Computer Sciences Department
University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

AN INTERACTIVE, HEURISTIC PROGRAM FOR LEARNING TRANSFORMATIONAL GRAMMARS

by

Sheldon Klein and Michael A. Kuppin

Technical Report #97

August 1970

A portion of this paper was presented at the 1970 Summer Meeting of the Linguistic Society of America, July 24-26, Columbus, Ohio, under the title, "A Transformational Grammar of English Discovered by a Computer."

ABSTRACT

An Interactive, Heuristic Program for Learning Transformational Grammars

by

Sheldon Klein and Michael A. Kuppin

An interactive system for learning transformational grammars of natural languages has been programmed in ALGOL and is operational on a B5500 time-sharing system. The program is part of an "automated linguistic fieldworker" system intended to duplicate the functions of a human linguist in working with a live informant.

Using a battery of analytic heuristics, the program derives grammars containing context-free phrase structure rules and transformations via interaction with an informant who, minimally, is expected to be bilingual in English and the language under analysis. The informant must input sentences in the subject language, indicating spaces between morphological units, and may be required to answer questions about the validity of the program's test productions.

A skilled linguist may choose to use the program as a partner in fieldwork. He may inspect the grammar at any time and guide the analysis by inserting or deleting rules of his choice.

An Interactive, Heuristic Program for Learning Transformational Grammar*

by

Sheldon Klein and Michael A. Kuppin

1. Introduction

1.1 Purpose

David G. Hays in his introductory textbook on computational linguistics makes the following distinction among degrees of computer participation in linguistic research (11):

> "The lowest level uses the computer merely as a compiler of data; programs to make concordances or maintain dictionaries are effective, but merely clearical, aids. The middle degree is computer testing of information gained in other ways, as when a machine-translation program and table of translation equivalences is tried out on new text. The highest degree is reached when the computer program actually embodies the linguist's analytic ideas; a device that could be parachuted into a jungle to learn an unknown language without supervision would be participating in linguistic research to the highest degree."

It is toward this highest degree of participation that our research is directed.

The program described in this paper is part of AUTOLING, an automated linguistic fieldworker intended to duplicate the functions of a human fieldworker in learning a grammar through interaction with a live human informant. The goals for the total system include integration of the analysis of phonology and morphology. In this paper, however, we report on the component that learns context-free grammar rules and transformations.

*This research has been sponsored by the National Science Foundation, grant No. G-52595 and the Wisconsin Alumni Research Foundation.

1.2 Methodological Significance

An algorithmic discovery procedure is one that guarantees success; an heuristic discovery procedure is one that may work but provides no *a priori* guarantee of success. The validation of algorithmic discovery procedures should be deductive in nature. Heuristic methods can only find their validation empirically, through real world testing.

Many linguists, including Chomsky and Lamb, feel that their theoretical models are independent of grammar discovery procedures, particularly because the existence of such procedures is open to serious question. A strong empiricist (logical postivist, operationalist, etc.) might argue that linguistic theories have no meaning unless the means of discovering the required grammars can be formulated. Archetypical of the situation is the contrast between Chomsky's embrace of the rationalism of Liebniz and Descartes as a means of defending empirically untestable hypotheses (1) and Charles Hockett's renunciation of all linguistic theory since Bloomfield (12), coupled with a labelling of Chomsky as a "neo-medieval philosopher" (13).

While numerous linguists have given informal and semi-formal directions for writing grammars, such directions have not led to any satisfactory formal or empirical demonstrations of the efficacy of their methods. Creation of a system like AUTOLING, containing a formal statement of discovery method that is subject to rapid, direct, empirical verification, establishes an intermediate theoretical battleground where rationalist and empiricist may find common criteria for the validation of theoretical models.

1.3 Brief History of Automated Grammar Discovery Research

The history of discovery procedures in language analysis is as old as the history of linguistic fieldwork. However, one of the earliest attempts at providing formal methods based on distributional criteria was made by Zellig Harris, 1946 (9) and 1951 (10). Rulon Wells was also a pioneer in this methodology, 1947 (24). Both men suggested analytic techniques for deriving grammars that can be viewed as phrase-structure in nature. The 1946 paper of Harris also marked the introduction of linguistic transformations into modern linguistic theory.

Solomonoff suggested the use of a computer for deriving phrase-structure grammar from texts in 1959 (22). His analytic methods were anticipated by Harris, although not as concisely. Paul Garvin suggested the use of heuristics in the choice of distributional tests on text data in 1961 (5). In 1964, E. Mark Gold indicated that a discovery algorithm for learning context-free grammar might be possible if informant interaction were used (8). Garvin has discussed the use of heuristics in automated informant fieldwork, 1965 (6), and has worked on an interactive morphological analysis program, 1967-1970 (7). J. A. Feldman, 1967 (2), 1969 (3), and J. F. Horning, 1969 (14), are among others who have discussed various methods for grammar inference.

There are a number of programs in existence involving some form of language learning. K. C. Knowlton, 1962 (18), and Mc Conlogue and Simmons, 1965 (19), produced programs that enable a parser to learn correct parses. These programs required the informant to supply corrective information in either a phrase-structure or dependency notation. L. Uhr, 1965 (23), and L. Siklossy, 1968 (21), produced programs that learn finite state grammars and involve some morphological analysis. Feldman, et. al., 1969 (4), produced a program that learns context-free grammars.

Descriptions of work on AUTOLING appeared in 1967 (15,16) and 1968 (17), and described unconnected programs for learning morphology, context-free grammars and mono- and bilingual transformations.

1.4 Division of Labor between Man and Machine

Minimally, the human informant is assumed to be bilingual in English and the language under analysis. He must also be able to input sentences on a teletype in the subject language in a consistent transcription with spaces left between morphological units. (Eventually, a program to analyze morphological units will be added, eliminating this requirement.)

The informant is also required to answer 'YES' or 'NO' questions, posited by the program, of the form:

CAN YOU SAY: α

where α is a test sentence generated by the program from its tentative grammar of the language. The program may also ask the informant to supply corrections for illegal sentences (i.e., for test productions that he has previously rejected) if he can think of any corrections.

In practice, the informant may be a skilled linguist. He may choose to guide the form of the grammar under analysis by a careful choice of input sentences (it is at his option to inspect the embryonic grammar at any time). Also, he may, whenever he wishes, use special system commands to insert or delete rules of his choice, setting the mold for an application that would use AUTOLING as a partner in the analysis of the language.

The program, by referring to a number of analytic heuristics, continually posits new rules and revises old ones to account for new informant input sentences. When rules are coined or modified, their implications are tested by offering the informant sample productions derived from the modified rules in the posited grammar. As the informant accepts or rejects these test sentences, the grammar is modified accordingly.

The program also attempts to parse each new sentence input by the informant, a successful parse indicating the adequacy of the grammar for that sentence. If the (multi-path) parser cannot find any complete parse, the partially completed analyses (top nodes) are used as data for the rule coining heuristics. Test sentences that are rejected by the informant serve as a control on future rule construction, in that the program will not coin rules that permit an illegal sentence to parse. In some cases,, the program asks the informant for a correction of the rejected sentence and uses the response to posit and test a new transformation. Valid sentences that are supplied or approved by the informant serve as a control on the coining of new transformations, for the testing mechanism avoids the construction of transformations that block the production of sentences known to be legal.

1.5 Form of the Grammar

Because current transformational grammar theory is in a rapid state of change, we felt it advisable to build a model that was not restricted to the requirements of any particular theoretical variant. As a result, the system coins rules with very few restrictions as to form. Because of this paucity of restrictions, the context free (CF) rules essentially describe a surface structure. The transformations can be of almost any form that operates on a tree, and they provide for a variety of phenomena, including insertion, deletion and permutation. Specialized testing of particular theoretical models requires the addition of special restrictions to the rule coining processes (see section 4).

In one sense the kind of grammar produced is a function of the informant. In the absence of a morphological analysis component, he may rewrite his inputs morphemically or chose to distinguish allomorphs. In the latter case, the system will learn rules governing their distribution, doing the work of the missing component. If the informant transcribes phonologically conditioned allomorphs as separate units, the system will coin rules describing their distribution without reference to phonological conditioning. Minimally, the informant should make morphs with identical spellings but different meanings appear to the program as being different; otherwise, they will be treated as identities.

Much also depends on the extent to which the informant is willing to accept semantically nonsensical but grammatically correct test sentences. If he rejects such sentences, the program will, of necessity, learn a radically more complicated grammar that must account for aspects of the real world in its rules.

1.6 Aspects of the System Implementation

The program is operational on a Burroughs B5500 time-sharing system and is programmed in Burroughs Extended ALGOL. The program itself involves major use of list-processing techniques coupled with intensive data packing in partial words. Space is provided for a maximum of 1022 conflated CF rules (i.e., sets of rules with identical left-hand sides) and 3072 morphemes. The program requires approximately 30K words, at four instructions per word, and the arrays approximately 15K. When executed on the B5500 time-sharing system at the University of Wisconsin, an average, time-shared core allotment of 18 K is required.

During the course of language analysis, the program saves the entire state of the system at frequent intervals. Accordingly, it is possible to restart the analysis from an earlier point, if the informant makes an error, or to break off analysis until a later time. With this feature analysis of literally thousands of languages can be conducted at the same time, shifting from one to another as desired, and limited only by the available disk storage space.

2. The AUTOLING System

2.1 Overview

The overall flow of AUTOLING is shown in figure 1. The dashed lines append important subblocks which arise from deep within another block.

(insert figure 1 here)

The major subsystems assumed in figure 1 are:

a) The Multi-Path Parser, which yields all complete parses or sets of partial parses.

b) The Generator, which expands a CF rule into a terminal P-marker whose syntactic validity the informant must specify.

c) The Context-Free Inference Heuristics, which attempt to infer changes or additions to the CF grammar to account for a new utterance.

d) The Default Context-Free Heuristic, which will always coin a new CF rule to account for a new utterance.

e) The Substitution Heuristics, which attempt to substitute the name of a rule into another rule or transformation.

