A CONTEXTUAL RECOGNITION SYSTEM FOR FORMAL LANGUAGES*

Stephen C. Darden
The University of Texas at Austin
Austin, Texas

Introduction

This paper describes work in progress aimed at obtaining a practical recognition system by filtering the noisy output of an inexpensive character classifier through multiple stages of context analysis to produce high-quality final output. The system is designed to process the text of hand-printed formal languages, such as programming languages and mathematical notation. The system is syntax-directed in that a contextual processor for a particular language is constructed by a one-time, algorithmic analysis of the language grammar. The resulting analysis engine massages the classifier output by successive vocabulary matching, name-space"" clustering, and syntactic analysis to produce as output a syntactically valid interpretation of the text, with as many errors corrected as possible. A more complete description of the system can be found in Darden.

Figure 1 depicts the process by which the system "learns" the language that is to be recognized. This knowledge comprises a convenient representation of the grammar and a dictionary of basic symbols which occur in the grammar (e.g., DIMENSION, CONTINUE, ...).

Figure 2 depicts the resulting analysis engine, which accepts as input the output of a single-character classifier, in the form of a list of lists of alternatives for each character. This system is designed to exploit three types of redundancy exhibited by the class of languages under study:

vocabulary redundancy results from the grammatical specification of a set of strings (basic symbols) which have a high *a* priori probability of of occurrence;

statistical redundancy results from the high probability of multiple occurrence of members of the name-space of such languages;

structural redundancy results from the grammatical constraints on concatenation of well-formed strings.

All three types of redundancy are necessary to successfully clean-up garbled text. Indeed, there are certain errors (such as recognizing XMAX as YMAX) that can only be detected by considering semantic redundancy (resulting from the property of these languages that only a fraction of the syntactically valid strings are meaningful, either a_ priori or in context). It is hoped that uncorrected-error rates on the order of 1% can be achieved without introducing the staggering complexities of semantic analysis.

Related Work

A lucid discussion of past approaches to the use of contextual post-processors to improve the performance of pattern classifiers is contained in the recent paper by Duda and Hart.** We shall not duplicate their efforts here.

The use of an error-correcting parser as a contextual post-processor was first suggested by Irons.* The FORTRAN context analyzer of Duda and Hart is a specialized instance of such a processor which has proven the soundness of the idea. They have developed a formal decision-theoretic solution to the utilization of context and have shown that the application of syntax analysis does not do violence to the formal approach. Our own work in this area was well underway before we became familiar with their investigation.[2,3] We have naturally been influenced by their contributions and have noted the points in the body of this report where our development has profited from their efforts.

The unbounded-context parsing system described in this report is a major departure from classical parsers such as that described by Irons, which do not take into consideration sufficient context to make reliable error-correction decisions. The parsing system most similar to ours is reported by Unger. In the final chapter of this paper we discuss possible extensions to the unbounded-context parser which derive from the quick-checks used by Unger.

A Normal Form for Context-free Languages

This section describes a particular representation for context-free (CF) grammars that has been developed to satisfy the needs of an error-correcting parser.

When a human context-analyzer is given the task of correcting a sample of garbled text

This work was supported in part by a grant from Texas Instruments Incorporated and in part by Public Health Services Grant GM15769 (NIH).

The collection of names, occurring in the text, which are supplied by the author of the text, i.e., identifiers.

* Following Duda and Hart, [2] we refer to this data structure as a "P-list"; the alternative lists are called "L-lists." Each alternative list is composed of a list of pairs (l.c) where i is a character and c is a confidence proportional to the logarithm of the probability that £ is, in fact, the correct character.

