IV

THE CAMBRIDGE LANGUAGE RESEARCH UNIT COMPUTER PROGRAM FOR SYNTACTIC ANALYSIS

A. F. PARKER-RHODES
R. McKINNON WOOD
M. KAY
P. BRATLEY

See

The Cambridge Language Research Unit
Computer Program for Syntactic Analysis
A.F. Parker-Rhodes et al
Cambridge Language Research Unit, M.L. 136

CAMBRIDGE LANGUAGE RESEARCH UNIT
20 MILLINGTON ROAD
CAMBRIDGE, ENGLAND

ABSTRACT

The program developed for the Cambridge Computer EDSAC II is outlined. The program is divided into an interlingual part, given by the model, applicable in principle to any language (apart from details of strategy dependent, for instance, on the amount of information taken as a single computer 'word'), and a unilingual part containing procedures for dealing with the particular problems of one language, (in this case English).

PART IV. THE CAMBRIDGE LANGUAGE RESEARCH UNIT COMPUTER PROGRAM FOR SYNTACTIC ANALYSIS

The program is at present being tested on EDSAC II, the Cambridge University Mathematical Laboratory computer, by permission of the Director, Dr. Wilkes. Results obtained so far have been satisfactory, and a report will be issued shortly on the tests so far carried out for English.

For practical convenience in design and testing the program is divided into five sections co-ordinated by a master program.

Sections 1 and 2 are unilingual routines designed to remove some characteristic ambiguities of English, such as the omission of the relative pronoun in phrases such as "the man the woman saw". They also sort out, for example, the verbal and participial uses of words like "believed". They cannot, however, resolve genuine ambiguities: thus, for instance, there will always be a doubt as to whether "the old man and woman" means "the old man and the woman" or "the old man and the old woman". As the second interpretation is more usual these cases are always treated in this way. (It should be noticed that this does not really remove the ambiguity, but merely provides a reasonable basis for bracketting). In Section 3 the information required for bracketting is worked out from the entries in the word and bracket group dictionaries, after the first stage this is carried out on the basis of the bracketting already obtained in Section 5. The section includes a device for picking up alternative interpretations of particular substituents. Section 4 deals with conjunctions. Section 5 carries out the bracketting on the basis of the information provided by

Sections 3 and 4, and after each stage in the bracketting gives the 'united reading' for those parts of the sentence which have been bracketted: that is, a record of the brackets made. The last united reading is the completely bracketted sentence. All the sections except for Section 4 have been tested.

The input and output routines are of no linguistic interest, so there is no need to discuss them here. It should merely be noted that at present the initial dictionary look-up is not carried out mechanically: the input to the program therefore consists of the alphabetic list numbers of the text words, with their corresponding dictionary entries, in text order. (The bracket group dictionary is contained in the machine).

In the detailed description of the program given below the master program and each of the five sections are treated separately. Each description has three parts: first an outline of the logical character of the section of the program concerned; secondly a flow chart illustrating the program; and thirdly, the appropriate sequence of EDSAC orders, with annotations. The flow charts act as a link between the discursive outline and the actual orders, and the connections between the three are further emphasised by a system of cross-references. These are numbers defining series of orders in the program itself, and may not appear consecutively elsewhere. For convenience they are printed in bold-face type throughout.

Note on the special features of EDSAC II which have affected the construction of the program.

a) EDSAC II has a word-length of 40 bits; the syntactic information for each English text word can be accommodated in this space as follows: 10 bits are needed to identify the word, each word being represented by an alphabetic list number; 24 bits record its occurrence in the 12 kinds of bracket group used in English; and the remaining 6 bits carry habitat and concord information.

One of the 10 bits used to identify a text word is the sign-digit of the computer word; this distinguishes ordinary text words from the composite items constructed in the course of the program. A sign-digit will usually be required in a bracketting program and is a feature of most computers. With the 9 remaining bits only 512 text words can be differentiated, and to accommodate a more realistic dictionary economies would have to be made elsewhere; but it is also true that more than 40 bits (that is, more than one computer word) would certainly be required for all the information needed in a full translation program. For test purposes, however, the advantages of dealing with only one word are so great that every attempt is being made to use abbreviated coding and retain this advantage as long as possible.

b) EDSAC II has a quick-access store which will hold 1024 40-bit words, and an auxiliary magnetic-tape store. At present, when allowance has been made in the former for the program and working space, only sentences of up to about 40 words can be processed. A program using double-length computer words, and therefore requiring more orders, would thus be very evidently less convenient for this machine.

c) EDSAC II is a single-address machine; machine orders occupy 20 bits each, 11 being assigned to their address numbers. Facilities are available for performing various operations on the address numbers alone, and as these are comparatively quick considerable time-saving is possible. At several points in the program this device is used to accommodate information about kinds of bracket group in a convenient form.

d) The computer has only one control unit, and input, output, and machine orders can only be carried out one at a time. This simplifies the programming, but impedes certain limitations on the logical design of the program, and there is no reason why these limitations should be carried over to other machines.

It may be of interest to note that the predominantly non-arithmetical operations characteristic of this kind of program can be carried out at speeds ranging from 20 to 50 microseconds. Input and output are on five channel tape read at 1000 characters per second and punched at 300 characters per second.

SECTION 0

This section contains the master program which controls the five sections of the program. In these sections it will be noticed that an order is often annotated "return to master program".