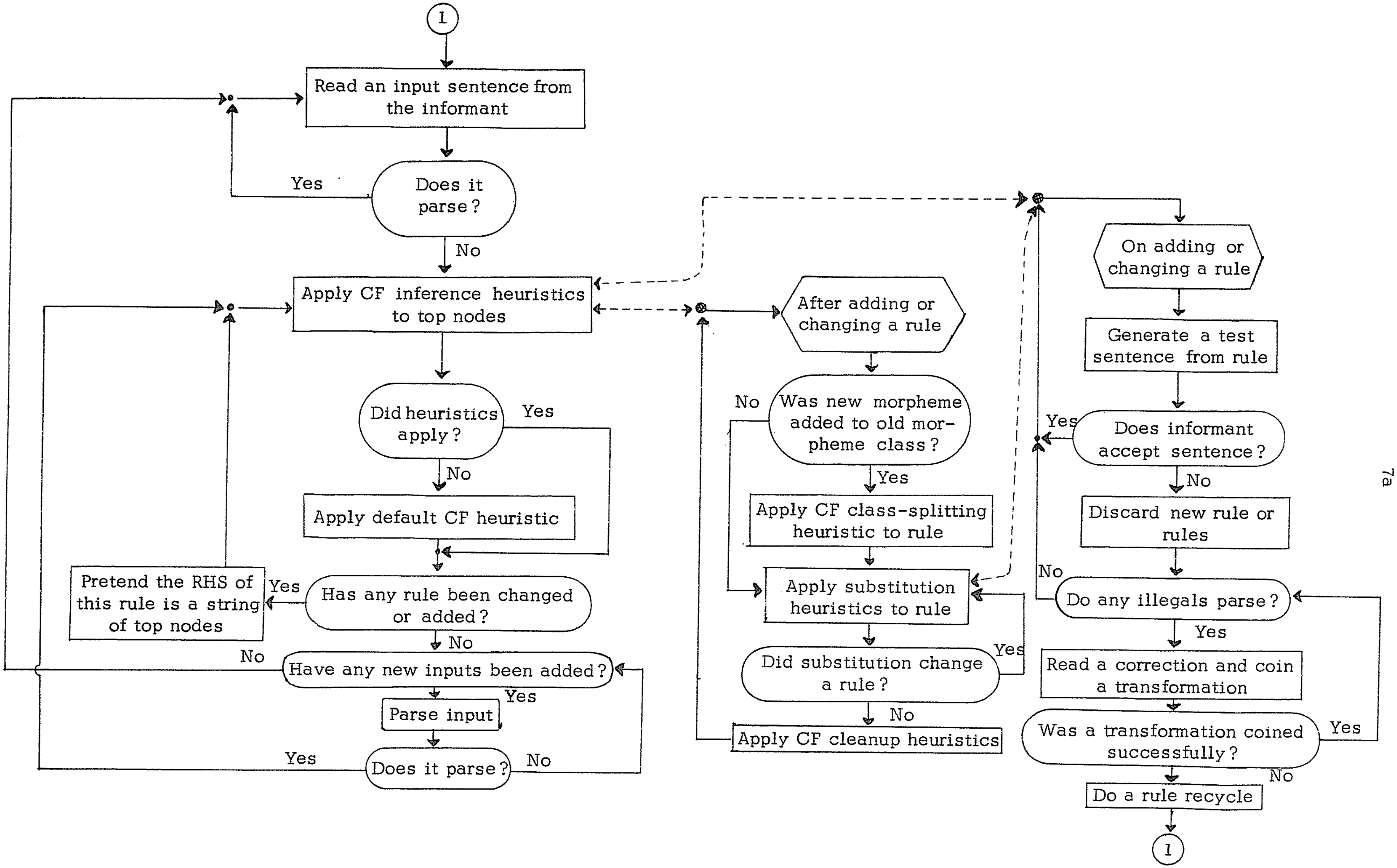

1
Read an input sentence from the informant
Does it parse?
Yes
No
Apply CF inference heuristics to top nodes
Did heuristics apply?
Yes
No
Apply default CF heuristic
Has any rule been changed or added?
Yes
No
Pretend the RHS of this rule is a string of top nodes
Have any new inputs been added?
No
Yes
Parse input
Does it parse?
Yes
No
After adding or changing a rule
Was new morpheme added to old morpheme class?
No
Yes
Apply CF class-splitting heuristic to rule
Apply substitution heuristics to rule
Did substitution change a rule?
Yes
No
Apply CF cleanup heuristics
On adding or changing a rule
Generate a test sentence from rule
Does informant accept sentence?
Yes
No
Discard new rule or rules
Do any illegals parse?
No
Yes
Read a correction and coin a transformation
Was a transformation coined successfully?
Yes
No
Do a rule recycle
1

FIGURE 1

Flowchart of Overall System Flow

f) The Class-Splitting Heuristic, which splits a morpheme class, as required, into two or more subclasses.

g) The Context-Free Cleanup Heuristics, which remove redundancies and extraneous rules from the CF component.

h) The Transformation Learning Heuristic, which attempts to coin a transformation from an illegal sentence and a correction.

i) The Transformation Combining Heuristic, which attempts to combine two transformations into a single transformation.

j) The Recycle procedure, which discards the current grammar and rebuilds a new one from the existing set of legal utterances.

An important aspect of AUTOLING, hinted at in figure 1, is the manner in which the illegal utterances are used to prevent the coining of bad rules. Basically, all the heuristics of AUTOLING attempt to induce rules or transformations with the widest possible generality. After any rule is changed, a sentence is generated and presented to the informant. If the informant accepts the sentence, the change and the generalities it implies are assumed correct. If the sentence is rejected, the change is discarded and a less general change may be proposed or not, depending on the particular heuristic. As the testing of the implications of a change is only partial, errors in the grammar will inevitably occur. One corrective measure is to coin a transformation. Before this is done, however, several less drastic remedies are usually tried. First, the changes made in the rule are withdrawn and an attempt is made to reparse the illegal (or illegals) which were generated in testing the change. If they no longer parse, the error is corrected. If they continue to parse, earlier changes, if any, are thrown out and the illegals retested. If illegals still parse, it is then necessary to coin a transformation. Further protection is provided by attempting to parse all known illegals whenever a change is made and before querying the informant.

As it is possible to coin CF rules and transformations which will permit illegal sentences to be generated, so is it possible to coin a transformation which blocks the generation of a known, legal sentence (called a "blocked" legal). This case arises whenever a transformation rewrites the P-marker of a legal sentence in such a manner as to yield a different terminal string. This problem is prevented by parsing and applying transformations to each legal sentence during the transformation coining process, thereby blocking the coining of any such transformations. The creation of such a bad transformation can also arise indirectly by a change to a CF rule. This situation is tested for periodically and, if found, leads to a rule recycle.

2.2 Multi-Path Parser

The parser is fully multi-path and bottom-to-top. It computes all possible complete parses of a string or, if there are none, will compute all possible partial parses. The highest level nodes of each partial parse (called "top nodes") are used by the CF heuristics in infering changes to the grammar, but only those strings of top nodes whose length is the same as that of the shortest string found are retained. In some cases levels of top nodes beneath the highest level will be used by the heuristics.

2.3 Generator

The generator is a rule testing heuristic, invoked whenever a change is attempted in the existing CF grammar or a new transformation is coined. Given a CF rule or a set of nodes derived from a newly coined transformation, it constructs a

P-marker containing this rule or set of nodes, expanded to terminal elements and dominated by a sentence rule. Expansion is primarily random but may be directed in the vicinity of the given rule or set of nodes. Transformations are then applied to the P-marker to yield a final form. If this form is unknown, the informant must pass on its syntactical validity. If it is a known illegal or a "blocked" legal sentence, the test has failed. If it is a known legal sentence, another sentence is generated, unless all possible sentences have already been examined.

2.4 Context-Free Inference Heuristics

2.4.1 Form of the Context-Free Grammar

The CF grammar consists of:

a) A set of terminal elements, T_i, which are morphemes.

b) A set of non-terminal elements, S_i.

c) A set of non-sentence rules, λ_i, of either one of two forms:

1) The morpheme class rule

$$\lambda_{k_i}: -S_k \rightarrow T_\ell$$

or

$$\lambda_{k_i}: -S_k \rightarrow S_\ell$$

where S_ℓ is itself a morpheme class rule.

2) The general form

$$\lambda_{k_i}: S_k \rightarrow \beta_1 \beta_2 \ldots \beta_n$$

where β_i is either a terminal, T_{β_i}, or a non-terminal, S_{β_i}. In general AUTOLING does not create any rules of this type with terminal and non-terminal elements intermixed.

d) A set of sentence rules, λ_i^*, of the form:

$$\lambda_{k_i}^*: \; {*S_k} \rightarrow \beta_1 \; \beta_2 \ldots \beta_n$$

where β_i is terminal or non-terminal. All non-sentence rules must be dominated by one or more sentence rules. A sentence rule may validly dominate another sentence rule.

Any rule (except a morpheme class rule) may be recursive.

2.4.2 Inference of New and Revised Rules

For each new utterance the CF inference heuristics first propose and then make and test changes and/or additions to the CF grammar. Each proposed change is based on the comparison of a string of top nodes against an existing rule in the grammar. Two major divisions are made, dependent on whether the match frame

yields a single or multiple mismatches. Further subdivision and classification is based on the type of mismatch(es) which occurs and permits a ranking of proposed changes by order of desirability. After establishing such a ranking over all strings of top nodes and rules, the highest ranked set of proposed changes is made to the grammar and tested by selective generation (see figure 2). If this test fails, the old grammar is restored and the next highest ranked changes are tried, etc. Should all sets of proposed changes fail, the default CF heuristic is invoked.

(insert figure 2 here)

The comparison of a string of top nodes with the right-hand side (RHS) of an existing rule results in a pattern of matched and mismatched elements. Only two such patterns are felt to infer a logical sequence of changes and/or additions to the CF grammar. By order of ranking (desirability) these are (see figure 3):

a) A single pair of mismatched strings and at least one pair of matched strings.

b) An alternating pattern of matched and mismatched pairs of strings, where all mismatched strings are of length one and both full strings are of the same length.

The basic heuristic is to coin for each mismatch two new rules whose RHS's are the mismatches and to change the rule on which the frame test was made to reflect these new rules (see figure 3).

(insert figure 3 here)

There are, however, several variations of this scheme, classified according to mismatch types as follows:

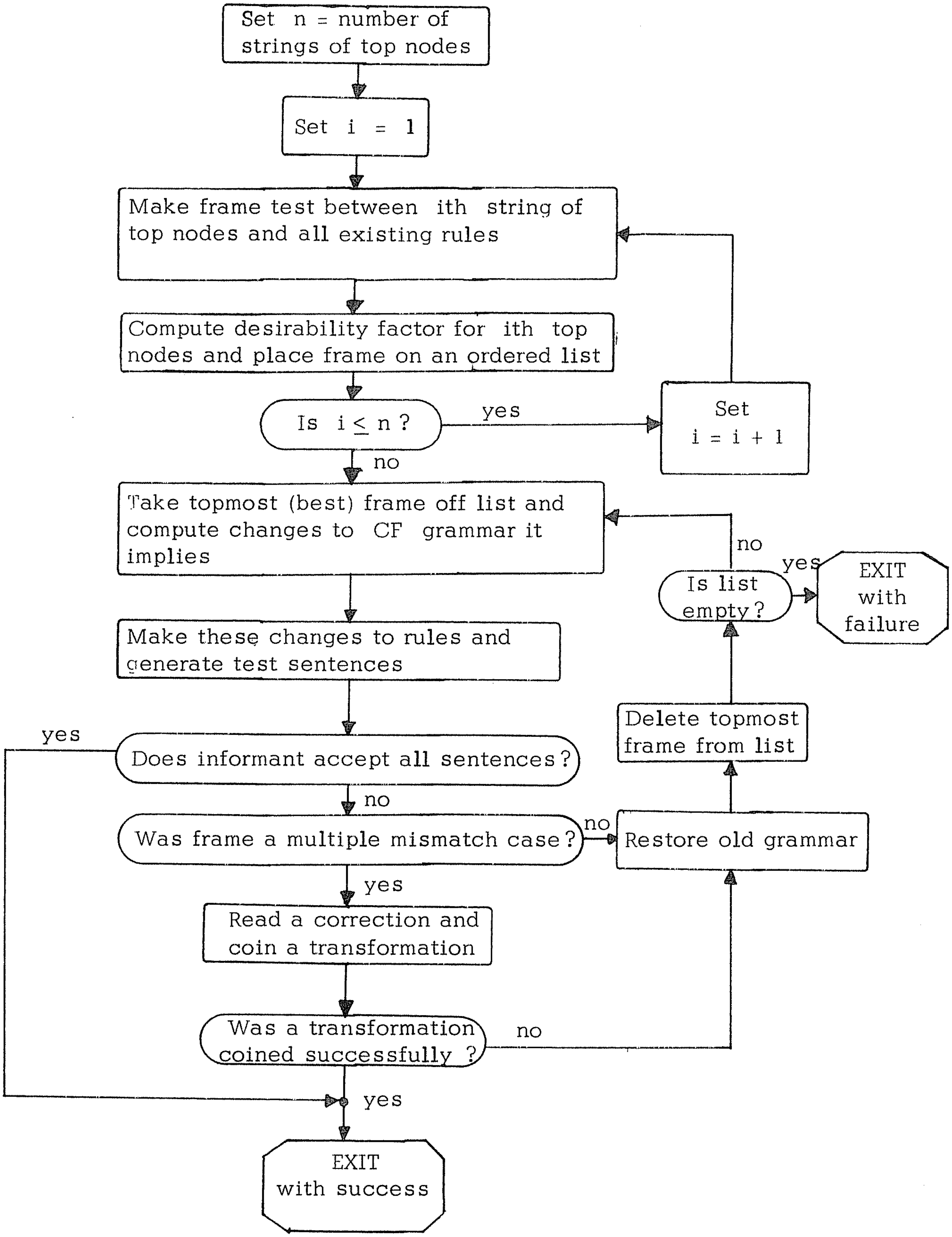

Set n = number of strings of top nodes
Set i = 1
Make frame test between ith string of top nodes and all existing rules
Compute desirability factor for ith top nodes and place frame on an ordered list
Is i ≤ n ?
yes
Set i = i + 1
no
Take topmost (best) frame off list and compute changes to CF grammar it implies
no
yes
Is list empty?
EXIT with failure
Make these changes to rules and generate test sentences
Delete topmost frame from list
yes
Does informant accept all sentences?
no
no
Was frame a multiple mismatch case?
Restore old grammar
yes
Read a correction and coin a transformation
Was a transformation coined successfully ?
no
yes
EXIT with success

FIGURE 2: Flowchart of Context-Free Inference Heuristics

$$\begin{array}{lll} \begin{array}{l} \text{Rule:} \\ \text{Topnodes:} \end{array} & \begin{array}{r} S_5 \rightarrow S_2\ \text{MAN} \\ S_2\ \text{MAN} \end{array} & \boxed{\begin{array}{l} S_6\ S_4 \\ S_7\ S_3\ S_9 \end{array}} \end{array}$$

Changed rule: $S_5 \rightarrow S_2$ MAN S_x

New rules: $S_x \rightarrow S_6\ S_4$
$S_x \rightarrow S_7\ S_3\ S_9$

(a) The single mismatch case

$$\begin{array}{lllllllll} \begin{array}{l} \text{Rule:} \\ \text{Topnodes:} \end{array} & \begin{array}{c} {*S_{11}} \rightarrow \\ \end{array} & \boxed{\begin{array}{c} S_3 \\ \text{THE} \end{array}} & \begin{array}{c} S_7 \\ S_7 \end{array} & \boxed{\begin{array}{c} \text{HIT} \\ \text{SIT} \end{array}} & \begin{array}{c} S_6 \\ S_6 \end{array} & \begin{array}{c} \text{BIG} \\ \text{BIG} \end{array} & \boxed{\begin{array}{c} S_4 \\ S_9 \end{array}} \end{array}$$

Changed rule: ${*S_{11}} \rightarrow S_3\ S_7\ S_x\ S_6$ BIG S_y

New rules: $-S_3 \rightarrow$ THE
$-S_x \rightarrow$ HIT | SIT
$S_y \rightarrow S_4 \mid S_9$

(b) The multiple mismatch case

1. S_3 must be a morpheme class marker.
2. If S_4 and S_9 are both morpheme class markers, the new rule will be

$$-S_y \rightarrow S_4 \mid S_9\ .$$

FIGURE 3

Examples for the Context-Free Inference Heuristic

a) A single morpheme vs. a single morpheme class marker. Coin a single new rule which adds the morpheme to the morpheme class (see figure 3b). Change the matched rule, if required.

b) A single element vs. a single element (except case (a)). Make the normal changes, but if the elements are either both morphemes or both morpheme class markers, mark the new rules as morpheme classes also (see figure 3b). The case of a morpheme vs. a non-morpheme class marker is not allowed.

c) An overlapping mismatch of non-terminals of the form

$$S_{\alpha_1} \ldots S_{\alpha_n} \quad \text{vs.} \quad S_{\alpha_1} \ldots S_{\alpha_n} S_{\alpha_1} \ldots S_{\alpha_n}$$

Coin the new left-recursive rules

$$S_x \rightarrow S_{\alpha_1} \ldots S_{\alpha_n}$$

$$S_x \rightarrow S_x S_{\alpha_1} \ldots S_{\alpha_n}$$

and substitute S_x into the matched rule. If the testing of these changes fail, try again as case (d).

d) A mismatch of two strings of non-terminals (except case (c)). Make the normal changes (see figure 3a).

Note that two specific possibilities are not allowed: adding a morpheme to a non-morpheme class and coining a rule with terminals and non-terminals intermixed on the RHS.

2.4.3 Detection of Redundancies and Comments

Several types of redundancies and loops are checked for and avoided when coining new rules. These redundancies arise only in cases (a) and (b) above, and are as follows:

a) For the mismatches S_x vs. S_y (or S_x vs. T_z), the rules $S_x \rightarrow S_y$ or $S_y \rightarrow S_x$ (or $S_x \rightarrow T_z$) already exist. Add no new rules but substitute the existing rule name in the matched rule.

b) For the mismatches α vs. β, there already exist rules $S_x \rightarrow \alpha$ and $S_x \rightarrow \beta$. Add no new rules but substitute S_x in the matched rule.

c) In the case of multiple mismatches, duplicates may occur. For example, in the frame

$$\begin{array}{c|c|cc|c|c|c|}\cline{2-2}\cline{5-5}\cline{7-7} P & Z & A & B & R & C & W \\ P & W & A & B & J & C & Z \\ \cline{2-2}\cline{5-5}\cline{7-7}\end{array}$$

the mismatch 'Z' vs. 'W' occurs twice. In such cases only one set of new rules is coined.

d) A loop will arise if rules such as

$$S_1 \rightarrow S_2$$

$$S_2 \rightarrow S_1$$

are coined. This is detected and prevented.

As noted earlier a single mismatch frame is ranked higher than a multiple mismatch case. There is one exception. In such a case as the frame

$$\begin{array}{c|c:c:c|cc}\cline{2-4} A & X & B & Y & C & Z \\ A & W & B & U & C & Z \\ \cline{2-4}\end{array}$$

rather than coining rules from 'XBY' and 'WBU', it is preferable to treat this as the double mismatch 'X' vs. 'W' and 'Y' vs. 'U'.

In the case of a multiple mismatch, there is a strong possibility that a context-sensitive pattern is present. Accordingly, if such a frame fails, the coining of a transformation is attempted before proceeding to the next ranked frame.

2.5 Default Context-Free Heuristic

This heuristic is applied when the CF inference heuristics all fail or when none apply. The basic default heuristic coins a new rule whose RHS is composed of a string of top nodes (see figure 4). Each string of top nodes is proposed as a rule in turn until one succeeds or all fail. If all fail, each string of top nodes is again proposed as a rule and an attempt is made to coin a corrective transformation, until a transformation is successfully coined or all attempts have failed. If all else fails, a new rule is coined whose RHS is the original utterance (morpheme string).

(insert figure 4 here)

A special case is that in which the string of top nodes of of length one, implying that the rule corresponding to the top node would yield a successful parse if it were a sentence rule. The heuristic thus attempts to make this rule into a sentence rule, and, if successful, all other rules with the same name are also proposed as sentence rules. As usual, each top node is tried in turn until one succeeds or all fail.

2.6 Substitution Heuristics

These heuristics are invoked whenever an existing rule is changed or a new rule is added. The RHS, Λ, of the changed (new) rule is compared against the RHS of each CF rule in the grammar. If the matched RHS contains the string Λ, in whole or in part, an attempt is made to substitute the name of the changed (new) rule for the matching part. There are three cases:

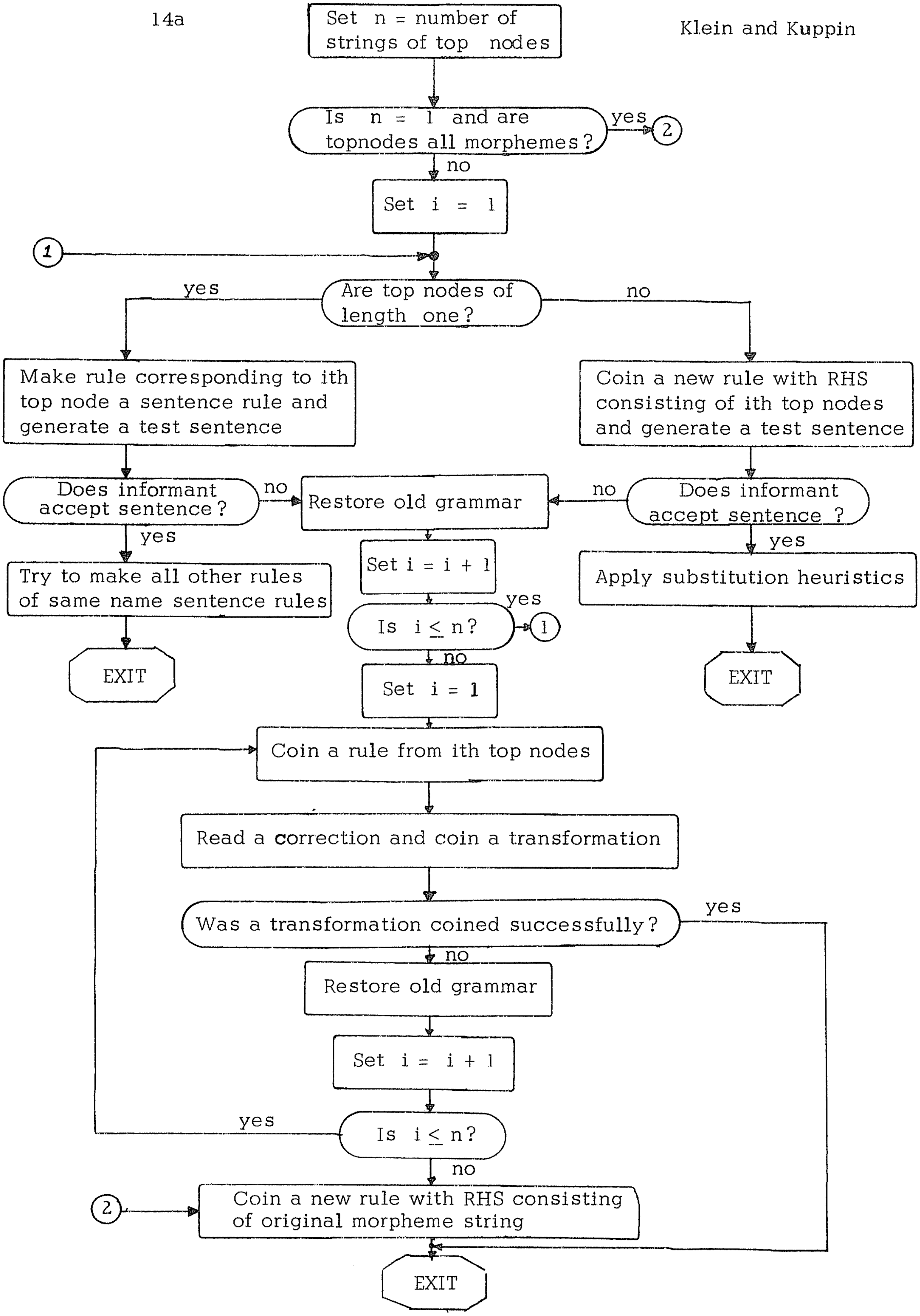

Set n = number of strings of top nodes
Is n = 1 and are topnodes all morphemes?
yes
2
no
Set i = 1
1
Are top nodes of length one?
yes
no
Make rule corresponding to ith top node a sentence rule and generate a test sentence
Coin a new rule with RHS consisting of ith top nodes and generate a test sentence
Does informant accept sentence?
no
Restore old grammar
no
Does informant accept sentence ?
yes
yes
Try to make all other rules of same name sentence rules
Set i = i + 1
Apply substitution heuristics
yes
Is i ≤ n?
1
EXIT
no
EXIT
Set i = 1
Coin a rule from ith top nodes
Read a correction and coin a transformation
Was a transformation coined successfully?
yes
no
Restore old grammar
Set i = i + 1
yes
Is i ≤ n?
no
2
Coin a new rule with RHS consisting of original morpheme string
EXIT

FIGURE 4: Flowchart of Default Context-Free Heuristic

a) Only part of the RHS is matched by Λ (see figure 5a). Proceed normally.

b) The RHS is identical to Λ (see figure 5b). This implies that the two classes of rules to which the RHS and Λ belong each have a rule with a common RHS. An attempt is made to merge these two classes into one.

c) Both sides of the changed (new) rule and the matched rule are identical. The changed (new) rule is discarded as redundant.

The substitution heuristic is then reapplied in turn to any rule which was itself changed during the substitution process.

(insert figure 5 here)

Substitution is also applied to each transformation in the grammar. The RHS, Λ, of the changed (new) rule is compared against the LHS of each transformation. If the LHS contains the string Λ in part. (but not wholely), the RHS of the transformation is checked to see that all occurrences of the string Λ on the LHS occur *in toto* on the RHS one or more times. If this condition is met, the substitution is made. For example (see 2.9.1 for a description of the format of transformations), for the transformation

$$\begin{array}{ccccccccccc} & & & & & 1 & 2 & 3 & 2 & 2 & 3 \\ A & B & C & D & \rightarrow & A & B & C & Z & B & C \\ & & & & & 1 & 2 & 3 & 4 & & \end{array}$$

and the rule

$$S_x \rightarrow BC,$$

a substitution would yield the new transformation

$$\begin{array}{ccccccc} & & & & 1 & 2 & Z & 2 \\ A & S_x & D & \rightarrow & A & S_x & Z & S_x \\ & & & & 1 & 2 & 3 & \end{array}$$

Changed rule: $S_5 \rightarrow S_6\ S_9$

Matched rule: $*S_{21} \rightarrow S_2\ S_6\ S_9$ THE S_7

New rule: $*S_{21} \rightarrow S_2\ S_5$ THE S_7

(a) Normal substitution

Changed rule: $-S_7 \rightarrow$ MAN

Matched rule: $-S_{12} \rightarrow$ MAN

Other rules: $-S_7 \rightarrow$ BOY

$-S_{12} \rightarrow$ RAM

$*S_{16} \rightarrow$ THE $S_7\ S_{19}$

New rules: $S_{12} \rightarrow$ MAN

$S_{12} \rightarrow$ BOY

$S_{12} \rightarrow$ RAM

$*S_{16} \rightarrow$ THE $S_{12}\ S_{19}$

(b) Class union

Note: All occurrences of S_7 on the RHS of a CF rule or transformation are replaced by S_{12}.

FIGURE 5

Examples for the Substitution Heuristics

If the transformation were of the form

$$\begin{array}{cccccccccc} & & & & & 1 & 2 & 3 & Z & 2 \\ A & B & C & D & \rightarrow & A & B & C & Z & B \\ & & & & & 1 & 2 & 3 & 4 & \end{array}$$

no substitution would take place as the partial string 'B' occurs on the RHS.

2.7 Class-Splitting Heuristics

Whenever a new morpheme is added to an existing morpheme class by the CF inference heuristics, all other occurrences of this class in the grammar are tested with the new morpheme. If the morpheme belongs in the class in some cases and is not permitted in other cases, the morpheme class is split and the occurrences of the class adjusted accordingly (see figure 6). When a new morpheme is added to a class which is already split, the morpheme may actually belong at a different level then that at which it was introduced. In such cases the correct level at which to enter the new morpheme is determined.

(insert figure 6 here)

2.8 Context-Free Cleanup Heuristics

Extraneous, redundant, and duplicate CF rules can arise from the normal actions of the CF and substitution heuristics. Duplicate rules are usually detected and eliminated by the substitution heuristic. It should not be possible for two identical rules to exist except during intermediate stages of rule coining.

There are two main cases in which redundant and extraneous rules arise and are resolved:

a) There is a rule $S_x \rightarrow S_y$, but no other occurrences of S_y on the RHS of any rule. All rules named S_y are renamed S_x.

New morpheme: $-S_2 \rightarrow \mathrm{WATER}$

Old rules: $-S_2 \rightarrow \mathrm{TREE}|\mathrm{LAND}$

$S_{19} \rightarrow \mathrm{THE}\ S_2\ S_9$ (WATER okay)

*$S_{21} \rightarrow \mathrm{A}\ S_2\ S_{14}\ S_7$ (WATER not permitted)

New rules: $-S_x \rightarrow S_2$

$-S_x \rightarrow \mathrm{WATER}$

$-S_2 \rightarrow \mathrm{TREE}|\mathrm{LAND}$

$S_{19} \rightarrow \mathrm{THE}\ S_x\ S_9$

*$S_{21} \rightarrow \mathrm{A}\ S_2\ S_{14}\ S_7$

FIGURE 6

Example for the Class-Splitting Heuristic

b) There is only a single rule named S_x, of the form $S_x \rightarrow S_y$. Change all references to S_x to S_y.

In both cases the redundant rule is then discarded and transformational references are revised.

2.9 Transformation Learning Heuristic

2.9.1 Form of the Transformations

The following format for transformations has been adopted:

a) All transformations are mandatory and ordered.

b) Each transformation in turn is applied repeatly until it no longer applies then the next transformation is applied, etc. In order to prevent infinite application loops (since it is not possible to restrict the form of transformation coined to achieve this end), a count is made of the number of times a transformation may initially apply to a P-marker. Using this count as a heuristic control the transformation is reapplied to the P-marker until the possibility of a loop is indicated. An extended count prevents more subtle loops from arising by arbitrarily terminating the application process after a maximum number of applications, dependent of the number of nodes in the P-marker.

c) The LHS of a transformation specifies a labelled string of nodes which must be logically adjacent in the P-marker. The RHS specifies the resolution of each and every LHS-node, plus any insertions. Each LHS-node may be replaced by a labelled LHS-node (including itself) or by a terminal or non-terminal element, or may be deleted. Labelled LHS-nodes (including duplicates) and terminal or non-terminal elements may be inserted before or after any LHS-node. Insertions are attached to the highest common point between the nodes at which the insertion is made.

d) If any replacement or insertion is a non-terminal element, it is expanded randomly to a terminal string, as the P-marker must always be terminal.

A transformation may apply to either a partial or a complete sentence. Figure 7 shows an example of a transformation and its application to a P-marker.

(insert figure 7 here)

2.9.2 Overview of Transformational Learning

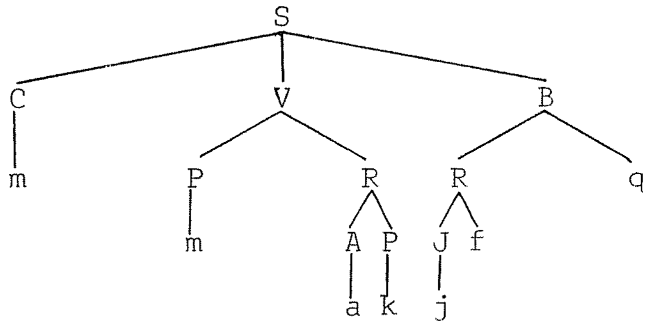
An attempt is made to coin a transformation when an irresolvable inconsistency in the CF grammar is detected or when there is a strong possibility that a context-sensitive pattern is present. A transformation is coined that transforms the P-marker of an illegal sentence into the P-marker for a corrected version of the illegal. In the coining process extensive testing is done in order to develop the most general possible transformation that will resolve the inconsistency or context-sensitive pattern.