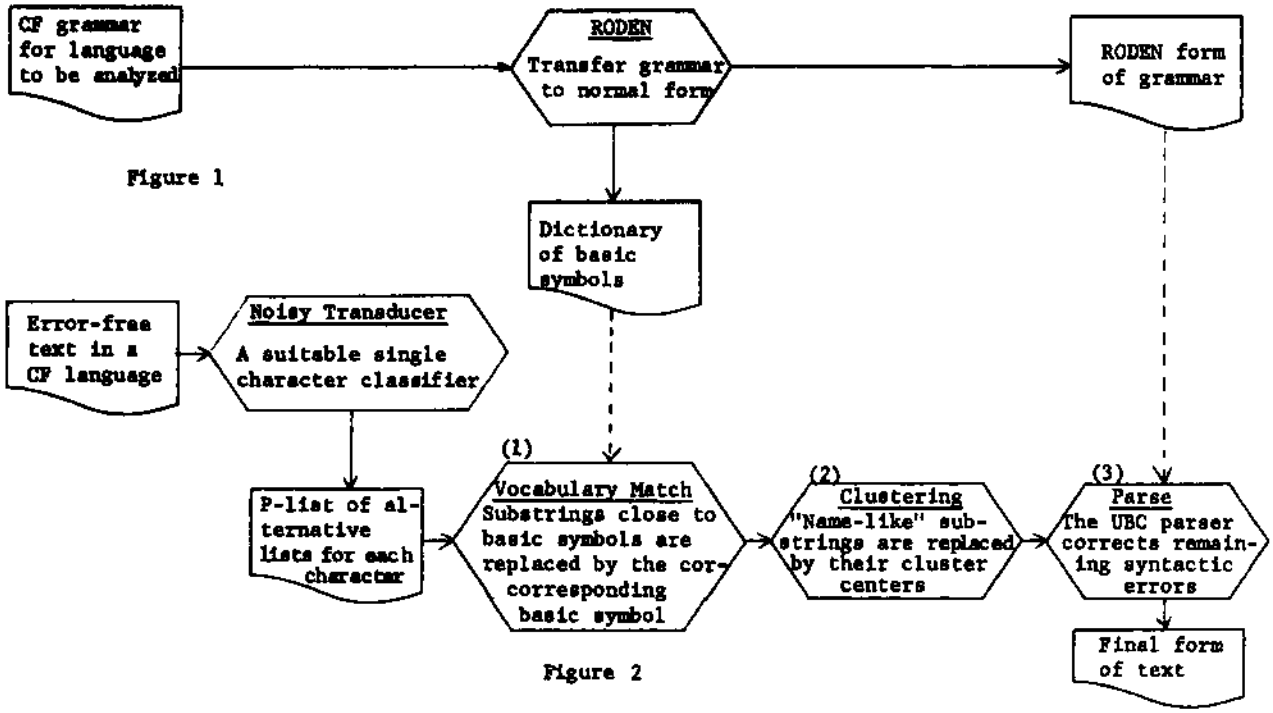

Figure 1

Figure 2

(e.g., Figure 3), he is usually able to immediately select the appropriate linguistic pattern (e.g., ⟨for statement⟩ or ⟨conditional statement⟩ in the example) to be used in a more detailed analysis of the text. He abstracts from all the noise, the presence of the for __ := __ step __ until __ do, and if __ then __ else patterns. The main purpose of the normal form is to give to the parser this global view of syntactic structures.

```
TOR I := 1 STEP ; ONTIL N DO V[I] := 0;
EUP T ;- / I2G9 I VM22F H OU YL1I ;- D:
FDQ J ,  , 6J88 ) YHJT6 V GD UC2T ,  U,
K A F    I 5567 / KV5J[ W Q  JIT2    O
5   S    ) 8 FZ 1 UW FG M 8    J,    P
    1      =       SE          F     Q
    2              1           S

 TF Y = R TH=N  CO2O IT ELSE X := X+1
 SK X - P 2K8V  GUTU F2 6GIF Y ;= Y*/
 15 *   Q J*6W  FDJD SF FF6= * ,  *=,
 28 4   A 5MFM  6 5  15 =E5G 4    4TI
 IT        NEH  0    2  6688         )
 J          4G        T  8[ 6
 F                    J
```

Figure 3

The casual reader will probably wish to skip the remainder of this section, referring back as needed from the discussion of syntactic analysis. The normal form will hereafter be abbreviated as RODEN-form (for Regular, Operator, Disjunctive, Embedding, Normal). In the following, it is assumed that the reader is familiar with the rudiments of context-free languages. The survey by Floyd[6] is recommended as an introduction, and it has a comprehensive bibliography.

A RODEN-form grammar G comprises a set of rules $A \rightarrow \alpha$ in 1-1 correspondence to a vocabulary of non-terminal symbols (see example G_2 below). The genatrix, A, is a non-terminal. The pattern, α, is an expression constructed from the vocabulary of non-terminals together with the vocabulary of basic symbols, and the operations of concatenation (or conjunction, denoted by juxtaposition), disjunction (denoted by |), replication (or finite closure, denoted by superscript *), and selection (denoted by superscript ° ; selection is identical to the disjunction of the empty string with the single operand of the selection operator).

Both left and right recursion have been eliminated in favor of replication by iteration. In addition, each pattern α is in operator form; i.e., there is no derivation $\alpha \overset{*}{\Rightarrow} \emptyset U_1 U_2 \omega$ where U_1, U_2 are non-terminal symbols. Finally, each pattern is in disjunctive normal form; i.e., all disjunction operators occur at the highest level, and the disjuncts are ordered by descending redundancy (presence of head or tail basic symbols, minimum number of basic symbols).

An example should serve to give the flavor of the RODEN-form representation. G_1 is the usual CF-grammar for ALGOL arithmetic expressions; G_2 is the equivalent RODEN-form grammar.

Observe that (a) disjunction, and its associated branching and backtracking, has been almost eliminated; (b) subgoals are uniquely determined by delimiting basic symbols (thus the operator-normal form requirement); and (c) if self-embedding subphrases were properly suppressed, the parser could immediately assign a complete parse at the current level using all occurrences of the

pattern's basic symbols, in the entire text under consideration.

G_1

⟨aexp⟩ → ⟨simple aexp⟩|⟨if clause⟩⟨simple aexp⟩
else ⟨aexp⟩

⟨if clause⟩ → if ⟨Bexp⟩ then

⟨simple aexp⟩ → ⟨term⟩| ± ⟨term⟩|⟨simple aexp⟩
± ⟨term⟩

⟨term⟩ → ⟨factor⟩|⟨term⟩*/÷ ⟨factor⟩

⟨factor⟩ → ⟨primary⟩|⟨factor⟩↑⟨primary⟩

⟨primary → ⟨unsigned number⟩|⟨variable⟩|
⟨function designator⟩|(⟨aexp⟩)

G_2

⟨aexp⟩ → [if ⟨Bexp⟩ then ⟨simple aexp⟩ else]* ⟨simple aexp⟩

⟨simple aexp⟩ → [±]° ⟨term⟩ [± ⟨term⟩]*

⟨term⟩ → ⟨factor⟩ [*/÷ ⟨factor⟩]*

⟨factor⟩ → ⟨primary⟩ [↑⟨primary⟩]*

⟨primary⟩ → (⟨aexp⟩)|⟨function designator⟩|
⟨variable⟩|⟨unsigned number⟩

For example, the string

X + Y + (N + 1) * M

cannot be parsed directly as a ⟨simple aexp⟩ because the embedded + would lead to the assignment of ⟨term⟩ to both (N and 1) * M. Rather, the parser should see the string as

X + Y + * M
(N + 1)

where the embedded substring is not "visible" at the current level.