0.0. The parameters of the program run are printed out: these refer to the addresses of orders in the machine which are required for test purposes.

0.0.1 The first sentence is read.

0.1 This calls Sections 1 and 2, the unilingual routines, into operation. (OA)
When these have been carried out control is returned to the master program.

0.2 The initial conditions for Section 3 are set and it is (OB)
called into operation.

0.3.0 If a bracketting possibility for the words in the working (OC)
store is found in Section 3, Section 5 is entered and the (OD)
bracket is recorded in the united reading. Control is then (OE)
returned to the master program at 0.1 and the cycle is repeated.

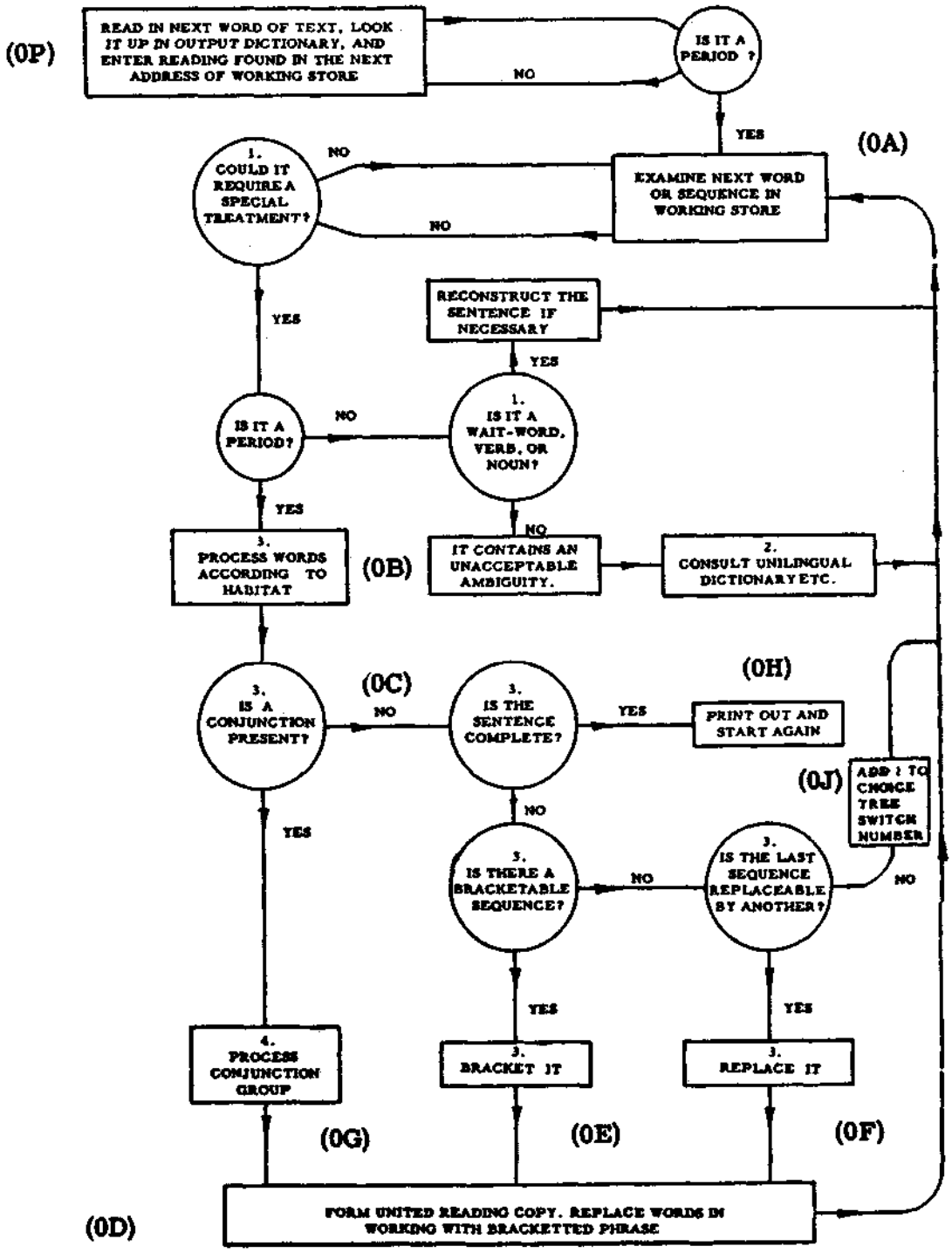
0.3.1 If no suitable bracketting possibility is found in Section 3 (OF)
the working store is enlarged to include more words for consideration under 0.3.0. If the working store contains the whole sentence, and this is represented by one bracket group, bracketting has been successful and 0.4.0 is entered. If the working store contains the whole sentence as more than one bracket group the procedure is restarted in 0.4.1 from the original text words with alternative interpretations of

bracket groups where possible.

0.3.2 If a conjunction is found in Section 3 Section 4 will be entered. (OG)

0.4.0 If bracketting has been successful the contents of the united reading (i.e. output) store are printed out according to order set on manual register of the control panel. The computer will then either stop or try the next sentence. It can also restart the sentence just bracketted, trying alternatives, as if the last attempt had not been successful. This is an extremely useful feature of the program. (OH)

0.4.1 If the bracketting has failed the choice tree level and choice tree switch number last reached (see description of Section 3) can be printed out if called for by the manual register. 1 is now added to the choice tree switch number in the appropriate place. If this makes the choice tree switch number so big that the computer indicates overflow, this shows that all possible ways of bracketting the sentence have been tried, and the computer will either stop or start the next sentence as ordered on the manual register If the computer does not indicate overflow, another attempt to bracket the sentence will be made using the new choice tree switch number. The computer will also print out, if required, the brackets formed before the failure. (OJ)



FLOW CHART OF FULL PROGRAM

SECTION 1

This is the first of two routines designed to remove certain kinds of ambiguity characteristic of the English language.