The transformational coining process can be divided into four basic steps (see figure 8):

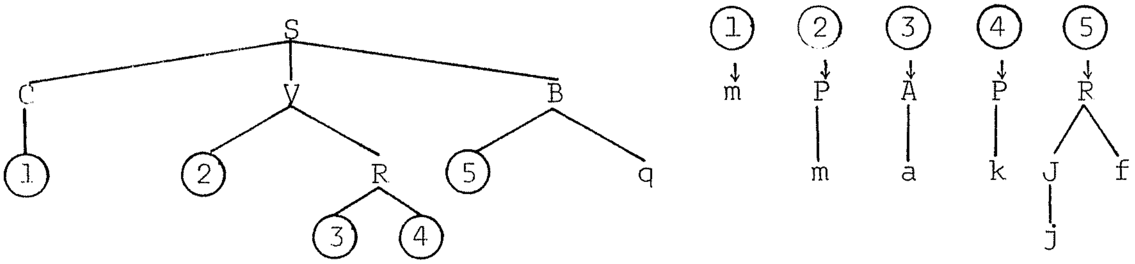
a) Compute and rank all possible allignments between the illegal and corrected strings. Retain only the best allignment or, if there is more than one "best" allignment, retain all of them.

```
                                 5 1 * 3 2 Z 4 k 5
Transformation:      m P A P R ⟹ R m * A P Z P k R
                     1 2 3 4 5      1 2 3     4   5
```

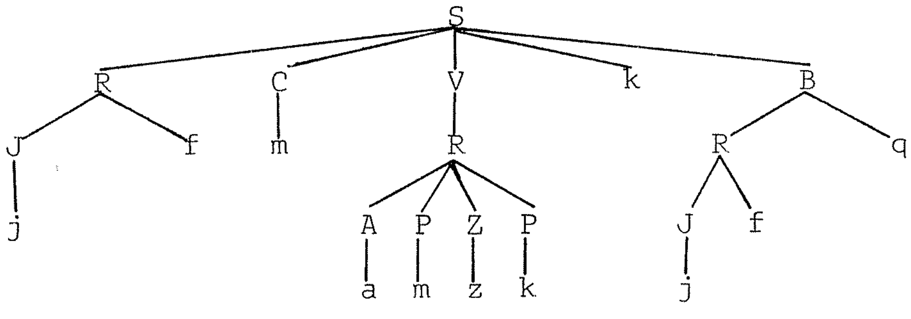
Original P-marker:



Decomposed P-marker:



Transformed P-marker:



1. Capital letters represent non-terminal elements and small letters represent terminal elements.
2. An asterisk (*) indicates a deleted node.
3. The numbers beneath the RHS correspond to LHS-nodes on a one-to-one basis. Unnumbered elements are insertions. The upper numbers and nodes on the RHS indicate what the insertion or replacement is to be. An upper number, i, indicates that the ith node on the LHS is to replace the node (or be inserted) on the RHS.
4. Because 'Z' is a non-terminal, it is expanded in the transformed P-marker.

FIGURE 7: Example of Transformation Application.

b) Using the best allignment, construct a P-marker for the correction by transferring structure from the illegal P-marker. If no transformation can be coined from this allignment, repeat the process on the next allignment, if any.

c) Temporarily delete any extraneous context from the ends of the illegal and correction strings. If an attempt to coin a transformation with the deleted context fails, restore it a piece at a time until a successful transformation is coined, or all fail.

d) Each node of the illegal P-marker is raised selectively and tested by generation. If the generation fails or if the transformation blocks the coining of a known legal, the node is lowered back to its former level. In this manner the most genral possible transformation is coined.

e) If all else fails, a transformation is coined whose LHS is the illegal morpheme string and whose RHS is the corrected morpheme string.

(insert figure 8 here)

2.9.3 Ranking the Allignments and Assignment of Structure

An algorithm has been developed which efficiently computes and ranks all possible allignments. The ranking is based on the number and length of matched (alligned) pieces and on the centering. Additional weight is given to allignments which preserve higher level structure. In figure 9 three of the better allignments for an illegal/correction pair are shown.

(insert figure 9 here)

(a) and (b) are ranked markedly higher than (c) because both preserve higher level structure whereas (c) does not. (a) is preferable to (b) because it is better centered.

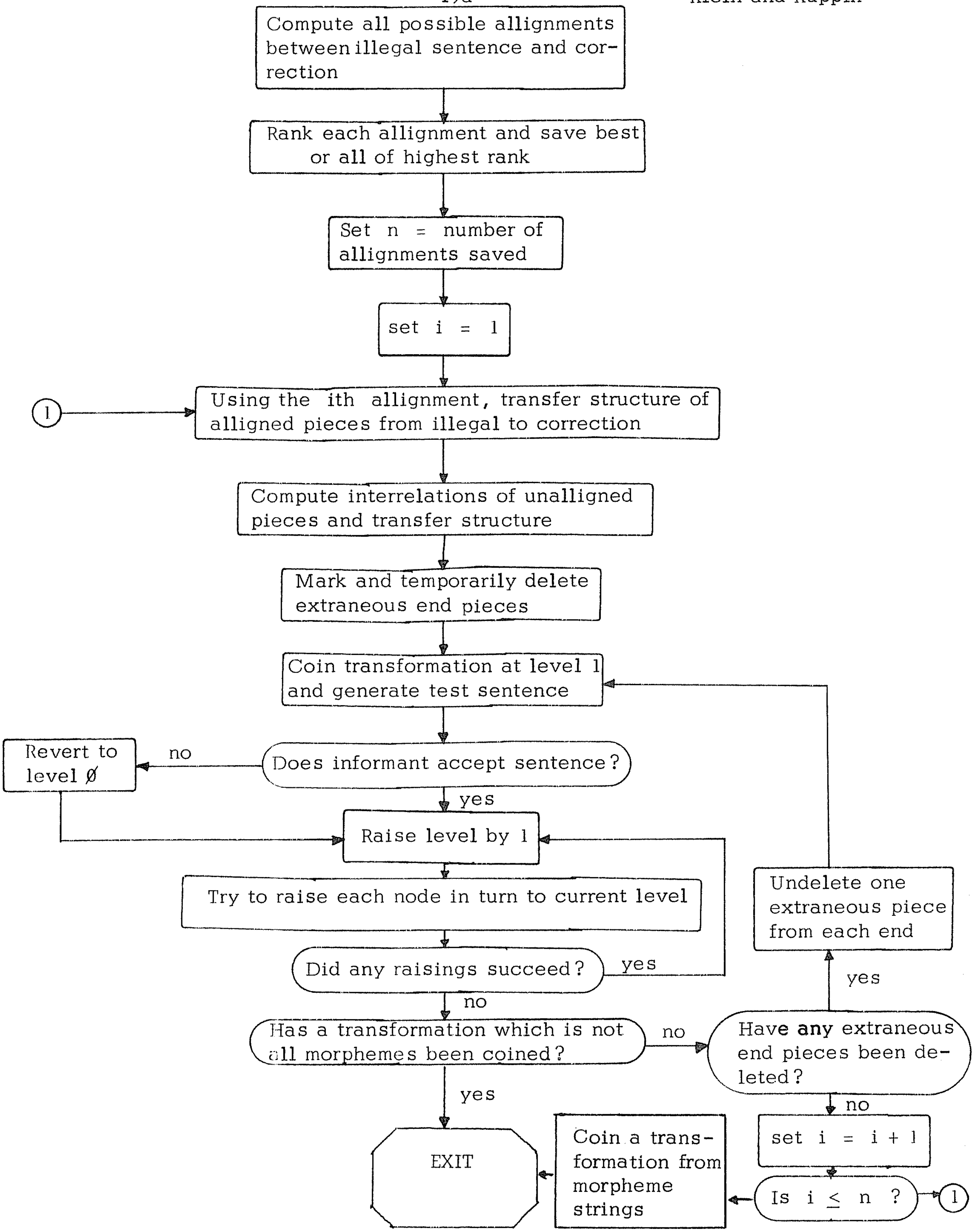
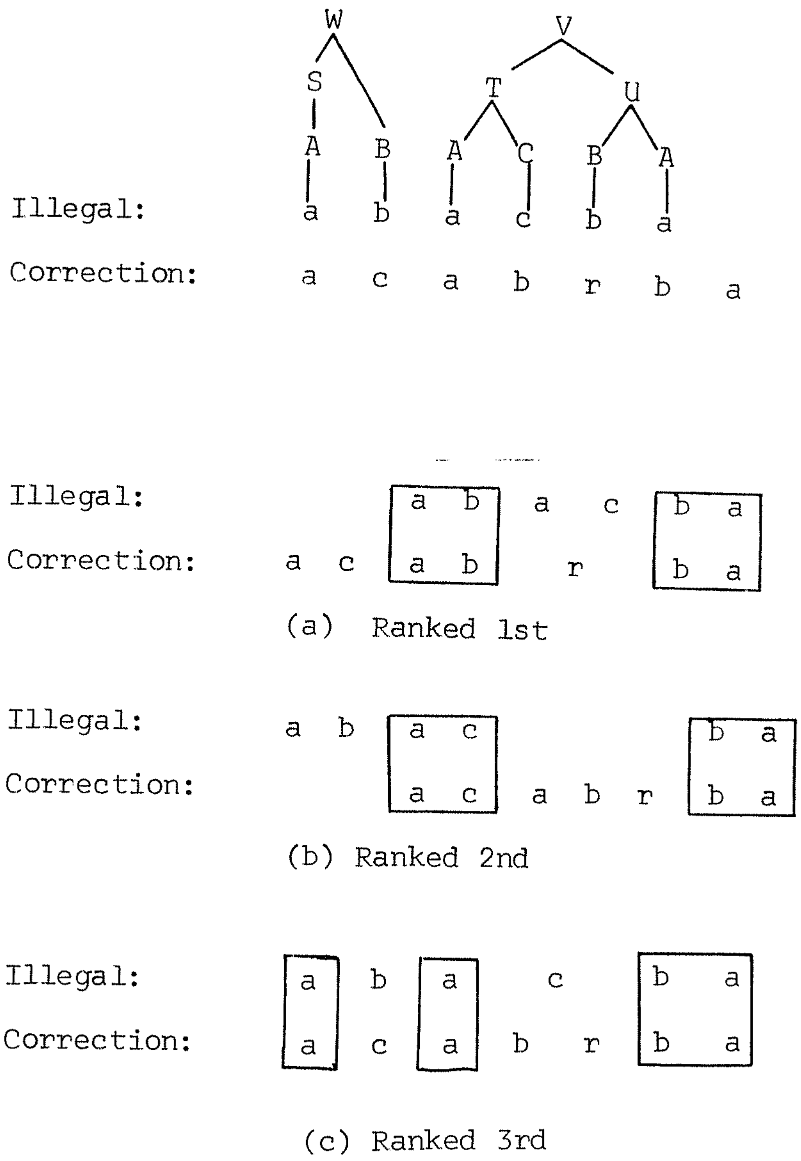

Compute all possible allignments between illegal sentence and correction
Rank each allignment and save best or all of highest rank
Set n = number of allignments saved
set i = 1
1
Using the ith allignment, transfer structure of alligned pieces from illegal to correction
Compute interrelations of unalligned pieces and transfer structure
Mark and temporarily delete extraneous end pieces
Coin transformation at level 1 and generate test sentence
Does informant accept sentence?
no
Revert to level Ø
yes
Raise level by 1
Try to raise each node in turn to current level
Did any raisings succeed?
yes
no
Has a transformation which is not all morphemes been coined?
no
Have any extraneous end pieces been deleted?
yes
Undelete one extraneous piece from each end
no
set i = i + 1
Is i ≤ n ?
1
yes
EXIT
Coin a transformation from morpheme strings

FIGURE 8: Flowchart for Transformation Learning Heuristic


W
V
S
T
U
A B A C B A
Illegal: a b a c b a
Correction: a c a b r b a
Illegal: a b a c b a
Correction: a c a b r b a
Illegal: a b a c b a
Correction: a c a b r b a
Illegal: a b a c b a
Correction: a c a b r b a

(a) Ranked 1st

(b) Ranked 2nd

(c) Ranked 3rd

FIGURE 9: Examples of Transformational Allignment

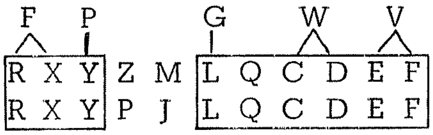
Given an allignment a P-marker must be constructed for the correction. Structure above alligned pieces of the illegal is transferred in toto to the correction. To assign structure to the unalligned pieces an interrelation map is drawn. This map correlates each unalligned node in the correction with a node in the illegal, except for newly inserted terminal or non-terminal elements. Each unalligned element of the correction is associated with the first occurrence of that element in the illegal. Figure 10 shows the interrelation maps and complete P-markers for the examples in figures 9a and 9b.