In order to characterize this suppression process we need to introduce the concepts of admissible basic symbols and their associated minimal cover lists. Briefly, if A → α, a basic symbol t is said to be admissible with respect to the genatrix A if either

(a) t does not occur in α or

(b) for each instance of t ∈ α one of the following two conditions is satisfied:

(1) α is of the form tØ (or Øt). That is, a corresponding instance of t in any valid ground-derivation* σ of A is uniquely determined by matching t to the head (or tail) of σ.

(2) α is of the form Øtω. Then either

(i) there is no subphrase B ∈ α such that $B \overset{*}{\Rightarrow} \eta t \mu$ or

(ii) if B occurs in α, and if $B \overset{*}{\Rightarrow} \eta t \mu$ is a valid ground-derivation, then this derivation must have the form

$$B \overset{*}{\Rightarrow} \ldots X \ldots \rightarrow \ldots \underset{t}{[} \ldots \underset{t}{]} \ldots \overset{*}{\Rightarrow} \ldots \underset{t}{[} \ldots t \ldots \underset{t}{]} \ldots$$

where X is a non-terminal and $[_t \neq]_t$ are basic symbols distinct from t. The pair of brackets $[_t ,]_t$ are said to cover the lower-level occurrence of t. If, however, t = [(or t =]), then α must satisfy the additional condition $\alpha = \emptyset t \omega = \eta [_t \lambda]_t \mu$ where B ∈ λ and t does not appear in any ground-derivation of either η or μ.

When an admissible basic symbol t is of type (ii), the set of bracket pairs, which uniquely delimit lower-level instances of t, is called the minimal cover list for t. For example, in grammar G_2 we can show that + is admissible with respect to ⟨simple aexp⟩, by condition (ii). We observe that B in this case can only be ⟨term⟩ and that there are only three ways to obtain a + in a ground-derivation from B:

⟨term⟩→...⟨factor⟩...→⟨primary⟩... → ...((aexp))
... → ...⟨function designator⟩
... → ...⟨variable⟩

We shall consider only the simplest case:

((aexp)) → (... ⟨simple aexp⟩ ...)
→ (... ⟨term⟩ + ⟨term⟩ ...)

where we see that the + in (... ⟨term⟩ + ⟨term⟩ ...) is indeed "covered" by the pair of brackets (...). In a similar fashion, the other two cases lead to the discovery of the additional pair of brackets [ ].

We shall say that a basic symbol is admissible in a grammar if it is admissible with respect to every genatrix of the grammar.

When every basic symbol of a grammar is admissible, a valid parse is obtained by preprocessing every text segment for a well-formed nest of minimal covers for all the basic symbols of a pattern, before associating the basic symbols to corresponding occurrences in the text. Each covered segment is pushed down one level to be analyzed by subgoals descended from the current level of analysis. The existence of such nests also provides another powerful consistency check on the text, permitting the cover processor to correct many errors involving the covering brackets themselves.

*$A \overset{*}{\Rightarrow} \omega$ is said to be a ground-derivation if ω is composed entirely of basic symbols.

Our parser requires that every basic symbol of a RODEN-form grammar must be admissible. However, in general, CF-grammars may contain non-admissible symbols. Furtunately, for the CF-grammars in common use, this is not a problem. All basic symbols in FORTRAN and LISP are admissible. In ALGOL, the set of symbols jLf, then, else, for, :■ do are inadmissible due to the construction which permits

if-then-else for- :« - do JLf - then-else ...

In actual practice, this construction almost never appears. Indeed, in ALGOL-68, the problem has been eliminated by providing conditional statements with a closing basic symbol: if-then-else-fi.

In summary, the RODEN translator transforms a broad class of CF-languages to RODEN-form; in the process, basic symbols are examined for admissibility and their corresponding lists of minimal covers are associated with the proper genatrices. Additionally, lists of basic symbols occurring in the final form of the handle are associated with the corresponding genatrix, and a dictionary of all basic symbols is compiled. This dictionary is then analyzed for any entries which can be "extended" (hence the term extended dictionary). That is, each word is examined to see if every occurrence of this word, in strings of the language, is contained in some larger unique context. For example, ignoring spaces, this leads to the replacement, in the dictionary, of all LISP functions (fun) by ((fun), of all FORTRAN verbs (verb) by :(verb), of FORMAT by FORMAT(, etc. (: is used as a column 7 indicator, ; is an end of line in processing FORTRAN).

III. Simulation of a Single-character Classifier

Data for testing the context analysis system are provided by a computer-simulated character recognizer. The input to the simulator is error-free text from the language to be investigated. The garbled output is in the form of a P-list for the context analyzer.

The operation of the simulator is controlled by a confusion matrix (constructed bo approximate the error maps reported by Duda and Munson^ for their TOPO 2 program), a noise factor, and a "consistency" factor.

On the assumption that classifiers can be constructed so that the correct character will almost always appear among the first few entries of the output alternative list, the confusion matrix has about six non-zero entries for each character.

The noise factor controls the undetected -error rate of the simulated classifier. The consistency factor is a measure of the probability that, given two alternative vectors for the same character, a character which occurs in one vector will appear in the other vector. The reliability of the inter-string distance measure (see 4.2), used by the system, improves as the mutual consistency of alternative vectors increases.

IV. Context Analysis

The first two phases of the context analysis system are aimed at reducing the combinatorial complexity of the task to be performed by the syntax analysis phase. Since, In principle, all except name-space errors can be corrected by syntax analysis, care is taken to ensure that the dictionary matching and clustering do not Introduce errors; however, neither is foolproof.

4.1 Extended-dictionary Matching

The strategy of extended-dictionary matching (box 1, Figure 2) is similar to the table look-up method used by Vossler and Branston.® Spaces are treated as error-free characters, and each substring of length n delimited by spaces is processed in turn. The substring is matched against all possible dictionary entries of length n, n-1, ..., k until the first entry is found which is closer than a fixed minimum distance to a (possibly proper) substring of the current object P-list. The lower limit, k, on the length of entries to be matched is established experimentally. Presently, with up to 35% noise and 50% consistency (see Section III) in the simulated data, a value of k = 2 has been found to introduce no erroneous matches in the processing of LISP, ALGOL, and FORTRAN texts.