Operations are carried out in the following order:

1.1 The next substituent in the sentence store is examined. (1A)

1.2 If it is a 'wait' word the remainder of the sentence is scanned for the corresponding 'awaited' word. The latter is picked up and the sentence re-arranged so that the two words are together. "Either", for example, is a wait word, with "or" the corresponding awaited word. In a sentence such as "Either the man or the woman...", therefore, when "either" is encountered the following words are scanned for "or"; this when found is removed and inserted after "either" so that the sentence reads "The man either or the woman...". (IB)

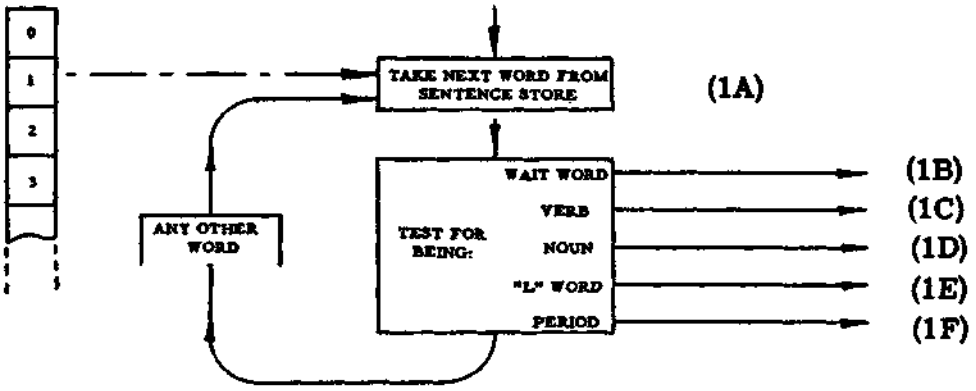
1.3 When a verbal substituent is encountered the immediate context is searched to discover whether a relative pronoun has been omitted, and if so, the omitted pronoun is replaced. Thus the sentence "The man the woman gave the apples to was.." would become "The man to whom the woman gave the apples was...". (IC)

1.4 Similarly, the immediate context of a nominal or pronominal substituent is searched so that a reversed subject and predicate can be picked up. The final punctuation mark is checked to see if the sentence is a question such as "Is he here?", for example, would read "He is here?", in sentences such as "Had he come I should have seen him" the word "if" is (ID)

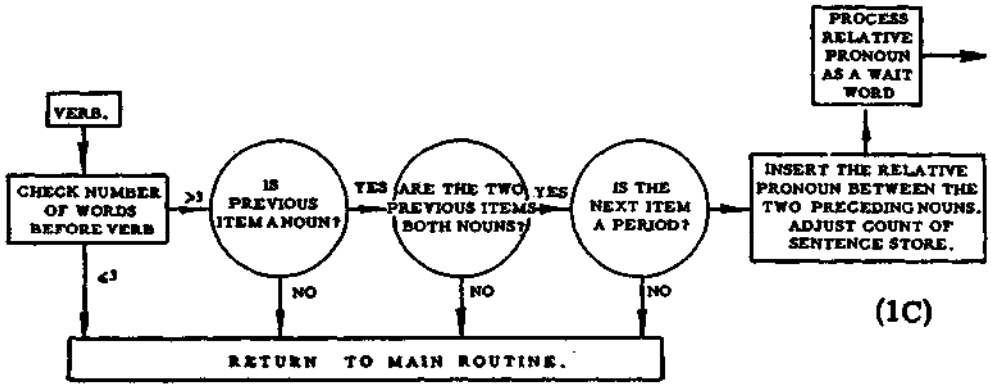
added to give "If he had come ..... ".

1.5 If a word with an L habitat is encountered control is transferred to Section 2. (1E)

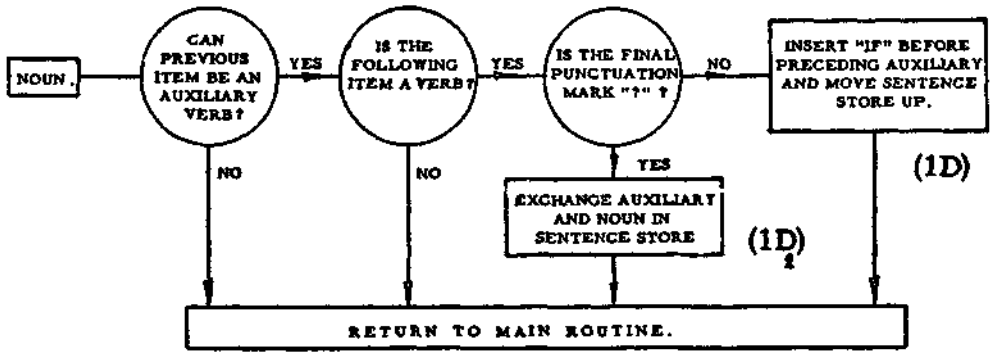
1.6 As sentences containing more than 30 words cannot at present be processed, when 30 words have been read without a full-stop occurring the machine signals failure and stops. If a full-stop is found it shows that the complete sentence has been scanned under Section 1 and Section 3 may then be entered or re-entered. (1F)



FLOW CHART OF PROGRAM SECTION 1.