(insert figure 10 here)

The interrelation map governs the transferrence of structure to the correction and further distributes operations performed on the illegal over the correction. Thus in figure 10a any operation on the first 'a' of the illegal will also be performed on the first and third 'a' of the correction.

2.9.4 Deletion of Extraneous Context

Extraneous context is defined as any alligned pieces on the ends of the illegal/correction pair that do not govern the context of the transformation. In figure 10 the right-end string 'BA' is such an extraneous piece and would be deleted on the first attempt to coin a transformation. This heuristic also takes into consideration the structure over the end pieces, in that it deletes (on each end) only those nodes up to, but not including, the first node which is not dominated by a higher level node of length two or more. For example, for the case



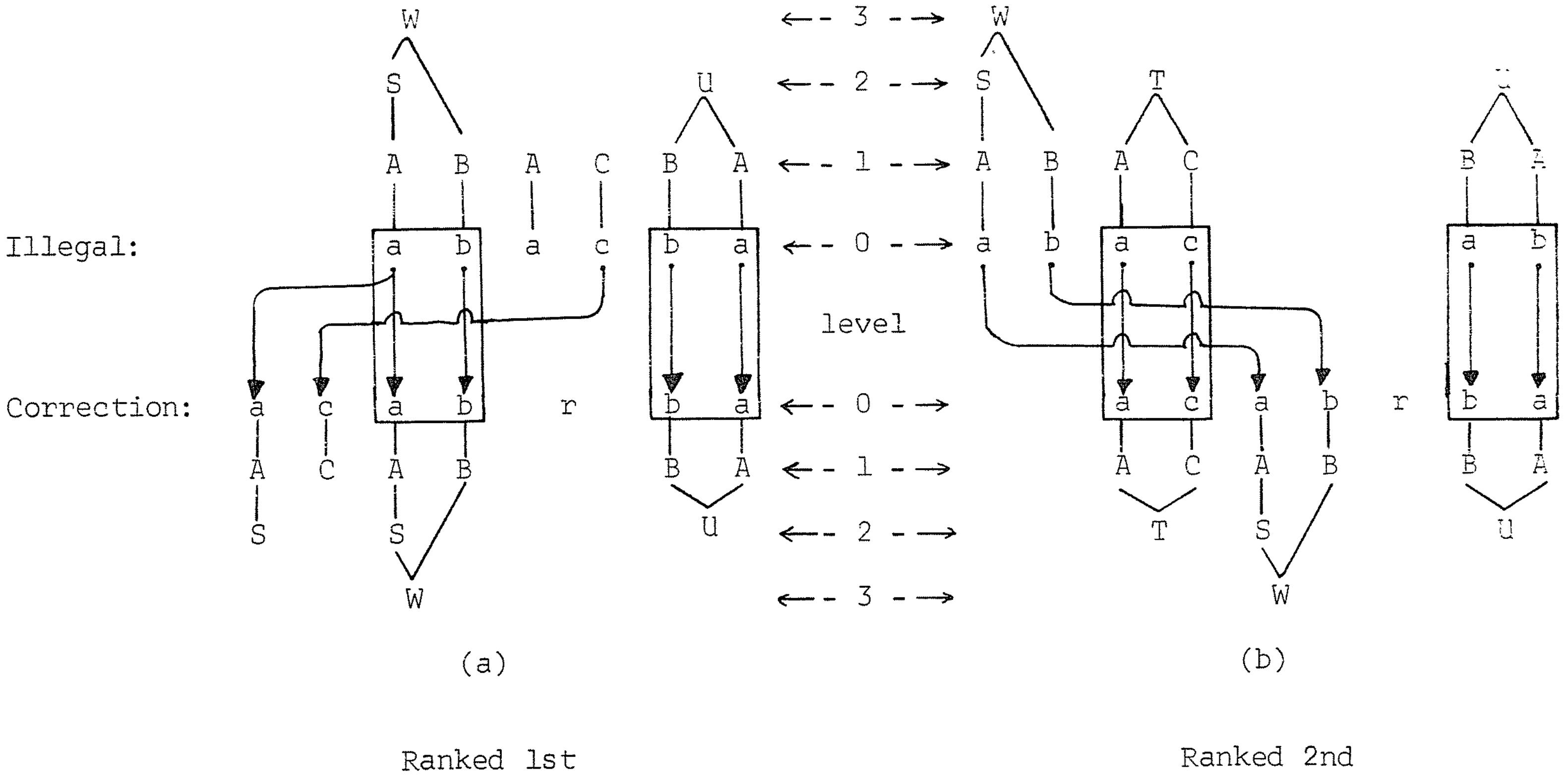


FIGURE 10: Examples of Interrelation Maps and Structural Assignments

'RX' and 'CDEF' would be deleted, but not 'Y' or 'LQ'. When an attempt to coin a transformation fails, any deleted end pieces are restored, each restoration cycle restoring one piece on each end. Thus for the above example 'RX' and 'CD' would be restored on the first cycle and 'EF' on the second.

2.9.5 Coining and Testing a Transformation

With the interrelation map, the corresponding P-markers, and the set of temporarily deleted end pieces, a transformation can finally be coined. The most general transformation would be that coined from the highest nodes of the P-markers, but this is not the most probable, as it imposes the fewest context-sensitive restrictions. Rather, the first attempt is made on a less general form, which will then be generalized in stages until the least restrictive possible transformation is obtained.

As a first attempt the transformation consisting of all level 1 nodes is coined, where the morpheme string is at level 0 (see figure 10). Nodes which do not rewrite to level 1 are retained at level 0. For the example in figure 10a raising all nodes to level 1 would yield the transformation

$$\begin{array}{ccccccccccc} & & & & & 1 & 4 & 1 & 2 & r & * \\ A & B & A & C & \Longrightarrow & A & C & A & B & r & * \\ 1 & 2 & 3 & 4 & & & & 1 & 2 & 3 & 4 \end{array}$$

where the end piece, 'BA', is deleted. This transformation is tested by pretending that the RHS of the transformation, plus all deleted end pieces, is a rule and then generating under it. Generation is completely random under the deleted end pieces, but under the rest, the specific rules involved in the structure are excluded from the first level of generation. Thus the rules A → a, B → b, C → c will not be used in the above example (unless they are unique).

If the level 1 transformation fails, all nodes are dropped to level 0. In either case, the following generalization cycle is applied to each node in turn at each level until all applications at a single level fail:

a) Given a level, μ, raise each node in turn, serially from left to right, up to level μ if it is already at level $\mu - 1$. If the level μ node dominates more than one node, these lower nodes are absorbed.

b) Parse all legals and see if any are blocked from being generated. If so, restore the node to level $\mu - 1$.

c) Do a test generation on the corresponding transformation. If the test fails, restore the node to level $\mu - 1$.

d) After all nodes have been tried at level μ: if any node has been raised to level μ, repeat this process at level $\mu + 1$; otherwise, quit.

All raising and lowering of nodes is done on the LHS and is transferred to the RHS through the interrelation map. When interrelations are lost by raising a LHS-node too high, the last level achieved on the RHS is retained. For example in figure 10a the transformation

```
          S 3 1 r *
W A C ==> S C W r *
1 2 3         1 2 3
```

would be coined in the most general case that all nodes are raised to their highest level. The first 'a' of the illegal controls the first 'a' of the correction, but the first correction 'a' cannot be raised above an 'S' node, even when the illegal 'a' is raised to a 'W'.

Certain other criteria must be met in coining a transformation:

a) Neither side may contain only a single node.

b) The two sides must not be identical.

These conditions are checked before testing the transformation on the informant. If any condition applies, the raised node is lowered and testing proceeds as usual. As a last restore a transformation is coined from the two morpheme strings.

2.10 Transformation Combining Heuristic

This heuristic is invoked whenever an existing transformation is changed or a new transformation is added. The RHS, Γ, of the changed (new) transformation is compared against the RHS of every other transformation. If Γ and the matched RHS are of the same length and both contain identical rewrite descriptors, a table is made of all mismatched pairs of nodes. The rewrite descriptors are the numbers above and below the RHS of the transformations. These must be identical for a match to be accepted. Given the table of RHS mismatches, the LHS's of the two transformations are compared. For each LHS mismatch pair there must be one or more identical RHS pairs (unless neither member of the LHS pair appears on the RHS) and to each RHS pair there must correspond at least one LHS pair. If such sets of mismatches exist, a set of CF rules is proposed in the same manner as for the CF inference heuristics (see 2.4.2) and the changes are made to the two matched transformations and tested. If the tests succeed, the earliest (lowest numbered) transformation is discarded and the other matched transformation is retained, as changed. Otherwise, the combining fails and the original transformations are restored. In the example in Figure 11 the three RHS pairs are (X,Q), (Y,R), and (Y,R). Furthermore, the rewrite descriptors of T_i and T_j are identical. The LHS pairs are (X,Q), (Y,R), and (B,D). The first two LHS pairs account for all three RHS pairs. The pair (B,D) is permitted as it does not occur on the RHS.

(insert figure 11 here)

Figure 11

$$T_i:\quad \underset{1}{A}\ \ \underset{2}{X}\ \ \underset{3}{Y}\ \ \underset{4}{B}\ \rightarrow\ \overset{2}{X}\ \ \underset{1}{\overset{1}{A}}\ \ \underset{2}{\overset{*}{*}}\ \ \underset{3}{\overset{3}{Y}}\ \ \underset{4}{\overset{C}{C}}\ \ \overset{3}{Y}$$

$$T_j:\quad A\ \ Q\ \ R\ \ D\ \rightarrow\ \overset{2}{Q}\ \ \underset{1}{\overset{1}{A}}\ \ \underset{2}{\overset{*}{*}}\ \ \underset{3}{\overset{3}{R}}\ \ \underset{4}{\overset{C}{C}}\ \ \overset{3}{R}$$

LHS mismatch pairs:

$$\begin{array}{|c|cc|c|c|c|}
2 & 1 & * & 3 & C & 3 \\
X & A & * & Y & C & Y \\
 & 1 & 2 & 3 & 4 & \\
2 & 1 & * & 3 & C & 3 \\
Q & A & * & R & C & R \\
 & 1 & 2 & 3 & 4 &
\end{array}$$

RHS mismatch pairs:

$$\begin{array}{c|c|c|c|}
\cline{2-4}
A & X & Y & B \\
1 & 2 & 3 & 4 \\
 & & & \\
A & Q & R & D \\
1 & 2 & 3 & 4 \\
\cline{2-4}
\end{array}$$

Proposed combined transformation and new CF rules:

$$T_j:\quad \underset{1}{A}\ \ \underset{2}{S_\alpha}\ \ \underset{3}{S_\beta}\ \ \underset{4}{S_\gamma}\ \rightarrow\ \overset{2}{S_\alpha}\ \ \underset{1}{\overset{1}{A}}\ \ \underset{2}{\overset{*}{*}}\ \ \underset{3}{\overset{3}{S_\beta}}\ \ \underset{4}{\overset{C}{C}}\ \ \overset{3}{S_\beta}$$

$$S_\alpha \rightarrow X|Q$$

$$S_\beta \rightarrow Y|R$$

$$S_\gamma \rightarrow B|D$$

2.11 Recycle

Even by coining a transformation it is not always possible to repair inconsistencies in the grammar. Normally this situation arises as follows: at some point a transformation is coined to resolve an inconsistency. Later, another, related inconsistency is introduced, but the transformation which will correct this new inconsistency also nullifies or reverses the effect of the earlier transformation. The result is a deadlock. In such a case, or whenever a transformation cannot be coined because the informant can give no correction or an irresolvable "blocked" legal is encountered, a rule recycle takes place. First, the entire grammar is discarded and AUTOLING is basically reinitialized, except that the lists of known utterances are retained. Then the known, legal utterances are fed back through AUTOLING as if the informant had typed them in. Because the illegal utterances or the "blocked" legal which caused the recycle are present throughout the reprocessing, the inconsistency which led to the recycle is prevented from reoccurring. The recycle is a costly and therefore always a last-resort undertaking.

