20

1965 International Conference on
Computational Linguistics

ENDOCENTRIC CONSTRUCTIONS AND THE COCKE PARSING LOGIC

Jane J. Robinson

The RAND Corporation
1700 Main Street
Santa Monica, California 90406

1

ABSTRACT

Automatic syntactic analysis is simplified by disengaging the grammatical rules, by means of a parsing logic, from the computer routines that apply them. A case in point is the John Cocke logic. It iterates on five simple parameters and finds all structures permitted by the grammar, thus testing the rules, which can then be changed without changing the routines. The rules themselves need not be ordered so far as the logic of the system is concerned. However, in operating with an IC grammar, rules for bracketing endocentric constructions must be made quite complex merely to avoid multiple analyses of unambiguous or trivially ambiguous expressions. The rules can be simplified if they are classified and if the system is provided with an additional capability for applying them in a specified order. Although an additional parameter is introduced into the system, the disengagement of grammar from routine is preserved. The additional parameter controls the direction, left-to-right or right-to-left, in which constructions are put together. The decision as to which direction should be specified is a grammatical decision, and is related to Yngve's hypothesis of asymmetry in language. It does not affect the operation of the parsing logic.

ACKNOWLEDGMENTS

I wish to acknowledge the assistance of M. Kay and S. Marks in discussing points raised in the paper and in preparing the flowchart. A more general acknowledgment is due to D. G. Hays, who first called my attention to the problem of ordering the attachment of elements.

ENDOCENTRIC CONSTRUCTIONS AND THE COCKE PARSING LOGIC

Automatic sentence structure determination (SSD) is greatly simplified if, through the intervention of a parsing logic, the grammatical rules that determine the structure are partially disengaged from the computer routines that apply to them. Some earlier parsing programs analyzed sentences by routines that branched according to the grammatical properties or signals encountered at particular points in the sentence, making the routines themselves serve as the rules. This not only required separate programs for each language, but led to extreme proliferation in the routines, requiring extensive rewriting and debugging with every discovery and incorporation of a new grammatical feature. More recently, programs for SSD have employed generalized parsing logics, applicable to different languages and providing primarily for an exhaustive and systematic application of a set of rules.(1,2,3,5) The rules themselves can be changed without changing the routines that apply them, and the routines consequently take fuller advantage of the speed with which digital computers can repeat the same sequence of instructions over and over again, changing only the values of some parameters at each cycle.

The case in point is the parsing logic (PL) devised by John Cocke in 1960, for applying the rules of a context-free phrase structure grammar (PSG), requiring that each structure recognized by the grammar be analyzed into two and only two immediate constituents.(1)

Although all PSGs appear to be inadequate in some important respects to the task of handling natural language, they still form the base of the more powerful transformational grammars, which are not yet automated for SSD. Moreover, even their severest critic acknowledges that "The PSG conception of grammar...is a quite reasonable theory of natural language which unquestionably formalizes many actual properties of human language."(6,p.78) Both theoretically and empirically the development and automatic application of PSGs are of interest to linguists.

The PSG on which the Cocke PL operates is essentially a table of constructions. Its rules have three entries, one for the code (a descriptor) of the construction, the other two specifying the codes of the ordered pair of immediate constituents out of which it may be formed. The logic iterates in five nested loops, controlled by three simple parameters and two codes supplied by the grammar. They are: 1) the string length, starting with length 2, of the segment being tested for constructional status; 2) the position of the first word in the tested string; 3) the length of the first constituent; 4) the codes of the first constituent; and 5) the codes of the second constituent.