FLOW CHART OF PROGRAM SECTION 1-2.



FLOW CHART OF PROGRAM SECTION 1-3.

SECTION 2

This is the second unilingual routine and is entered whenever a word with an L habitat is discovered by Section 1.

2.1 Examine the word in question. (2A)

2.2 Suppose this is the word "to". This can be classified under four heads: (2B)

i) as part of the infinitive form of a verb, as in "to err is human";

ii) as a verbal proposition as in "he is he man to do the job", where "to do the job" qualifies "man";

iii) as a preposition, as in "come to me";

iv) as a pro-infinitive, as in "I don't want to". When the relevant use has been selected the word is appropriately marked so that its use is known in later sections of the program.

2.3 If the word being examined is a past participle it is classified as participle or complete verb, as in "he was believed" and "he believed" and "he believed it was true" respectively. As in 2.2 the word is marked for future reference. (2C)

2.4 This section is very like 2.3 and is intended to remove ambiguities in the use of the present participle. A present participle is marked as belonging to one of the four categories participle/gerund, participle, gerund or infinitive. Examples are "It was running", "he is running", "running is difficult" and "running a business is difficult". (2D)

2.5 In this section the uses of the word "that" are examined in the same way as those of "to" under 2.2. "That" is classified under five heads: (2E)

i) as an article, as in "that boy is good";

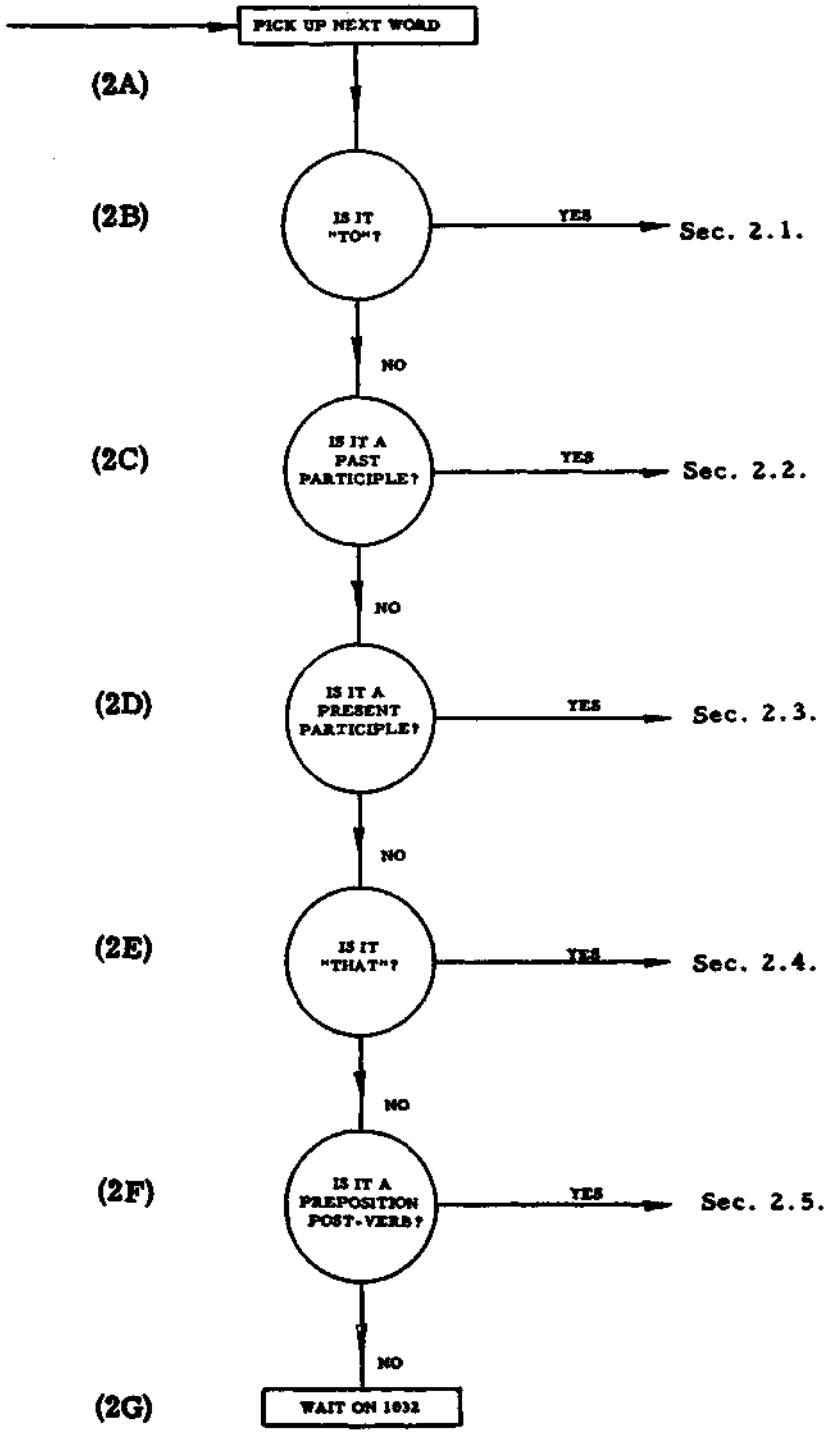
ii) as a relative pronoun, as in "the book that the dog is chewing";

iii) as article/pronoun, as in "he said that man is mortal";