3. Examples and Discussion

Full histories of the analysis of natural language problems by AUTOLING are too lengthy for presentation in this paper. The analysis of some artificial languages are offered in sections 3.1, 3.2, 3.3, 3.4, and 3.5. Segments of the analyses of subsets of English are offered in section 3.6 and 3.7.

3.1 $A^n B^n$

A complete interactive dialogue, plus final grammar, for $A^n B^n$:[1]

```
1: A B←

2: A A B B←

3: A A A B B B←
CAN YOU SAY:
.1: A A A A A B B B B B
YES←

4: *TYPE←
 *S3 := A S3 B
 *S3 := A B

4: A A A A A A A A A A B B B B B B B B B B ←
-PARSED OK-
```

The response "PARSED OK" to input 4 indicates that the grammar can account for this sentence.

3.2 $A^n B^m C^n$

A complete interactive dialogue, plus intermediate and final grammars, for $A^n B^m C^n$:

[1] Lines terminated with a left-arrow (←) are informant input (numbering is supplied by the program).

```
1: A B C←

2: A BB C←

3: A B B C←
CAN YOU SAY:
.1: A BB BB BB B C
YES←

4: *TYPE←
 *S1 := A S3 C
 -S2 := BB
 -S2 := B
  S3 := S3 S2
  S3 := S2

4: A A B C C←
CAN YOU SAY:
.1: A A A A A BB BB C C C C B BB C
NO←

5: *TYPE←
 -S2 := BB
 -S2 := B
  S3 := S3 S2
  S3 := S2
 *S4 := A S4 C
  S4 := S3

5: A A A A A A A B B B B C C C C C C C←
-PARSED OK-

6: A A A B BB C C C←
-PARSED OK-
```

To derive this grammar it was necessary to provide 'B' with the companion 'BB', because the CF inference heuristics, under certain conditions, block the derivation of rules containing both terminal and non-terminal elements on the RHS (see section 2.4.2). For an alternate method of handling this restriction see section 3.6.

3.3 Well-Formed Arithmetic Expressions

An abbreviated interactive dialogue (with queries to the informant and his replies excluded), plus final grammar, for some well-formed arithmetic expressions:[1]

```
1: A PLUS B←

2: A DIV B←

3: B PLUS A←

4: A MULT B←

5: A MIN B←

6: LP A PLUS B RP←

7: A MIN LP B MULT B RP←

8: LP A PLUS LP A MULT B RP MIN B RP←

9: A PLUS B MIN A←

10: A PLUS B PLUS A MULT B DIV A←

11: *TYPE←
 -S2 := MIN
 -S2 := MULT
 -S2 := DIV
 -S2 := PLUS
 -S3 := B
 -S3 := A
 *S8 := S8 S2 S8
  S8 := S3
 *S8 := LP S8 RP
```

[1] 'LP' and 'RP' stand for '(' and ')' , respectively.

3.4 Complex Agreement Example

Legal sentences in the following language require agreement among occurrences of X, Y, and Z, and also among occurrences of A, B, and C. Thus X A X A and Y B Y B are legal, but not X A X B or Y A X B. Sentence types *S1 and *S3 were introduced as a rapid means of combining X, Y, and Z and A, B, and C in the same classes.

```
11: *TYPE
 *S1  := F F S2
 -S2  := Z
 -S2  := Y
 -S2  := X
 *S3  := G G S4
 -S4  := D
 -S4  := C
 -S4  := B
 -S4  := A
 *S6  := S6 S6
  S6  := S2 S4

                               1  2  1  4
T2: S2 S4 S2 S4   =:    S2 S4 S2 S4   (1)
     1  2  3  4          1  2  3  4

                               1  2  3  2
T5: S2 S4 S2 S4   =:    S2 S4 S2 S4   (1)
     1  2  3  4          1  2  3  4
```

3.5 Complex Agreement Example

This language is similar to that of 3.4 except that sentences such as X A, X B, and Y A are also legal. As a result, agreement can be handled with a single transformation because of the existence of a higher level PS unit not present in the language of 3.5. Recursion is introduced into the language, and the final grammar reflects an updating of both the phrase structure rules and the agreement transformation.

```
9: *TYPE FULL←
 *S1  := F  F  S2
 -S2  := Z
 -S2  := Y
 -S2  := X
 *S3  := G  G  S4
 -S4  := C
 -S4  := B
 -S4  := A
 *S5  := S2  S4
 *S6  := S5  S5

                      1   1
T1: S5  S5   =:   S5  S5   (1)
     1   2         1   2


9: X  A  X  A←
-PARSED OK-

10: X  A  X  A  X  A←
CAN  YOU  SAY:
.1: Y  A  Y  A  Y  A  Y  A
YES←
CAN  YOU  SAY:
.2: Y  C  Y  C  Y  C  Y  C
YES←
CAN  YOU  SAY:
.3: Y  B  Y  B
YES←
-LEGALS PARSED-
*EMMA2.2 SAVED*

11: *TYPE FULL←
 *S1  := F  F  S2
 -S2  := Z
 -S2  := Y
 -S2  := X
 *S3  := G  G  S4
 -S4  := C
 -S4  := B
 -S4  := A
 *S5  := S2  S4
 *S7  := S7  S7
  S7  := S5
                      1   1
T1: S7  S5   =:   S7  S7   (1)
     1   2         1   2
```

3.6 Morphological Agreement Problem in English

A special mode introduced by the label '###ENTER MORPHS###' permits the direct inputting of morphs or other strings without subsequent application of the usual phrase structure heuristics. An as yet to be created morphological analyzer will make use of this mode. Its primary function in the following example is to permit the phrase structure heuristics to apply to morphs having no other members in their class. The introduction of elements in the morph mode provides them with a class name, thereby avoiding the heuristic application restrictions mentioned at the end of section 2.4.2.

In this example, the plural morphs have been treated as separate units. A preliminary morphological analysis might have combined 'S' and 'ES' into a higher level unit on the basis of phonological conditioning, and perhaps added 'EN' on other grounds. The following example indicates the program's ability to derive a grammar for this type of data without such pre-analysis. Note in particular the way transformations are combined during the course of analysis.

I. A Full Learning Sequence

```
###ENTER MORPHS###

1: EN←

2: S←

3: SG←

4: ES←

5: *SWITCH←


###ENTER SENTENCES###


5: THE CAT←

6: THE DOG←

7: THE FOX←

8: THE BUSH←

9: THE OX←
-LEGALS PARSED-
*LSUN1.1 SAVED*

10: THE CAT S←
CAN YOU SAY:
.1: THE BUSH S
NO←
CORRECTION:
.2: THE BUSH ES←
CAN YOU SAY:
.3: THE DOG ES
NO←
CORRECTION:
.4: THE DOG S←
CAN YOU SAY:
.5: THE OX S
NO←
CORRECTION:
.6: THE OX EN←
CAN YOU SAY:
.7: THE FOX ES
YES←
CAN YOU SAY:
.8: THE DOG EN
NO←
CAN YOU SAY:
.9: THE FOX EN
NO←
CORRECTION FOR CYS .8:
.10: THE DOG S←
```

```
11: *TYPE FULL←
 -S1  := EN
 -S3  := SG
 *S5  := THE S6
 -S6  := OX
 -S6  := BUSH
 -S6  := FOX
 -S6  := DOG
 -S6  := CAT
 *S9  := S5 S10
 -S10  := ES
 -S10  := S
 *S12  := S5 S1
                       1    2  ES
T1: THE BUSH S10  =:  THE BUSH ES   (1)
     1    2   3        1    2   3

                      1   2  S
T2: THE DOG S10  =:  THE DOG S   (1)
     1   2   3        1   2  3

                     1   2 EN
T3: THE OX S10  =:  THE OX EN   (1)
     1   2  3        1   2  3

                     1   2  S
T4: THE DOG S1  =:  THE DOG S   (1)
     1   2   3       1   2  3


11: THE CAT S←
-PARSED OK-

12: THE BOOK S←

GIVEN ILLEGAL:
    THE FOX EN
CORRECTION:
.1: THE FOX ES←
CAN YOU SAY:
.2: THE CAT ES
NO←
CORRECTION:
.3: THE CAT S←
CAN YOU SAY:
.4: THE CAT EN
NO←
CORRECTION:
.5: THE CAT S
CAN YOU SAY:
.6: THE BOOK EN
NO←
```

```
13: *TYPE FULL←
 -S1  := EN
 -S3  := SG
 *S5  := THE S6
 -S6  := S16
 -S6  := OX
 -S6  := DOG
 -S6  := CAT
 *S9  := S5 S17
 -S10  := ES
 -S10  := S
 *S15  := THE BOOK S10
 -S16  := BUSH
 -S16  := FOX
 -S17  := S10
 -S17  := S1
                                1   2 EN
T3: THE OX S17   =:  THE OX EN   (1)
     1   2  3         1   2  3

                                 1   2  S
T4: THE DOG S17   =:  THE DOG S   (1)
     1   2   3         1   2  3

                                 1   2   ES
T11: THE S16 S17 =:  THE S16 ES   (1)
      1   2   3        1   2   3

                           1 S
T12: S5 S17   =:  S5 S   (1)
      1  2          1 2


13: THE RAT S←
CAN YOU SAY:
.1: THE RAT ES
NO←
CORRECTION:
.2: THE RAT S←
CAN YOU SAY:
.3: THE RAT
YES←

14: *TYPE FULL
 -S1  := EN
 -S3  := SG
 *S5  := THE S20
 -S10  := ES
 -S10  := S
 *S15  := THE BOOK S10
 -S16  := BUSH
 -S16  := FOX
 -S17  := S10
 -S17  := S1
 *S19  := S5 S17
 -S20  := CAT
```

```
-S20 := OX
-S20 := S16
-S20 := DOG
-S20 := RAT
```

```
                     1   2 EN
T3: THE OX S17  =:  THE OX EN  (1)
     1   2  3        1   2  3

                       1   2  ES
T11: THE S16 S17  =:  THE S16 ES  (1)
      1   2   3        1   2   3

                1 S
T16: S5 S17 =: S5 S  (1)
      1  2      1 2
```

3.7 Intermediate Grammars of English Before and After Recycle

The grammar in 3.7.1 existed just before a recycle. After a reanalysis that included additional machine generated queries of the informant, the grammar of 3.7.2 was produced.