After a dictionary lookup routine has assigned grammar codes to all the occurrences in the sentence or total string to be parsed (it need not be a sentence), the PL operates to offer the codes of pairs of adjacent segments to a parsing routine that tests their connectability by looking them up in the stored table of constructions, i.e., in the grammar. If the ordered pair is matched by a pair of ICs in the table, the code of the construction formed by the ICs is added to the list of codes to be offered for testing when iterations are performed on longer strings.*

In the RAND program for parsing English, the routines produce a labeled binary-branching tree for every complete structural analysis. There will be one tree if the grammar recognizes the string as well-formed and syntactically unambiguous; more than one if it is recognized as ambiguous. Even if no complete analysis is made of the whole string, a resumé lists all constructions found in the process, including those which failed of inclusion in larger constructions.(8,9)

*This interaction between a PL and a routine for testing the connectability of two items is described in somewhat greater detail in Hays (2).

Besides simplifying the problem of revising the grammar by separating it from the problem of application to sentences, the PL, because it leads to an exhaustive application of the rules, permits a rigorous evaluation of the grammar's ability to assign structures to sentences and also reveals many unsuspected yet legitimate ambiguities in those sentences.(4,7) But because of the difficulties inherent in specifying a sufficiently discriminatory set of rules for sentences of any natural language and because of the very many syntactic ambiguities, resolvable only through larger context, this method of parsing produces a long list of intermediate constructions for sentences of even modest length, and this in turn raises a storage problem.

By way of illustration, consider a string of four occurrences, x_1 x_2 x_3 x_4, a dictionary that assigns a single grammar code to each, and a grammar that assigns a unique construction code to every different combination of adjacent segments. Given such a grammar, as in Table I, the steps in its application to the string by the parsing routines operating with the Cocke PL are represented in Table II. (The preliminary dictionary lookup assigning the original codes to the occurrences is treated as equivalent to iterating with the parameter for string length set to 1).

Table I

Rule #	IC1	IC2	CC	Rule #	IC1	IC2	CC
1.	A	B	E	8.	A	J	L
2.	B	C	F	9.	A	K	M
3.	C	D	G	10.	E	G	N
4.	A	F	H	11.	H	D	Ø
5.	E	C	I	12.	I	D	P
6.	B	G	J	13.	A	C	Q
7.	F	D	K	14.		etc.	

IC1: code of first constituent
IC2: code of second constituent

CC: code of construction

Table II

#	M	W	P	C(P)	C(Q)	C(M)	Rule #	Steps Combined	Structure Assigned
1.	1	1	1	A		A		Dictionary	x_1
2.	1	2	1	B		B		lookup	x_2
3.	1	3	1	C		C		assigning	x_3
4.	1	4	1	D		D		codes to:	x_4
5.	2	1	1	A	B	E	1.	1+2	(x_1+x_2)
6.	2	2	1	B	C	F	2.	2+3	(x_2+x_3)
7.	2	3	1	C	D	G	3.	3+4	(x_3+x_4)
8.	3	1	1	A	F	H	4.	1+6	$(x_1(x_2+x_3))$
9.	3	1	2	E	C	I	5.	5+3	$((x_1+x_2)x_3)$
10.	3	2	1	B	G	J	6.	2+7	$(x_2(x_3+x_4))$
11.	3	2	2	F	D	K	7.	6+4	$((x_2+x_3)x_4)$
12.	4	1	1	A	J	L	8.	1+10	$(x_1(x_2(x_3+x_4)))$
13.	4	1	1	A	K	M	9.	1+11	$(x_1((x_2+x_3)x_4))$
14.	4	1	2	E	G	N	10.	5+7	$((x_1+x_2)(x_3+x_4))$
15.	4	1	3	H	D	Ø	11.	8+4	$((x_1(x_2+x_3))x_4)$
16.	4	1	3	I	D	P	12.	9+4	$(((x_1+x_2)x_3)x_4)$

#: step number
M: string length of segment
P: length of first construction string
C(P): code of first construction

C(Q): code of second const. string
C(M): code for string, to be stored when C(P) and C(Q) are matched in the grammar.
C(M) = CC of grammar.

The boxed section represents the PL iterations.