The distance, d, between a dictionary entry

$$E: \ell_1 \ell_2 \cdots \ell_m$$

and a P-list of the same length

$$\sigma: \ (((c_{11},v_{11})\ldots(c_{1m_1},v_{1m_1}))$$
$$((c_{21},v_{21})\ldots(c_{2m_2},v_{1m_2}))$$
$$\ldots$$
$$((c_{n1},v_{n1})\ldots(c_{nm_n},v_{nm_n}))\ )$$

is given by

$$d = n/\sum_{i=1}^{n}\sum_{j=1}^{m_n} v_{ij}\delta(\ell_i,c_{ij})$$

where

$$\delta(a,b) = \begin{cases} 0 & \text{if } a \neq b \\ 1 & \text{if } a = b \end{cases}$$

provided that every ℓ_i corresponds to a c_{ik} for some k; otherwise, $d = \infty$. If a satisfactory match is found, the text P-list is modified by replacing the corresponding L-lists by singleton L-lists containing the letter of the dictionary entry paired with the highest possible confidence. This replacement is a level zero suppress operation.†

---

*Recall that the extended dictionary contains the basic symbols of the language (e.g., begin, end, ...).

†In each case, the deleted entries have been

For example, the P-list

```
6  S  E  T  8          ( S E T Q
I  I  =  5  B  becomes
C  6  G  2  F
(  5  6     Q
L     F
```

where the associated confidences are suppressed for clarity.

4.2 Name-space Clustering

The clustering procedure (box 2, Figure 2) employed by the system is related to the methods employed by Duda and Hart-3, but necessarily differs in several ways due to the requirements of handling a variety of languages. The strategy is based on the assumption that a sufficiently sensitive clustering method can discriminate names without a priori knowledge of the context in which names are likely to occur. This assumption obviously depends on both the length and average number of occurrences of names in the text. It is more important for the algorithm to identify at least one instance of all of the names than to locate all of the occurrences of each name. Those occurrences which are overlooked will very likely be found by the pattern associating routines which use the dictionary of known names compiled during this pass.

Typical performance of the method is indicated by the results of clustering analysis on a 272-character ALGOL program. The output of the simulated character classifier exhibited an uncorrected error rate of 33.1% and a consistency factor of 48.8%. The results are given in Table 1.

Name	Number of Occurrences	Number Identified	Unified Spelling
I	8	6	I
J	6	2	J
K	5	2	K
N	5	5	N
RAD	3	3	QAD
TRY	2	2	TR2
BASE	3	3	BASE
FIND	2	2	FING
PERM	4	2	PFRM
ORDER	5	4	ORDER

Table 1

descended from an auxiliary pointer on the L-list. This auxiliary list is used to restore the original structure if the selected entry is later found to be invalid and has the structure

$$((L_1, \text{L-list})\ (L_2, \text{L-list})\ \ldots\ (L_n, \text{L-list}))$$

where L_1. is the integer-level number at which the decision to suppress the associated L-list was made. We shall refer to this operation as the suppress operation.

Thirty-one out of forty-three total instances of all names were identified, including at least one instance of every name. After syntax analysis two instances of the name 'I' were incorrectly identified as 'J'; all other instances of names were correctly recognized.

4.2.1 Sample collection. The clustering algorithm processes the entire text for "name-like" substrings, constructing as many groups as needed of identical length samples. Presently, a P-list is treated as a possible name if

(a) it contains no spaces,
(b) it contains no characters classified as delimiters (this information is input along with the RODEN-form grammar) whose confidence exceeds a fixed value,
(c) a letter is among the alternatives for the first position,
(d) no longer P-list satisfying (a)-(c) contains it.

Since most formal languages define names in this way, this definition is built into the clustering program. It is clear that this sampling procedure will collect a number of strings which are not names at all, but substrings of names, composites of names and other symbols, etc. The assumption is that these undesirable strings will form degenerate singleton clusters of their own and will be ignored in the unification step at the end of this procedure. In the example described by Table 1, six such strings were collected by the sampling procedure; all formed degenerate clusters and were discarded.

4.2.2 Clustering. The clustering procedure is applied to each group of identical length samples in turn. First, a set of "representatives" is selected from the sample. The procedure for locating the representative samples is similar to that used by Casey and Nagy9 to select an initial set of centers for the clustering stage of their recognition system. The first sample is chosen as the first representative. The sample having maximum separation (defined below) from the first is taken as the second representative. Similarly, the remaining representatives are chosen to be those with the maximum minimum separation from those already selected. This procedure is iterated until a sample is selected whose separation is less than a fixed minimum distance. It is assumed that at this point an instance of each name of length n has been located.

The success of this procedure is in large measure dependent on the performance of the distance function, defined for two P-lists U and V where

$$U = (((a_{11}.u_{11})(a_{12}.u_{12})\ldots(a_{1m_1}.u_{1m_1}))$$
$$((a_{21}.u_{21})(a_{22}.u_{22})\ldots(a_{2m_2}.u_{2m_2}))$$
$$\ldots$$
$$((a_{n1}.u_{n1})\ldots(a_{nm_n}.u_{nm_n}))$$

$$V = (((b_{11}.v_{11})\ldots(b_{1k_1}.v_{1k_1}))$$
$$\ldots$$
$$(((b_{n1}.v_{n1})\ldots(b_{nk_n}.v_{nk_n}))$$

$$d(U,V) = n/\sum_{i}^{n}\sum_{j}^{m_i}\sum_{\ell}^{k_i}(u_{ij}+v_{i\ell})\delta(a_{ij},b_{i\ell})$$

where

$$\delta(a_{ij},b_{i\ell}) = \begin{cases} 0 \text{ if } a_{ij} \neq b_{i\ell} \\ 1 \text{ if } a_{ij} = b_{i\ell} \end{cases}$$

If in any position no two characters correspond, then d(U,V) ■ . If, in general, two alternative vectors for the same character have a reasonable high consistency, then this should be a richer distance measure than a simple sum of the highest confidence corresponding characters.