3.7.1 Before Recycle

```
*TYPE FULL←
 *S1  := S5 S13
 -S2  := NEED
 -S2  := TAKE
 -S2  := WANT
 -S2  := SEE
 *S3  := NOW S1
 -S4  := GIRL
 -S4  := CAT
 -S4  := BOOK
 -S5  := YOU
 -S5  := I
 *S6  := S1 NOW
 *S7  := S5 NEED ED S15 S4
 *S9  := TOMORROW S5 S17 S13
 *S10  := S5 WILL S13 TOMORROW
 *S11  := S5 SHALL S13
 *S12  := S5 S2 TO S13
 *S13  := S16 S15 S4
 -S14  := SG
 -S14  := ED
 -S15  := A
 -S15  := THE
  S16  := S2
  S16  := S15 S4 S2 S14
 -S17  := SHALL
 -S17  := WILL
 *S18  := YOU SHALL TAKE THE CAT TOMORROW
 *S23  := S30 S14 S15 S4
 *S26  := S5 TOOK S15 S4
 *S27  := S30 S14 S15 S4
 -S28  := SHE
 -S28  := HE
  S30  := S28 S2
  S30  := S5 S16
                              1 * TOOK  4   5
T1: S5 TAKE S14 S15 S4   =:  S5 * TOOK S15 S4   (1)
     1    2   3   4   5       1 2   3    4   5

                             1   2  *  4    5
T2: I NEED SG THE BOOK   =:  I NEED * THE BOOK   (1)
    1   2   3   4    5       1   2  3  4    5

                            1 * TOOK 4  5
T3: HE TAKE ED A CAT   =:  HE * TOOK A CAT   (1)
     1    2   3 4  5        1 2   3  4  5
```

3.7.2 After Recycle

```
28: *TYPE FULL←
 *S1  := S22 S6 ED S4 S21
 *S2  := S22 TOOK S4 S21
 -S4  := THE
 -S4  := A
 -S5  := BOOK
 -S5  := CAT
 -S6  := WANT
 -S6  := NEED
 *S7  := S10 S8 S4 S21
 -S8  := TAKE
 -S8  := S6
 -S8  := SEE
 *S9  := NOW S7
 -S10  := YOU
 -S10  := I
 *S11  := S7 NOW
 -S12  := HE
 -S12  := SHE
 -S12  := S10
  S14  := S4 S5
  S14  := S12
 *S15  := TOMORROW S12 S23 S19
 *S16  := S14 S23 S19 TOMORROW
 *S17  := S22 S23 S19
 *S18  := S10 S6 TO SEE S36
 *S19  := S8 S36
 *S20  := S45 S8 SG S36
 -S21  := S5
 -S21  := GIRL
  S22  := S4 S21
  S22  := S14
 -S23  := SHALL
 -S23  := WILL
 -S34  := ME
 -S34  := HER
  S36  := S14
  S36  := S34
  S36  := S22
  S45  := S4 S8
  S45  := S22
```

```
                    1   2 *
T2: S10 S6 SG  =:  S10 S6 *  (0)
     1   2  3       1   2 3

                       1  2  3 ME
T3: HE S8 SG S12  =:  HE S8 SG ME  (1)
     1  2  3  4        1  2  3  4

                          1  2  3  4 HER
T4: S4 S5 S8 SG S12  =:  S4 S5 S8 SG HER   (1)
     1  2  3  4  5        1  2  3  4  5

                            1   2  3  4  HER
T5: S10 S6 TO SEE S12  =:  S10 S6 TO SEE HER  (1)
     1   2  3  4   5        1   2  3  4   5
```

4. Discussion of Methodology and Future Plans

At least the first author of this paper is a logical positivist. Chomsky claims that logical positivism died many years ago [1], and has introduced the rationalism of Liebnitz and Descartes as an alternative scientific methodology. The logical positivist or even simple empiricist view assumes that there are only two sources of knowledge, observation and deduction. The rationalist view adds a third source, the acceptability of any idea that is clearly and distinctly formulated and seems true, even though it is empirically unverifiable and not to be derived by deduction. The logical postivist argues that a hypothesis for which no test can be formulated is 'meaningless'. Should a test be possible but not yet carried out, the status of the hypothesis is 'indeterminate.' After successful completion of a test one can then call it 'true' or 'false', but with the proviso that the truth or falsehood of the hypothesis is no more significant than the operations used in carrying out the test.

From the logical positivist point of view, any models that equally account for observed phenomena and yield the same predictions are equally valid, even if their internal structures differ radically. E. F. Moore has proven that two finite state automata can be input-output equivalent and yet have no isomorphisms in their internal structure [20]. He suggests that an engineer should build the cheaper one -- the equivalent of Occam's Razor.

It also follows that, given a model that accounts for all observed phenomena and yields perfect predictions, one is not entitled to believe that the internal structure of that model has any real correspondence with the structure of the real world phenomena it models. However one may, for convenience think in terms of the internal model structure for the personal comfort of an explanation, and, so long as one obtains valid predictions, the problem of isomorphism can be ignored.

What this means for linguistics is that the distinction between descriptive adequacy and explanatory adequacy disappears. The notion of explanatory adequacy, if adhered to, rises to the level of philosophy and poetry, and arguments for grammars made in its name have a methodological status equivalent to literary criticism. The distinction between models of competence and models of performance also disappears. From a logical positivist point of view their are only models of competence and the observation that some models may yield better and more accurate predictions about the real linguistic behavior of people than others. The point is that if one cannot claim internal isomorphism between a performance model and observed reality, the competence-performance distinction is vacuous.

Consider an improved version of AUTOLING that incorporates in its grammars phonological, morphological and semantic data. From the logical positivist point of view, if the grammars are accurate from descriptive point of view, then the program can be viewed as a satisfactory automated fieldworker, and, at the same time it can be viewed as an adequate model of human language learning.

However, the first author of this paper has no axe to grind; he appreciates literary criticism, poetry and philosophy and is very fond of transformations and transformational theories - so much so that he would like to provide the current field of transformational theory with a relatively sound empirical basis. One means of attaining this is through demonstrating the existence of workable methods for obtaining grammars. (The logical positivist mutters that if one has a theory whose basic units have no empirical verifiability one is in trouble.) Another tack is to build automated learning models that produce exactly the kinds of grammars that various transformational theorists like and write . One then has powerful analytic tools for viewing the assumptions and logical consistency of otherwise untestable models.

Future plans for the AUTOLING system include, most immediately, the addition of a morphological learning component and, eventually, a phonology learning component. To build a morphological analyzer that accurately accounts for the semantics of the segmented units, the incorporation of some system of universal semantics seems a logical necessity. We plan to add a model using universal semantic features; an essential part of the analytic heuristics will then involve the determination of the distinctive semantic features for a given language.

The current syntactic learning program, with its lack of restrictions, does not produce a deep structure that would satisfy most transformational theorists. A grammer with deep structure plus transformations could be obtained rather easily if a human linguist first entered a base component for a language into the grammar by hand, and then set the program to randomly generating outputs from that grammar, asking for validity checks and corrections. The program would then learn transformations to take the human-supplied deep structure into surface structure.[1]

For the program to learn a deep structure base component satisfying to many transformationalists, the incorporation of a highly structured universal semantic system appears to be a logical necessity. The first author can think of no other way of providing the program with the necessary constraints on the formulation of the phrase structure component. This requirement is analytic, and reflects what seem to be internally derived logical necessities of recent transformational models rather than any

[1] A user might prime the program with a good grammar for a core subset of a language and use the system to learn the rest of the grammar adding on to a basic framework that pleases the human fieldworker. The authors, of course, are interested in modelling theories rather than in applications, but are happy to provide spin-off tools for humans.

inate properties of human beings.

Actually, the first author, a logical postivist, has come to 'believe' that people acquire some kind of semantic distinctive features in their internal models during the course of language learning. The evidence is somewhat empirical, and is based on problems of testing the classification of new morphological units that are added to a grammar-in-construction. It is not possible to perform exhaustive tests of all newly formulated rules because of time constraints; the required amount of exhaustive testing grows by some factorial function of the number of rules. The current version of AUTOLING does much judicious spot-checking, but is often led into lengthy recycles because of bad rules that slip past the tests. One doubts that humans, in their language learning, reformulate their grammars as drastically and as often as AUTOLING. One suspects that exhaustive testing of new lexical items is avoided through powerful heuristics that make syntactic inferences on the basis of semantic criteria. A future version of AUTOLING will incorporate such heuristics.

References:

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

2. Feldman, J. A., First Thoughts on Grammatical Inference. Artificial Intelligence Memorandum No. 55, Computer Science Department, Stanford University, August 1967.

3. Feldman, J. A., Some Decidability Results on Grammatical Inference and Complexity. Artificial Intelligence Memorandum, Computer Science Department, Stanford University, September 1969.

4. Feldman, J. A., Gips, J., Horning, J. J. and Reder, S., Grammatical Complexity and Inference. Technical Report No. CS 125, Computer Science Department, Stanford University, June 1969

5. 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.

6. 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).

7. (Garvin, Paul L.), Computer-Based Research on Linguistic Universals. Bunker-Ramo Corporation Quaterly Progress Reports, under contract NSF576, Series GO96-8U3, Canoga Park, California, 1967-70.

8. 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.

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

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

11. Hays, D. G., Introduction to Computational Linguistics, American Elsevier, New York, 1967, p. 180. The statement is made in elaboration of ideas expressed in: Garvin, Paul L."Computer Participation in Linguistic Research." Language, Vol. 38, No. 4, October-December 1962.

12. Hockett, C., Review of E. H. Lenneberg's "Biological Foundations of Language" in Scientific American, November 1967.

13. Hockett, C., Review of S. Lamb's "Outline of Stratificational Grammar" in International Journal of American Linguistics, Volume 34, No. 2, April 1968.

14. Horning, J. J., A Study of Grammatical Inference. Technical Report No. CS 139, Computer Science Department, Stanford University, August 1969.

15. 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.

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

17. Klein, S , Fabens, Herriot, Katke, Kuppin & Towster, "The AUTOLING System." University of Wisconsin Computer Sciences Department Technical Report No. 43, September 1968.

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

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

20. Moore, E. F., "Gedanken-Experiments on Sequential Machines," in Automata Studies, edited by Shannon & McCarthy, Princeton University Press, 1956.

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

22. 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.

23. Uhr, L., "Pattern-String Learning Programs." Behavioral Science, Vol. 9, No. 3, July 1964.

24. Wells, R., "Immediate Constituents." Language, Vol. 23, pp. 81-117, 1947.