With such a grammar, the number of constructions to be stored and processed through each cycle increases in proportion to the cube of the number of words in the sentence. If the dictionary and grammar assign more than one code to occurrences and constructions, the number may grow multiplicatively, making the storage problem still more acute. For example, if x_1 were assigned two codes instead of one, additional steps would be required for every string in which x_1 was an element and iteration on string length 4 would require twice as many cycles and twice as much storage.

Of course, reasonable grammars do not provide for combining every possible pair of adjacent segments into a construction, and in actual practice the growth of the construction list is reduced by failure to find the two codes presented by the PL, when the grammar is consulted. If Rule 1 is omitted from the grammar in Table I, then steps 5, 9, 14, and 16 will disappear from Table II and both storage requirements and processing time will be cut down. Increasing the discriminatory power of the grammar through refining the codes so that the first occurrence must belong to class Aa and the second to class Bb in order to form a construction provides this limiting effect in essentially the same way.

Another way of limiting the growth of the stored constructions is to take advantage of the fact that in actual grammars two or more different pairs of constituents sometimes combine to produce the "same" construction. Assume that A and F (Table I) combine to form a construction whose syntactic properties are the same, at least within the discriminatory powers of the grammar, as those of the construction formed by E and C. Then Rules 4 and 5 can assign the same code, H, to their constructions. In consequence, at both steps 8 and 9 in the parsing (Table II), H will be stored as the construction code C(M) for the string x_1 x_2 x_3, even though two substructures are recorded for it: i.e. $(x_1(x_2 + x_3))$ and $((x_1 + x_2)x_3)$. The string can be marked as having more than one structure, but in subsequent iterations on string length 4, only one concatenation of the string with x_4 need be made and step 16 can be omitted. When the parsing has terminated, all substructures of completed analyses are recoverable, including those of marked strings.

Eliminating duplicate codes for the same string from the cycles of the PL results in dramatic savings in time and storage, partly because the elimination of any step has a cumulative effect, as demonstrated previously. In addition, opportunities to eliminate duplicates arise frequently, in English at least, because of the frequent occurrence of endocentric constructions, constructions whose syntactic properties are largely the same as those of one of their elements--the head. In English, noun phrases are typically endocentric, and when a noun head is flanked by attributives as in a phrase consisting of article, noun, prepositional phrase (A N PP), the requirement that constructions have only two ICs promotes the assignment of two structures, (A(N+PP)) and ((A+N)PP), unless the grammar has been carefully formulated to avoid it. Since NPs of this type are ubiquitous, occurring as subjects, objects of verbs, and objects of prepositions, duplicate codes for them are likely to occur at several points in a sentence.

Consideration of endocentric constructions, however, raises other questions, some theoretical and some practical, suggesting modification of the grammar and the parsing routines in order to represent the language more accurately or in order to save storage, or both. Theoretically, the problem is the overstructuring of noun phrases by the insistence on two ICs and the doubtful propriety of permitting more than one way of structuring them. Practically, the problem is the elimination of duplicate construction codes stored for endocentric phrases when the codes are repeated for different string lengths.

Consider the noun phrase subject in All the old men on the corner stared. Its syntactic properties are essentially the same as that of men. But fifteen other phrases, all made up from the same elements but varying in length, also have the same properties. They are shown below:

Table III

	Length	Noun phrase							
1.	7	All	the	old	men	on	the	corner	(stared)
2.	6		The	old	men	on	the	corner	
3.	6	All	the		men	on	the	corner	
4.	6	All		old	men	on	the	corner	
5.	5			Old	men	on	the	corner	
6.	5		The		men	on	the	corner	
7.	5	All			men	on	the	corner	
8.	4				Men	on	the	corner	
9.	4	All	the	old	men				
10.	3		The	old	men				
11.	3	All	the		men				
12.	3	All		old	men				
13.	2			Old	men				
14.	2		The		men				
15.	2	All			men				
16.	1				Men				