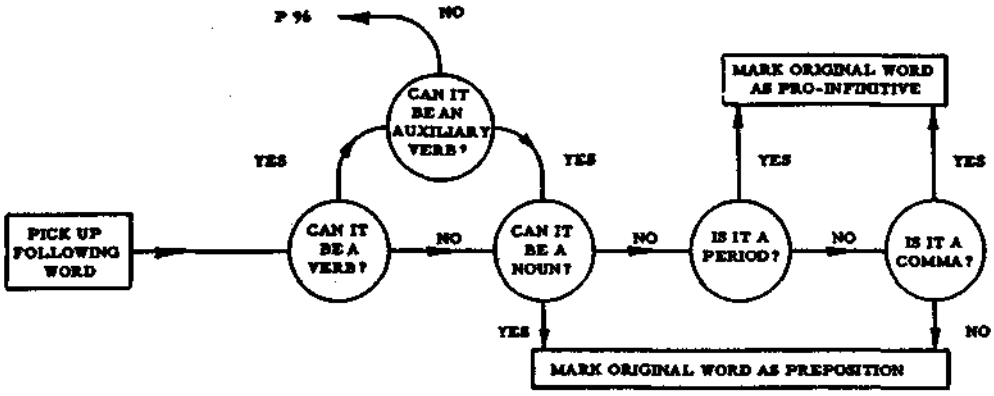
iv) as a noun clause marker, as in "he said that he would murder the brute". The fifth class is called merely THAT and deals with some special cases.

2.6 The final part of Section 2 resolves ambiguities arising from the use of a preposition immediately after a verb. For example, in "I am writing to my aunt" "to" occurs simply as a preposition and would be marked as such. On the other hand, in "I am writing up the report" "up" is not a simple preposition; that is, the sentence does not mean "I am starting at the bottom of the report and writing upwards". "Up" here really forms part of the verb and is classified as preposition post-verb. (2F)

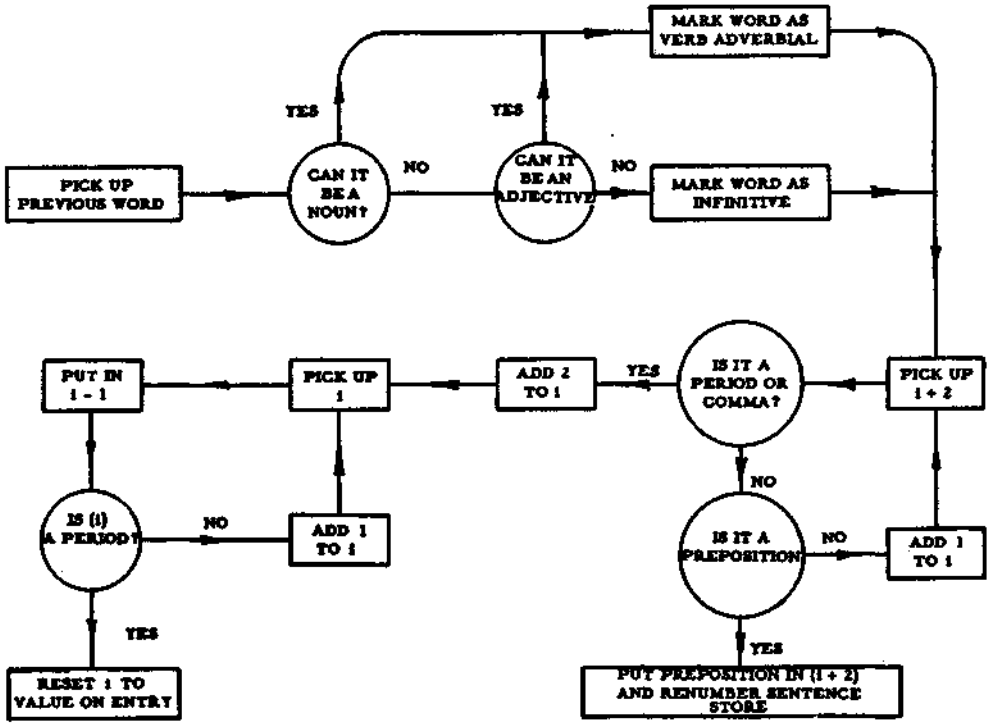
2.7 Should Section 2 have been entered for a reason not already dealt with, the computer will stop. (2G)



