP := "MAN" ←

. . . . MAN := ←

. . . MOD := "PREP", "NP" ←

. . . . PREP := "IN" ←

. . . . . IN := ←

. . . . NP := "D", "NP" ←

. . . . . D := "THE" ←

. . . . . . THE := ←

. . . . . NP := "PARK" ←

. . . . . . PARK := ←
```

```
. VP := "VP", "VP" ←
. . VP := "VSTMA", "SGA" ←
. . . VSTMA := "KISS" ←
. . . . KISS := ←
. . . SGA := "ES" ←
. . . . ES := ←
. . NP := "D", "NP" ←
. . . D := "A" ←
. . . . A := ←
. . . NP := "ADJ", "NP"
. . . . ADJ := "PRETTY" ←
. . . . . PRETTY := ←
. . . . NP := "GIRL" ←
. . . . . GIRL := ←
DO YOU WANT ANOTHER PARSE? NO ←
T( A ) := "A" ←
T( PRETTY ) := "PRETTY" ←
T( GIRL ) := "GIRL" ←
T( KISS ) := "KISS" ←
T( THE ):= "THE" ←
T( MAN ) := "MAN" ←
T( IN ):= "IN" ←
T( THE(1) ) := "THE" ←
T( PARK ):= "PARK" ←
TRANSLATION IS...
A PRETTY GIRL IS KISS ED BY THE MAN IN THE PARK
```

4.2.4 Learning a Bilingual Transformation.

```
SENT :=
. NP :=
. . PN :=
. . . HE :=
. VP :=
. . VP :=
. . . IS :=
. . NP :=
. . . NP :=
. . . . D :=
. . . . . A :=
. . . . NP :=
. . . . . MAN :=
. . . RC := "RP",
. . . . RP :=
. . . . . WHO :=
. . . . VP :=
. . . . . V :=
. . . . . . V :=
. . . . . . . LIVE :=
. . . . . . SG :=
. . . . . . . S :=
. . . . . MOD :=
. . . . . . PREP :=
```