The final stage of the clustering procedure is very simple: each remaining sample is assigned to the cluster with the closest representative.

4.2.3 Unification. When all clusters of a given length have been established, a unification procedure is applied to all but singleton clusters to select the highest confidence common spelling for each. Then, as in the dictionary matching, each cluster member is replaced in the text P-list by this common spelling. The deleted L-list entries are suppressed at level zero.

While there are certainly many refinements which can be incorporated into this procedure, our philosophy is to apply effort where the greatest returns appear to lie, and this is in the syntactic analysis phase. The only improvements currently planned for the clustering phase are (a) a refined distance measure which will attempt to detect the presence of prefixes (suffixes) (e.g., xmax, ymax) and give relatively more weight to the contribution of the prefix (suffix) to the distance; and (b) the dynamic computation of the center of a cluster as new members are added.

4.3 The Parse Graph

In order to follow the next section on syntactic analysis, it is necessary to have in mind the parse graph which represents the current state of that analysis. The overall structure of the parse graph is, naturally, a tree. The nodes of the tree have a branching structure which corresponds to the RODEN-form meta-operator at the node (thus there are conjunctive, disjunctive, iterative, and selective nodes). In addition, there are phrase nodes (e.g. (statement)) and terminal nodes (e.g., begin). Associated with each node are a number of indicators which describe the state of the parse represented by the subgraph descended from the node. Among the possible entries are those which give the value of the basic symbol match, the number of characters in basic symbols which selected this node, and the level number of the parent cover for covering brackets. The entries relevant to error recovery are: left- and right-extensibility indicators, left- and right-truncation indicators, and an entry which reflects the detection of an Invalid covering bracket assignment.

4.4 Syntactic Analysis

By combining the unbounded-context parser and a particular RODEN-form grammar we obtain an error-correcting parsing system which is nearly free of back-tracking. Through hierarchical consistency checking, the parser is able to discard fruitless paths inexpensively and is able to correct many errors before exploring the detailed structure of the text. At each level the parser associates with the current object P-list (a) an instance of the current pattern (p) on the basis of corresponding basic symbols. The method employed causes the pattern to be self-positioning. This property, together with the fact that all basic symbols of the pattern have been mapped with a high average confidence to the P-list, improves the parser's probability of success in selecting the correct recovery steps when errors are detected.

4.4 1. The basic strategy of parsing. In the following, the parser is considered to be analyzing *a* with respect to the current subgoal G. G may be a phrase (a single non-terminal), a basic symbol, or a skeletal instance of a pattern assigned by a higher-level goal. If G is a pattern (p), then M and S are the associated minimal cover lists, and basic symbol lists, respectively.

If G is a basic symbol, it has already been matched to the text at a higher level (by the association procedure), so the parser simply reports the value of that match.

If G is the phrase (identifier), the parser invokes a special process which attempts to exploit the relatively high a priori probability of occurrence of identifier dictionary entries.

If G is any other phrase, the parser retrieves the pattern, p, associated with G, and matches the pattern to the text in the three steps of covering, association, and resolution, described below.

If G is a pattern, the parser applies itself recursively to each subpattern of p. In the case of disjunctive patterns, the parser returns the first disjunct to achieve a satisfactory match. In the case of conjunctive patterns, failure of a subpattern causes the parser to invoke the recovery procedure appropriate to the type of failure. The details of failure recovery are discussed below. On exhausting all subpatterns of p, the parser invokes a scoring procedure to evaluate the success of the analysis for G. Errors which have been detected in disjunctive structures are reported to the parent goal at this point. In conjunctive patterns, comparison of the bounds of the substructure to the bounds of G reveals the presence of L-drop and R-drop errors (see Error Recovery and Correction below).

4.4.2 Analysis of the (identifier) phrase. The special processing of identifiers serves to compensate for the lack of structure of this phrase type, as defined for most languages. Otherwise, virtually any text segment would satisfy the syntax of (identifier). It is assumed that the identifier dictionary contains an instance of every identifier.

First, *o* is matched to all identifier dictionary entries of the same length. If a sufficiently good match is found, the parser reports the value of the match. This failing, the process attempts to determine if a is a head, tail, or proper substring of a valid identifier occurring in the text P-list which includes a. If an identifier is found which matches the P-list in a neighborhood of a, the parser reports the appropriate type of add or drop error.

4.4.3 Matching patterns to the text. As indicated above, when the parser begins analysis of a new subgoal, which is a phrase, the matching of the pattern associated with that subgoal proceeds in three successively more expensive steps, covering, association, and resolution. Each step must be successful before entering the next, and each has its own error detection and correction tactics.

4.4.3.1 Covering. The list, M, of minimal covers associated with p, is retrieved. If M is empty, the algorithm proceeds to the association step. The covering process accepts the text P-list, the bounds of G which delimit the portion of the P-list to be considered, and the cover list M of the form

$$M = ((\ell_1,r_1)(\ell_2,r_2)\ldots(\ell_n,r_n))$$

of n distinct pairs of covering brackets. The covering procedure is given the task of finding the highest confidence well-formed nest of these brackets.