A reasonably good grammar should provide for the recognition of all sixteen phrases. This is not to say that sixteen separate rules are required, although this would be one way of doing it. Minimally, the grammar must provide two rules for an endocentric NP, one to combine the head noun or the string containing it with a preceding attributive and another to combine it with a following attributive. The codes for all the resulting constructions may be the same, but even so, the longest phrase will receive four different structural assignments or bracketings as its adjacent elements are gathered together in pairs; namely:

```
          (all (the (old (men (on the corner)))))
          (all (the ((old men) (on the corner))))
          (all ((the (old men)) (on the corner)))
and       ((all (the (old men))) (on the corner))
```

If it is assumed that the same code, say that of a plural NP, has been assigned at each string length, it is true that only one additional step is needed to concatenate the string with the following verb when the PL iteration is performed for string length 8. But meanwhile a number of intermediate codes have been stored during iterations on string lengths 5, 6, and 7 as the position of the first word of the tested string was advanced, so that the list also contains codes for:

```
                  men on the corner stared   (length 5)
              old men on the corner stared   (length 6)
and       the old men on the corner stared   (length 7)
```

Again, the codes may be the same, but duplicate codes will not be eliminated from processing if they are associated with different strings, and strings of different length are treated as wholly different by the PL, regardless of overlap. If this kind of duplication is to be reduced or avoided, a different procedure is required from that available for the case of simple duplication over the same string.

But first a theoretical question must be decided. Is the noun phrase, as exemplified above, perhaps really four-ways ambiguous and do the four different bracketings correlate systematically with four distinct interpretations or assignments of semantic structure?(Cf. 4,7) And if so, is it desirable to eliminate them? It is possible to argue that some of the different bracketings do correspond to different meanings or emphases, or--in earlier transformational terms--to different orderings in the embeddings of *the men were old* and *the men were on the corner* into *all the men stared*. Admittedly the native speaker can indicate contrasts in meaning by his intonation, emphasizing in one reading that *all* the men stared and in another that it was all the *old* men who stared; and the writer can resort to italics. But it seems reasonable to assume that there is a normal intonation for the unmarked and unemphatic phrase and that its interpretation is structurally unambiguous. In the absence of italics and other indications, it seems *un*reasonable to produce four different bracketings at every encounter with an NP of the kind exemplified.

One way to reduce the duplication is to write the grammar codes so that, with the addition of each possible element, the noun head is assigned a different construction code whose distribution as a constituent in larger constructions is carefully limited. For the sake of simplicity, assume that the elements of NPs have codes that reflect, in part, their ordering within the phrase and that the NP codes themselves reflect the properties of the noun head in first position and are subsequently differentiated by codes in later positions that correspond to those of the attributes. Let the codes for the elements be 1 (all), 2 (the), 3 (old), 4 (men), 5 (on the corner). Rules may be written to restrict the combinations, as follows:

Table IV

R#	IC1 IC2	CC	
1.	1 + 4	→ 41	(all men)
2.	2 + 4	→ 42	(the men)
3.	3 + 4	→ 43	(old men)
4.	4 + 5	→ 45	(men on the corner)
5.	1 + 42	→ 412	(all the men)
6.	1 + 43	→ 413	(all old men)
7.	2 + 43	→ 423	(the old men)
8.	1 + 423	→ 4123	(all the old men)
9.	1 + 45	→ 415	(all men on the corner); but not *41 + 5 → 415
10.	2 + 45	→ 425	(the men on the corner); but not *42 + 5 → 425
11.	3 + 45	→ 435	(old men on the corner); but not *43 + 5 → 435
12.	2 + 435	→ 4235	(the old men on the corner); but not *423 + 5 → 4235
13.	1 + 4235	→ 41235	(all the old men on the corner); but not *4123 + 5 → 41235

With these rules, the grammar provides for only one structural assignment to the string: (all (the (old (men + on the corner)))).