```
. . . . . . . IN :=
. . . . . . NP :=
. . . . . . . D :=
. . . . . . . . THE :=
. . . . . . . NP :=
. . . . . . . . CITY :=
DO YOU WANT ANOTHER PARSE?
PLEASE TRANSLATE
HE IS A MAN WHO LIVE S IN THE CITY
C( HE ):=
C( IS ):=
C( A ):=
C( MAN ):=
C( WHO ):=
C( LIVE ):=
C( S ):=
C( IN ):=
C( THE ):=
C( CITY ):=
SENT ( NP , VP ):= NP VP
VP ( VP(1) , NP(1) ):= VP(1) NP(1)
NP(1) ( NP(2) , RC ):= NP(2)
NP(2) ( D , NP(3) ):= D RC -ENDER NP(3)
RC ( RP , VP(2) ):= VP(2)
VP(2) ( V , MOD ):= MOD V
```

MOD (PREP , NP(4)):= PREP NP(4)

NP(4) (D(1) . NP(5)):= D(1) NP(5)

V (V(1) , SG):= V(1)

[Program error: identity transformations should not have been printed.]

OR: NP - VP(1) - D + NP(3) - RP - V(1) + SG - PREP - D(1) + NP(5) -
:= NP - VP(1) - D - PREP - D(1) + NP(5) - V(1) - ENDER + NP(3) -

[A compounded error: the collapsed transformation used the identity transformations in the expansion. With corrections, it should read

NP - VP(1) - D + NP(3) - RP - V(1) + SG - MOD :=
NP - VP(1) - D - MOD - V(1) - ENDER + NP(3)]

5.0 Proving the Linguist Superfluous.

In terms of speed in producing a grammar, the phrase structure learning component of the AUTOLING system seems to have an advantage over the human linguist. For example, the Indonesian problem of Section 3.2.6 required about 45 minutes of the "informant's" time at the teletype, including the usage of less than 4 minutes of computer central processor time (in a time sharing environment involving relatively light demands by other users). AUTOLING's time advantage over a human analyst appears to increase with the size of the corpus, but precise tests have not been carried out.

With respect to completeness and quality, the existing AUTOLING system is not yet ready to replace human linguists. But gradual improvements are inevitable, and eventually the role of the human fieldworker may be challenged seriously, particularly if the state of the art ever permits the incorporation of adequate vocal communication between informant and computer.

The proof of the adequacy of the machine as linguist might eventually be demonstrated through a variant of the Turing test for artificial intelligence [19]. Let 5 or 10 human linguists each spend a set amount of time individually working with the same informant. Let the machine linguist do the same

Then let the grammars produced by <u>all</u> participants be presented, annonymously, to another group of linguists who must attempt to spot the machine's grammar. If the machine linguist is not determined as such with statistically significant frequency, one may assume it is at least as good a fieldworker as the weakest human analyst in the test group.

While such a success might make 'data collecting' linguists superfluous, it should free most for work in linguistic theory. Of course by that time the computer will have become an essential tool of the theorist, not just for data collection and analysis, but as a means of checking the implications of theoretical formulations and models.

REFERENCES

1. Chomsky, N. Aspects of the Theory of Syntax. MIT Press, Cambridge 1962.

2. Garvin, Paul L. Automatic Linguistic Analysis - a heuristic problem, in 1961 International Conference on Machine Translation of Languages and Applied Language Analysis, Her Majesty's Stationary Office, London, 1962.

3. Garvin, Paul L. The Automation of Discovery Procedure in Linguistics. Language, Vol. 43, No. 1, March 1967. (Presented at Linguistic Society of America Meeting in 1965).

4. Gold, E. M. Language Identification in the Limit. RAND memorandum RM-4136-PR, July 1964, RAND Corporation, Santa Monica. Also, published in a revised version in Information and Control, Vol. 10, No. 5, May 1967.

5. Harris, Z. S. From Morpheme to Utterance. Language, 22, 161-83, 1946.

6. Harris, Z. S. Methods in Structural Linguistics. University of Chicago, 1951.

7. Katz, J. & Fodor, J. The Structure of a Semantic Theory. Language. Vol. 39, No. 2, April-June, 1963.

8. Klein, S. Some Components of a Program for Dynamic Modelling of Historical Change in Language Using Monte Carlo Techniques. Paper No. 14 of Preprints of Invited Papers for 1965 International Conference on Computational Linguistics. Associational Conference on Computational Linguistics. (Also will appear in Russian, in a book on Language Contact, edited by V. Rosentsveig, Moscow).

9. Klein, S. Historical Change in Language Using Monte Carlo Techniques Mechanical Translation, Vol. 9, Nos. 3 & 4, Sept. & Dec. 1966. (Publ. May 1967). (Also will be reprinted in a source language version of immediately preceeding mentioned book edited by V. Rosentsveig, as a replacement for the earlier paper. Mouton, The Hague.)

10. Klein, S. Current Research in the Computer Simulation of Historical Change in Language. In press, Proceedings of the Xth International Congress of Linguists, September 1967, Bucharest.

11. Klein, Davis, Fabens, Herriot, Katke, Kuppin & Towster, AUTOLING: An Automated Linguistic Fieldworker, Second International Conference on Computational Linguistics, August 1967, Grenoble.

12. Knowlton, K. Sentence Parsing with a Self-organizing Heuristic Program. Ph.D. thesis, MIT, Cambridge, August, 1962.

13. Koutsoudas, A. Writing Transformational Grammars: An Introduction. McGraw-Hill, New York, 1966.

14. McConlogue, K. & Simmons, R. F., "Analysing English Syntax with a Pattern-Learning Parser." Communications of the ACM, Vol. 8, No. 11, November, 1965.

15. Nida, E. A. Morphology, The Descriptive Analysis of Words, 2nd Edition. University of Michigan Press, Ann Arbor, 1949.

16. Shamir, E. A Remark on Discovery Algorithms for Grammars, Information and Control, Vol. 5, September, 1962.

17. Siklossy, L. Natural Language Learning by Computer. Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, 1968.

18. Solomonoff, R. A New Method for Discovering the Grammars of Phrase Structure Languages. Information Processing, Proceedings of the International Conference on Information Processing, UNESCO, 1959.

19. Turing, A. M. Computing Machinery and Intelligence. Mind, 59: 433-460, October, 1950.

20. Uhr, L. "Pattern-string Learning Programs." Behaviorial Science, Vol. 9, No. 3, July 1964.

21. Wells, R. Immediate Constitutents. Language Vol. 23, 81-117, 1947.