The procedure presently under investigation abstracts from the text P-list a list (called a B-list) which has an L-list for each possible occurrence of any of the brackets. Each such L-list has the entries: the bracket together with its confidence, **(B,U)**, and a null character, 0, representing selection of a symbol other than the bracket, together with a confidence which represents the effect of suppressing the bracket, **(Ø,V)** (note the example below). If there are K brackets on the original L-list, there are **K+1** entries on the new L-list. For single-character brackets, the non-bracket confidence, V, is simply that of the highest confidence non-bracket character on the original L-list. For multiple-character brackets V it the average of the highest-confidence non-bracket characters. This latter measure has not been tested, since the extended dictionary-matching procedure has never failed to correctly classify a multiple-character basic symbol which was also * covering bracket.

The cover procedure selects a nest from the B-list by generating, by descending number of brackets (strings with equal numbers of brackets are ordered by descending total confidence), all possible strings, taking the first well-formed nest.

If a well-formed nest is found, the portions of the P-list bounded by each bracket pair are suppressed one level by combining that portion of the P-list with the new operator, *cover*.

For example, consider processing the P-list

```
P := CJ+I)/12;
8 ;- (]*J,1/):
7 ,  6)=F2  0,
Z    G8TS3  Z
9      , 1  I
       5 2  ]
         T
```

for the covers associated with (arithmetic expression) in ALGOL, where M contains the bracket pairs ( ) and [ ]. We have the B-list

```
Ø,45   Ø,35   ),45   Ø,35
(,30   ],25   Ø,30   ),25
       ),20          -,2
```

The algorithm selects the correct nest (_) _ and transforms the P-list to

```
P :=        / 1 2;
8 ;-(J + I) 1 / 0:
7 ,  8 * J      Z,
Z    , = F      I
9    3 T S
     2   1
         2
         T
```

which is represented in list notation as:

```
(((P.40)...)
 ((SP.100))
 ((:.50)...)
 ((=.60)...)
 ((SP.100))
 (*COVER* 5 (((LP.30))
            ((J.35)(8.15)(,.3)(5.2))
            ((+.45)...)
            ((I.30)...)
            ((RP.45)) )
 ((/.60)...)
 ((1.60)...)
 ((2.35)(Ø.20)(Z.15)(I.3))
 ((;.50)...))
```

The form of a P-list entry for a covered substructure is (*cover* ⟨length⟩ P-list) where ⟨length⟩ denotes the number of character positions in the suppressed P-list. Note that in the new P-list, the four L-lists which contained brackets have a new structure: those containing the selected brackets have been replaced by singleton lists, and the brackets have been suppressed in the other two lists.

4.4.3.2 Association. This is the critical step in unbounded-context parsing. Association establishes the correspondence between an abstract representation of the text, in the form of an

ordered list of principal basic symbols, and an abstract representation of the global structure to be assigned to the text, a RODEN-form pattern.

The success of the method depends on the ability of the association procedure to "spread the risk" of selecting a particular pattern over a sufficiently large number of characters in the text,--thus the need for eliminating recursion and adjacent non-terminals in patterns in order to bring to the surface the principal basic symbols which serve to identify the occurrence of that pattern (for example, note the explicit structure of (aexp) in example G2 of Section II). Another important property of the method is that the pattern is automatically positioned with respect to a. Thus, missing delimiters, which cause the bounds on a phrase to be set incorrectly, are detected by noting that the pattern was successful in matching the interior of the bounded text. Similarly, erroneous delimiters are detected by noting that the pattern has been truncated on one or both ends by an incorrectly positioned bound. For an example of the latter, consider the processing of the (for statement) of Fig. 3: after the first two phases of context analysis, the P-list might have the form

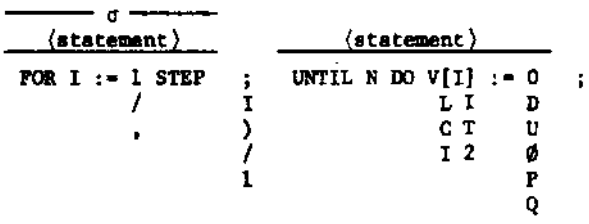

Figure 4

The erroneous ";" has caused a higher-level goal to partition the string into two (statement)'s. The high confidence head match of (for statement) to FOR____:*____will cause the association procedure to immediately report an "R-add" failure (i.e., a delimiter is hypothesized to have been "added" just beyond the right end of a).

The task of the association procedure is to process the bounded segment, a, of the P-list to produce a skeletal instance (N) of the pattern p.

First, a process is invoked to construct from *0* and B, the basic symbol list (W) for p; i.e., W is a list of occurrences of these basic symbols. The elements of W are triplets ((symbol),(value), (pointer)) where (value) is the average confidence that the string found at (pointer) in a is, in fact, the basic symbol (symbol). Symbols found at the head and tail of *a* are tagged with special*-, -4 indicators so that patterns which require head and/or tall symbols can distinguish them.

All possible occurrences of basic symbols of more than one character will be found on W. An unpleasant problem is encountered if all possible single-character symbols are also included: errors may be introduced which will not be detected by the subsequent context-free analysis. For example, in any algebraic language, the symbol which would be selected from the alternative list for some operator, such as *, will be the first operator in the hierarchy which occurs on the list. That is, If a + appears as the lowest confidence entry on an L-list for *, then the expression will parse as $\alpha + \beta$, and the error will never be detected. The following approach seems to circumvent the difficulty: when processing a single-character basic symbol, which is classified as a delimiter, an L-list is selected only if that delimiter has higher confidence than any other delimiter which may appear on the L-list. Delimiters which are missed with this procedure are detected as drop errors in subsequent syntactic analysis and corrected by the recovery procedure.