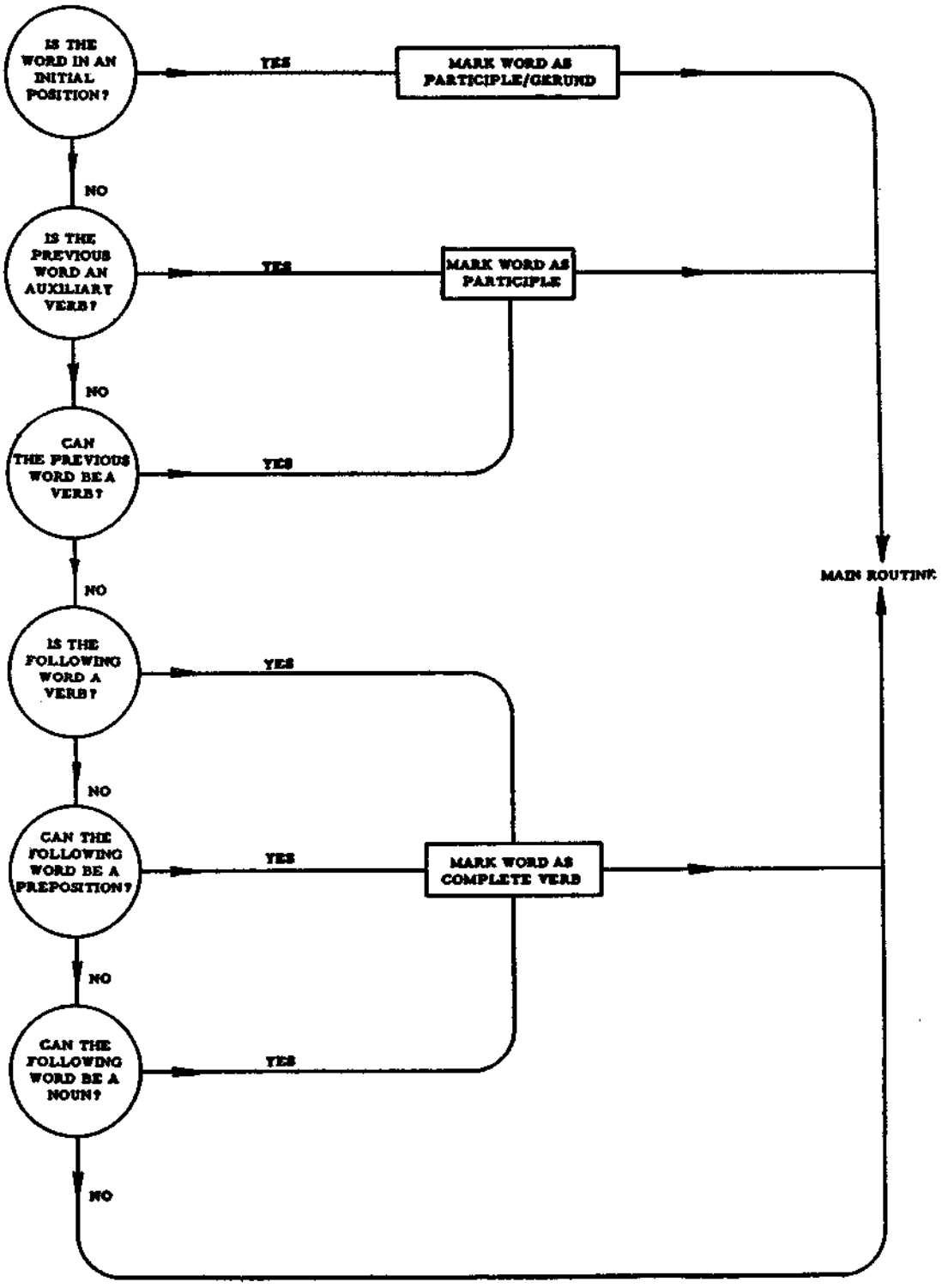
FLOW CHART OF PROGRAM SECTION 2



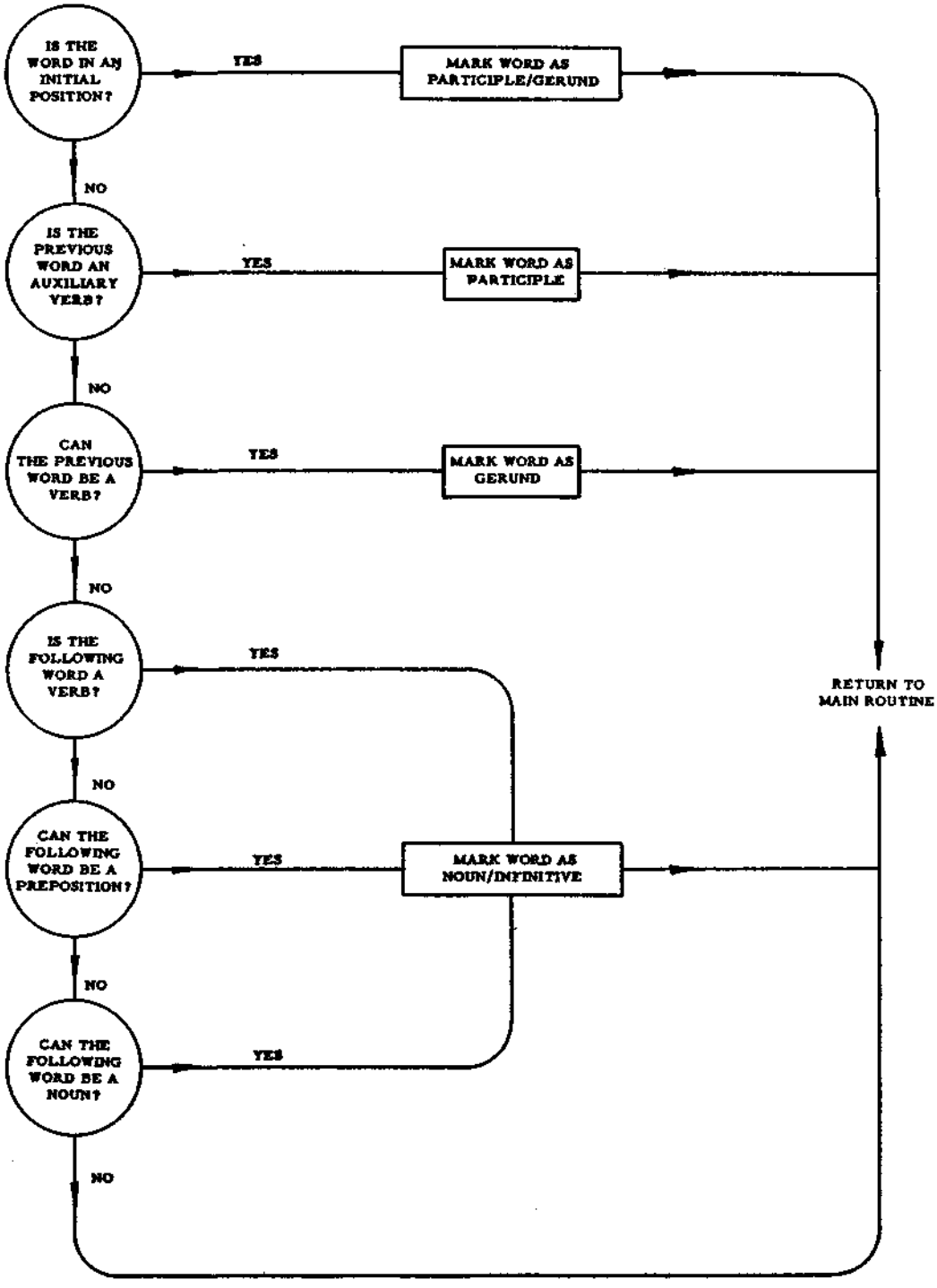