This method has the advantage of acknowledging the general endocentricity of the NP while allowing for its limitations, so that where the subtler differences among NPs are not relevant, they can be ignored by ignoring certain positions of the codes, and where they are relevant, the full codes are available. The method should lend itself quite well to code matching routines for connectability. However, if carried out fully and consistently, it greatly increases the length and complexity of both the codes and the rules, and this may also be a source of problems in storage and processing time. (cf. Hays, 2)

Another method is to make use of a classification of the rules themselves. Since the lowest loop of the PL (see Fig. 1) iterates on the codes of the second constituents, the rules against which the paired strings are tested are stored as ordered by first IC codes and subordered by second IC codes. If the iterations of the logic were differently ordered, the rules would also be differently ordered, for efficiency in testing. In other words, the code of one constituent in the test locates a block of rules within which matches for all the codes of the other constituent are to be sought; but the hierarchy of ordering by one constituent or the other is a matter of choice so long as it is the same for the PL and for storing the table of rules that constitute the grammar. In writing and revising the rules, however, it proves humanly easier if they are grouped according to construction types. Accordingly, all endocentric NPs in the RAND grammar are given rule identification tags with an A in first position. Within this grouping, it is natural to subclass the rules according to whether they attach attributives on the right or on the left of the noun head. If properly formalized, this practice can lead to a reduction in the multiple analyses of NPs with fewer rules and simpler codes than those of the previous method.

As applied to the example, the thirteen rules and five-place codes of Table IV can be reduced to two rules with one-place codes and an additional feature in the rule identification tag. The rules can be written as:

```
*A1   1   N   N
      2
      3

$A2   N   4   N
```

Although the construction codes are less finely differentiated, the analysis of the example will still be unique, and the number of abortive intermediate constructions will be reduced. To achieve this effect, the connectability test routine must include a comparison of the rule tag associated with each C(P) and the rule tags of the grammar. If a rule of type *A is associated with the C(P), that is, if an *A rule assigned the construction code to the string P which is now being tested as a possible first constituent, then no rule of type $A can be used in the current test. For all such rules, there will be an automatic "no match" without checking the second constituent codes. (See Fig. 1.) As a consequence of this restriction, in the final analysis, the noun head will have been combined with all attributives on the right before acquiring any on the left.

To be sure, the resumé of intermediate constructions will contain codes for _old men_, _the old men_, and _all the old men_, produced in the course of iterations on string lengths 2, 3, and 4, but only one structure is finally assigned to the whole phrase and the intermediate duplications of codes for strings of increasing length will be fewer because of the hiatus at string length 5. Of course, in the larger constructions in which the NP participates, the reduction in the number of stored intermediate constructions will be even greater.

Provisions may be made in the rules for attaching still other attributives to the head of the NP without great increase in complexity of rules or multiplication of structural analyses. Rule $A2, for example, could include provision for attaching a relative clause as well as a prepositional phrase, and while a phrase like _the men on the corner who were sad_ might receive two analyses unless the codes were sufficiently differentiated to prevent the clause from being attached to _corner_ as well as to _men_, at least the further differentiation of the codes need not also be multiplied in order to prevent the multiple analyses arising from endocentricity.