The second step in the association process has the task of selecting and expanding elements of the pattern p. The match is controlled by the list W, of occurrences of basic symbols. Briefly, the association procedure sequences through W, from left to right. The elements of W serve to select a subpattern of p if a correspondence can be established between the basic symbols of each. Selected subpatterns are appended to the growing skeletal node, N. Iterative and selective patterns, if not selected, are appended with an X (unexpended) indicator in the event that they may be needed in subsequent drop recovery. In rare cases, on exhausting the pattern, a string erroneously recognized as a basic symbol may result in a partially satisfied conjunctive pattern accompanied by unused symbols on W. (E.g., an erroneous do in processing a (for statement) in ALGOL could lead to an unsatisfied :=, with the :- and the correct instance of the jio left on W.) These cases are corrected by sequencing through the pattern (the match is now controlled by the pattern) and W from right to left, searching W for an instance of each unsatisfied symbol in the pattern. If all of these can be satisfied, and W is still not empty, we examine the remaining entries on W for symbols which are both in the proper order (position in *0)* and have higher confidence than a symbol already selected. If such an instance is found, the original symbol is replaced.

If in a conjunctive pattern a head or tail match is obtained, then the right-extensibility (R-add) or left-extensibility (L-add) indicators, respectively, are set at the node. The R-add indicator at a node specifies that the associated pattern is satisfied with the portion of the P-list bounded by G but that the remainder of the pattern can only be satisfied by matching more of the P-list to the right. Thus, if an error has, in fact, been detected, it must be an erroneously recognized basic symbol (an "add"), which caused the right bound of G to be incorrectly positioned. Similarly, the L-add indicator specifies a possible Incorrect left bound.

4.4.3-3 Resolution. If the association process was not able to correct all detected errors, the parser reports the failure to the parent

node. Otherwise, a procedure is invoked, given N and the bounds of G, to set the bounds for the subphrases of N. Then the parser is applied recursively to N. If the recursive call reports an error involving a covering bracket assigned at the current level, the parser restarts the cover, associate, resolve process with the next valid cover nest in the sequence (see "cover conflicts" below).

4.4.4 Error recovery and correction.* The only (syntactically-correctable) errors which remain to be considered are those which cause the incorrect positioning or positing of subgoals. When the parser reports a failure in the analysis of a subpattern, the indicated failure is of four main types, cover-conflict, add-failure, drop-failure, or total failure.

The treatment of failures is, of course, determined by the context of the type of pattern in which the error is detected. In disjunctive patterns, the failure may simply indicate a syntactic dead-end, i.e., incorrect positing of a subgoal. Therefore, error processing is postponed until the entire disjunctive pattern Is scored in hopes that a subsequent disjunct will succeed. This strategy proves to be reliable since the most difficult patterns to satisfy, i.e., the most redundant, are matched first.

4.4.4.1 Add-fallures. An add-fallure occurs when a substring has been erroneously recognized as a basic symbol, at the current or higher level, resulting in the assignment of a phrase to a substring which has been truncated at one or both ends.

The strategy of add-failure correction is to back the failure up the parse graph until

(a) a conjunctive branch point is encountered where an adjacent iterative or selective node can be removed along with the basic symbol which is presumed to be the source of the difficulty; or

(b) a disjunctive branch point is encountered where the parser can postpone the error recovery (the effort could be wasted on a syntactic dead-end).

4.4.4.2 Drop-failures. When a basic symbol is missed by the association process (i.e., a drop error), the result is an improperly bounded subgoal. The error is detected at a lower level, when the subgoal is analyzed, as a left-drop (L-drop) or right-drop (R-drop). The stragegy of drop recovery is to back up the parse graph until

(a) a context is encountered which permits the introduction of a new Iterative phrase to cover the unrecognized text; or

(b) basic symbol which delimited the subgraph containing the drop is encountered, permitting the parser to replace this instance of the symbol with another instance in the unrecognized portion of the text;or

(c) a selective node is encountered, allowing the parser to abandon the recovery effort, or

(d) a disjunctive branch point is encountered, allowing the parser to postpone error recovery.

4.4.4.3 Cover-conficts. Since the cover generation algorithm produces covers ordered by descending total number of brackets, cover errors are detected during an attempt to suppress a basic symbol which is also a bracket belonging to a cover established at level K. When this condition is detected, the parser sets the "request recover indicator" at the current node. As with the other error indicators, this information is backed up the parse graph until the parser either shifts to another successful analysis path at a disjunctive node or reaches the phrase node at level K where the culprit nest was assigned. At this point the parser restarts the cover, associate, resolve process with the next valid nest in the sequence.

4.4.4.4 Total failure. A total failure is usually an indication of a syntactic dead-end or is the first indication of an add-error. The strategy of total-failure recovery is to back up the error indication until either a disjunctive branch point is encountered, permitting the parser to try another analysis path, or a conjunctive branch point is encountered. In the latter case, the recovery investigates three possibilities: first, that the unsatisfied conjunct was an erroneously expanded iterative phrase; second, that one or more basic symbols, which delimit the conjunct, are mispositioned (i.e., a combined add/drop error has occurred); and third, if the conjunct which failed occurs at the head or tail of the pattern, then a higher-level L-add or R-add is assumed and reported to the parent node.

In the first case, we must have a pattern fragment ..aby..., with possibly one of a,y null and p an iterative subpattem. The basic symbols which selected p are suppressed at the current level and the text segment reanalyzed as an aY.

In the second case, we must have a pattern fragment of the form ..atb... where only one of a,p failed. Since we have symmetry, assume that p failed. If the recovery is to succeed, t must be an add-error and must have been dropped in the text bounded by p. Therefore, we suppress t and search for the highest confidence occurrence of a t in the text bounded by $\alpha t \beta$. If found, we reanalyze the text segment with the new assignment. The process is continued until there are no more t's to try or a successful parse is obtained. If errors still remain, which are only head or tail errors, then it is assumed that the third case holds, and the appropriate error is reported.