FLOW CHART OF PROGRAM SECTION 2.1



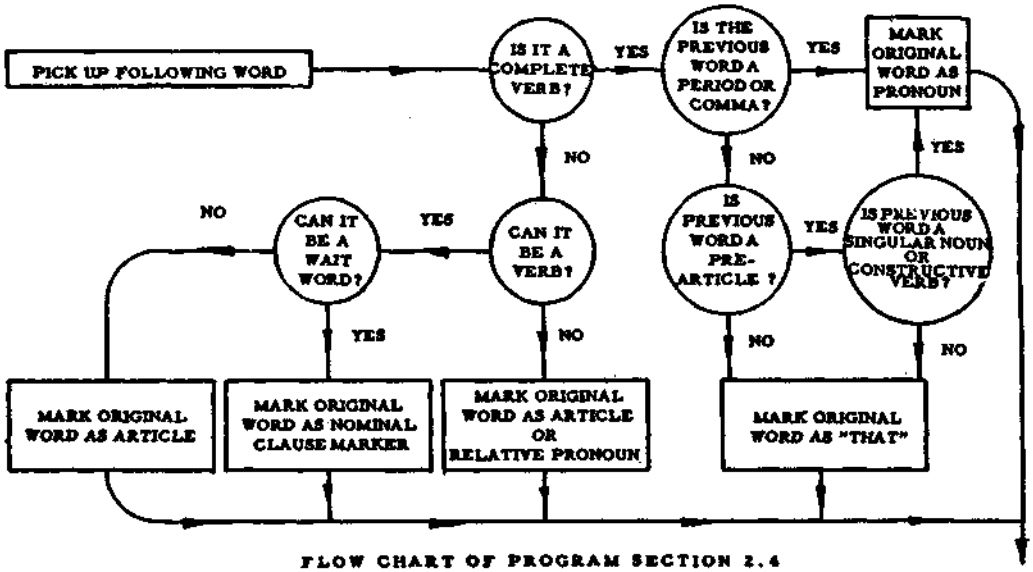
FLOW CHART OF PROGRAM SECTION 2.1
PART P %