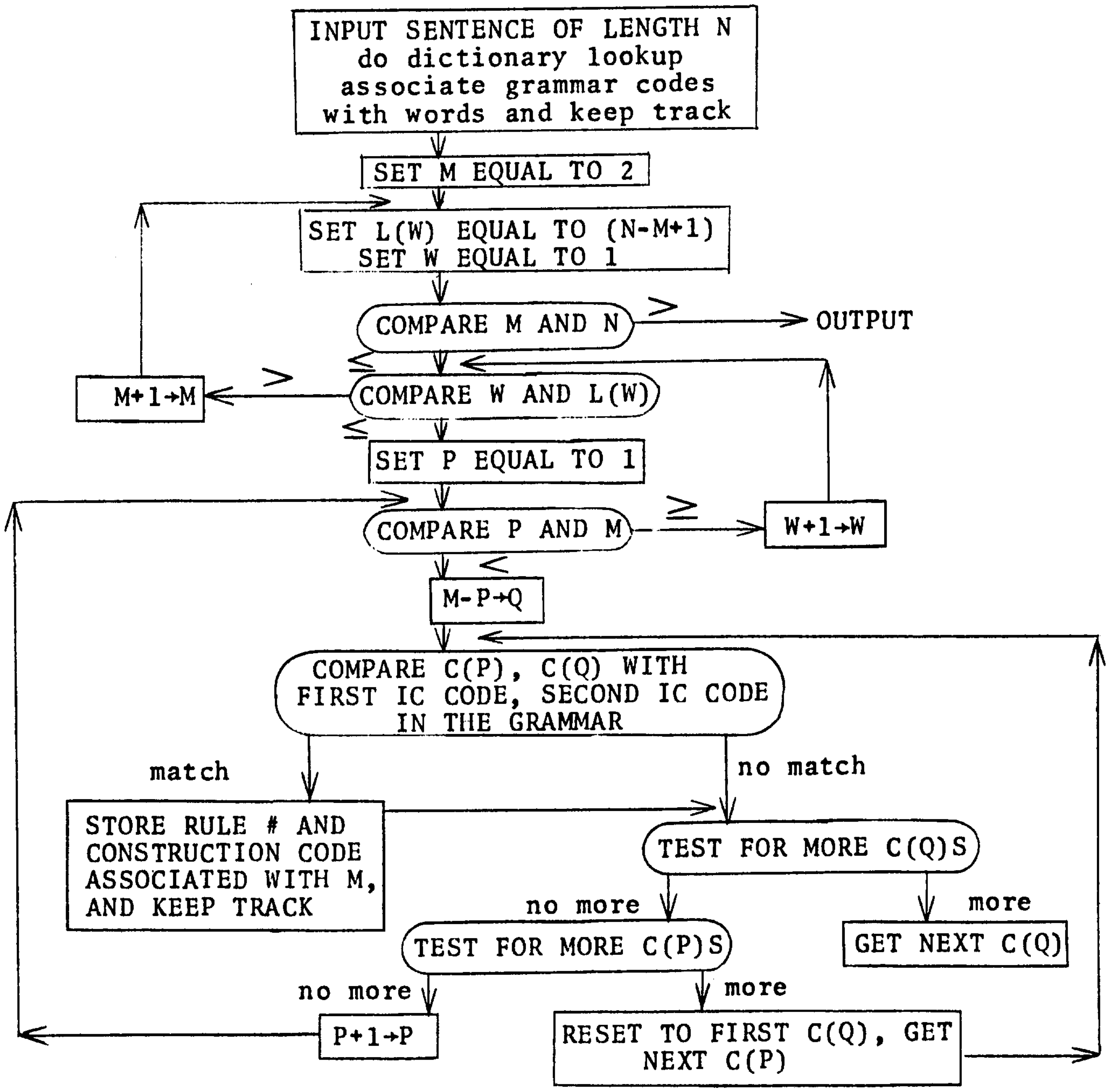
Similarly, for verb phrases where the rule must allow for an indefinite number of adverbial modifiers, a single analysis can be obtained by marking the strings and the rules and forcing a combination in a single direction. In short, although the Cocke PL tends to promote multiple analysis of unambiguous or trivially ambiguous endocentric phrases, at the same time increasing the problem of storing intermediate constructions, the number of analyses can be greatly reduced and the storage problem greatly alleviated if the rules of the grammar recognize endocentricity wherever possible and if they are classified so that rules for endocentric constructions are marked as left (*) or right ($), and their order of application is specified.