4.4.5 Scoring the complete node. When the parser completes the analysis of a particular node, the scoring procedure is invoked to evaluate the success of the analysis. The processing of a disjunctive node will not reach this

For examples, see Darden.

stage unless all the dlsjuncts reported some type of error. In this case the parser attempts to select the most promising path for error recovery by replacing the disjunctive node by the subnode reporting the highest basic symbol match score (ties are broken by taking the subnode with the largest number of selecting characters). If all subnodes reported total failure, then total failure is indicated for the node.

The scoring of conjunctive nodes is straightforward. The basic symbol match score has already been recorded by the association process. The complete node score is simply the logical product of the scores of all the subnodes. Iterative and selective nodes are simply given the score of their respective subnodes.

Summary of Context Analysis

The context analysis system comprises three phases. The first phase (extended-dictionary matching) replaces substrings of the text which are sufficiently close to basic symbols in the dictionary. The second phase (Name-space clustering) attempts to discriminate an instance of every name in the text and to assign a common spelling to all occurrences of the same name. The third phase (Syntactic Analysis) applies a strategy of unbounded-context parsing and hierarchical consistency checking to achieve reliable partitioning of the entire text into simpler subproblems. The main path through the parser accomplishes the matching of patterns to the text by successive covering, association, and resolution processes. Identifiers are given special treatment. Error detection and recovery is based on the discovery of either partial matches (e.g., head or tail matches) or matches which are properly contained within the bounds assigned to a subgoal.

V. Discussion

This paper has been concerned with the methods employed in a syntax-directed contextual post-processor for the detection and correction of character recognition errors. We conclude with a few observations about these methods and about possible directions for future work.

The high reliability of the extended-dictionary matching process follows from the fact that dictionary entries of the same length are generally quite distant from one another (in the sense of a Hamming-distance) and from arbitrary strings which may occur in the object language.

If one is satisfied with the assignment of a common spelling to all occurrences of the same name, the name-space clustering performs satisfactorily. Even if this spelling is different from that intended by the author of the text, if the name is local to the processed text, then the resulting program is semanticaliy equivalent to the original. However, the author of the text may be dismayed to find that every occurrence of his variable MIN has been changed to MOM.

It is clear that if the present strategy and tactics prove inadequate to achieve our goal of a 1% overall error rate, that there are a large number of extensions which can be expected to further enhance its performance. These generally involve improvements in the parser's handling of local context, for we are fairly confident that the parser has a firm grip on global syntactic structure.

The principal techniques currently employed in hierarchical consistency checking involve the parenthesis-counting of minimal cover nests and the basic-symbol matching of the association process. Many additional checks are possible: for example, constraints on the minimum and maximum (where maxima exit), lengths of substrings, checks for valid prefixes and suffixes and for substrings which cannot appear in the tentatively assigned subgoal. 4,[6]

There are two extensions which we are particularly interested in investigating: (a) The detection of prefixes or suffixes in identifiers. The intrinsic nature of clustering methods is to group very similar samples. Thus is it likely that a method which associates all instances of BUFFER will also group together all instances of both BUFFERI and BUFFERI. However, the distance measure should be able to recognize the cases where two strings have tail (head) substrings which are quite distant relative to the root substring and give proportionately more weight to the tail (head) contribution to the distance. (b) The concept of a name needs to be extended to Include names (such as arrays or functions) which nearly always occur with some accompanying structures (e.g., MATRIX [...,...] or FUN (. . . . . ,...))• The recognition of this structure will help both in discovering instances of the name and in correcting errors in the accompanying bracket structure.

A final observation is that the unbounded-context parser's grasp of global structure may well prove sufficient for the system to discover the language membership of a piece of garbled text. That is, if the system contains RODEN-form grammars for ALGOL, FORTRAN, LISP, and SNOBOL, the parser could determine which grammar to apply, based on an initial analysis of the text by each grammar, taking the most redundant grammars first.

Acknowledgements

The author wishes to thank his colleagues at The University of Texas at Austin, and especially Dr. Woodrow W. Bledsoe and Dr. E. M. Greenawalt.

References

1. Darden, S. C, "A Contextual Recognition System for Hand-printed Formal Languages," TSN-1, Computation Center, The University of Texas at Austin (January 1969).

2. Duda, R. 0., P. E. Hart, and J. H. Munson, "Graphical-Data-Processing Research Study and

Experimental Investigation," Fourth Quarterly Report, Contract DA 28-043 AMC-01901(E), SRI Project ESU 5864, Stanford Research Institute, Menlo Park, California (March 1967).

3. Duda, R. 0., and J. H. Munson, "Graphical-Data-Processing Research Study and Experimental Investigation," Fifth Quarterly Report, Contract DA 28-043 AMC-01901(E), SRI Project ESU 5864, Stanford Research Institute, Menlo Park, California (June 1967).

4. Duda, R. 0., and P. E. Hart, "Experiments in the Recognition of Hand-printed Text: Part II - Context Analysis," AFIPS Conf. Proc, Fall Joint Computer Conference (1968).

5. Irons, E. T., "An Error-correcting Parse Algorithm," Comm. ACM 6, 11, pp. 669-673 (November 1963).

6. Unger, S. H., "A Global Parser for Context-free Phrase Structure Languages," Comm. ACM 11, 4, pp. 240-247 (April 1968).

7. Floyd, R. W., "The Syntax of Programming Languages - A Survey," IEEE Trans. EC-13, 4 (August 1964).

8. Vossler, C. M., and N. M. Branston, "The Use of Context for Correcting Garbled English Text," Proc. ACM 19th National Conference, paper D2.4-1, D2.4-13 (1964).

9. Casey, R. G., and G. Nagy, "An Autonomous Reading Machine," IEEE Trans. EC-17, 5, pp. 492-503 (May 1968).