A final theoretical-practical consideration can at least be touched on, although it is not possible to develop it adequately here. The foregoing description provided for combining a head with its attributives (or dependents) on the right before combining it with those on the left, but either course is possible. Which is preferable depends on the type of construction and on the language generally. If Yngve's hypothesis that languages are essentially asymmetrical, tending toward right-branching constructions to avoid overloading the memory, is correct, then the requirement to combine first on the right is preferable.(10) This is a purely grammatical consideration, however, and does not affect the procedure sketched above, in principle. For example, consider an endocentric construction of string length 6 with the head at position 3, so that its extension is predominantly to the right, thus: 1 2 (3) 4 5 6. If all combinations were allowed by the rules, there would be thirty-four analyses. If combination is restricted to _either_ direction, left or right, the number of analyses is reduced to eleven. However, if the Cocke PL is used to analyze a left-branching language, making it preferable to specify prior combination on the left, then the order of nesting of the fourth and fifth loops of the PL should be reversed (Fig. 1) and the rules of the grammar should be stored in order of their second constituent codes, subordered on those of the first constituents.

Fig. 1

FLOWCHART FOR THE COCKE PL

N:	sentence length	W:	number of first word of M
P:	string length of first constituent	L(W):	N-M+1 = limit of first word
Q:	string length of second constituent	C(P):	code of first constituent
M:	P+Q = string length of construction	C(Q):	code of second constituent


INPUT SENTENCE OF LENGTH N do dictionary lookup associate grammar codes with words and keep track
SET M EQUAL TO 2
SET L(W) EQUAL TO (N-M+1) SET W EQUAL TO 1
COMPARE M AND N
>
OUTPUT
≤
M+1→M
>
COMPARE W AND L(W)
≤
SET P EQUAL TO 1
COMPARE P AND M
≥
W+1→W
<
M-P→Q
COMPARE C(P), C(Q) WITH FIRST IC CODE, SECOND IC CODE IN THE GRAMMAR
match
no match
STORE RULE # AND CONSTRUCTION CODE ASSOCIATED WITH M, AND KEEP TRACK
TEST FOR MORE C(Q)S
no more
more
TEST FOR MORE C(P)S
GET NEXT C(Q)
no more
more
P+1→P
RESET TO FIRST C(Q), GET NEXT C(P)

REFERENCES

1. Hays, D. G., "Automatic Language-Data Processing," Computer Applications in the Behavioral Sciences, Chapter 17, Prentice-Hall, 1962.

2. Hays, D. G. "Connectability Calculations, Syntactic Functions, and Russian Syntax," Mechanical Translation, Vol. 8, No. 1 (August 1964).

3. Kuno, S., and A. G. Oettinger, "Multiple-path Syntactic Analyzer," Mathematical Linguistics and Automatic Translation, Report No. NSF-8, Sec. I, The Computation Laboratory of Harvard University, 1963.

4. Kuno, S., and A. G. Oettinger, "Syntactic Structure and Ambiguity of English," AFIPS Conference Proceedings Vol. 24, 1963 Fall Joint Computer Conference.

5. National Physical Laboratory, 1961 International Conference on Machine Translation of Languages and Applied Language Analysis, Vol. 2, H. M. Stationery Office, 1962.

6. Postal, P. M. Constituent Structure, Publication Thirty of the Indiana University Research Center in Anthropology, Folklore, and Linguistics, January 1964.

7. Robinson, J., "Automated Grammars as Linguistics Tools," (Unpublished), Presented at the Thirty-ninth Annual Meeting of the Linguistic Society of America, New York, December 1964.

8. Robinson, J., The Automatic Recognition of Phrase Structure and Paraphrase, RM-4005-PR (Abridged), The RAND Corporation, Santa Monica, December 1964.

9. Robinson, J., Preliminary Codes and Rules for the Automatic Parsing of English, RM-3339-PR, The RAND Corporation, Santa Monica, December 1962.

10. Yngve, V. H., "A Model and an Hypothesis for Language Structure," Proceedings of the American Philosophical Society, Vol. 104, No. 5 (October 1960).