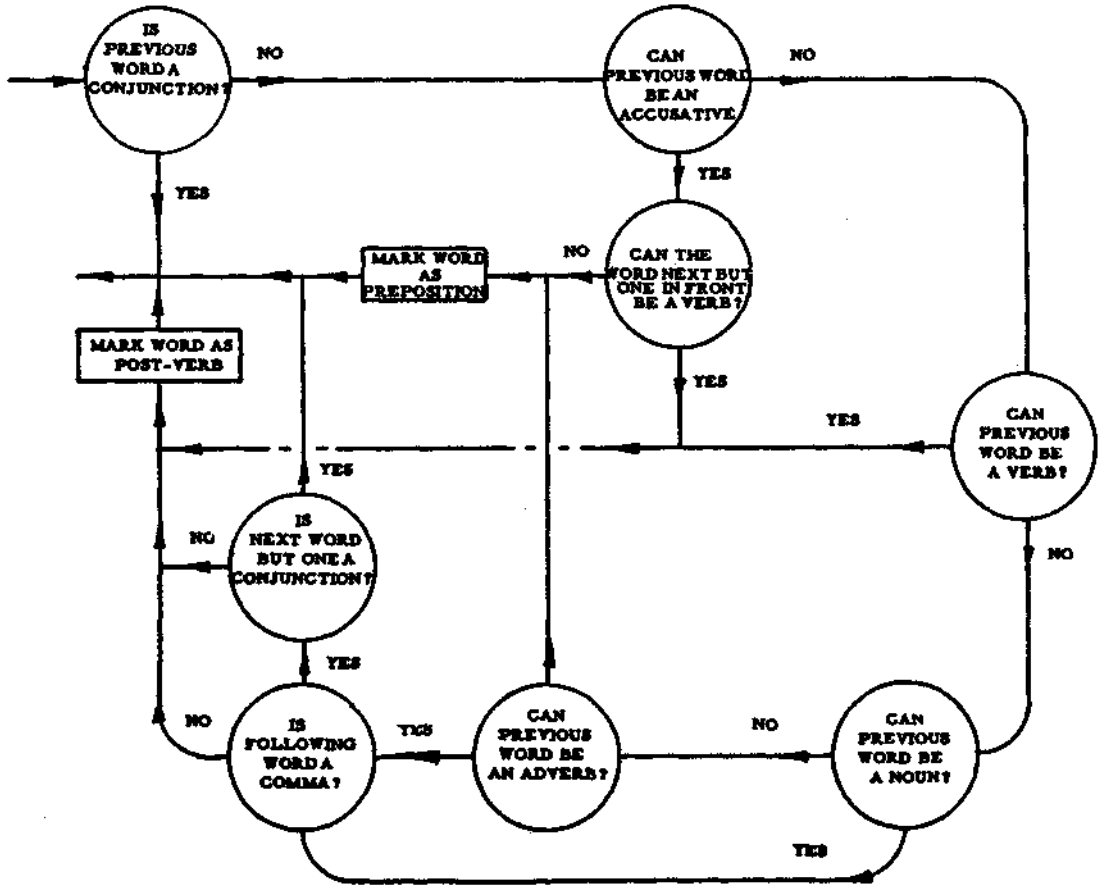
FLOW CHART OF PROGRAM SECTION 3.2



FLOW CHART OF PROGRAM SECTION 2.3



FLOW CHART OF PROGRAM SECTION 2.4



FLOW CHART OF PROGRAM SECTION 2.5

SECTION 3

This section chooses the brackets to be made. Before discussing the order in which the program actually performs the operations, we shall consider the way in which it performs them, and also the notion of a choice tree.

We shall first define a number called the "bracketting number" of a set of substituents. We know that a substituent may occur in one of four ways in any kind of bracket group; thus, it may not occur in that kind, it may occur only as a dependent, only as a governor, or as either. To every substituent, therefore, we attach a twelve digit number, in which each digit represents one kind of bracket group. The first digit represents the first kind, and so on. Each digit is either 0, 1, 2 or 3 according to the four ways in which it may occur as just described. Thus for example, a word with the number 320 10000 0000 attached could either occur as a dependent or a governor in kind 1, as a governor only in kind 2, and as a dependent only in kind 4; it could not occur in any other kind of bracket group.

Several substituents can be bracketted together in a particular kind of group only if it is possible to choose from them an arrangement with exactly one governor, all the other substituents being dependents in such a group. For example, substituents with associated numbers 100 00000 0000, 200 00000 0000, and 122 22222 222 could be bracketted together as a group of kind 1, but in no other way. Similarly, substituents with numbers 020 10000 3000, 010 20000 2022, and 010 20000 1000 could form a group of kind 2 or 9, but not of kind 4, where two of the

substituents are governors. We can thus associate with any set of substituents a number called the bracketting number of the set. This is another twelve digit number, and again each digit corresponds to one kind of bracket group. The digit in a position corresponding to a kind of group in which the set can be bracketted is a 1, and the other digits are 0.

Considering again the examples given above, we find for instance that with numbers

i) 100 00000 0000
ii) 200 00000 0000
iii) 122 22222 2222

the bracketting number is 100 00000 0000

Again, with numbers

i) 020 10000 3000
ii) 010 20000 2022
iii) 010 20000 1000

the bracketting number is 010 00000 1000

As a final example, we may suppose that we had items with numbers

i) 000 00312 1100
ii) 310 00322 1100
iii) 200 13231 2000

For these the bracketting number is 000 00110 1000.

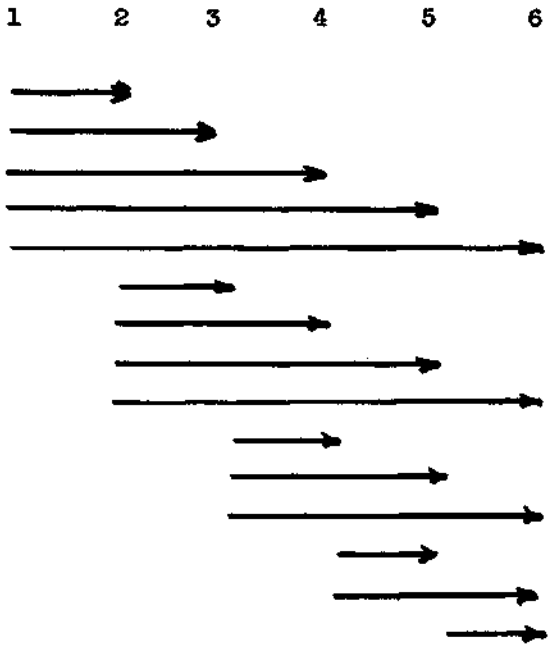
The digits representing kinds of bracket group are ordered from left to right according to the priority list mentioned in the introduction; when

we have a choice of kinds of group for a set of substituents, we follow the priority list by the simple device of taking the left-most 1 in the bracketting number.

The machine procedure is as follows:

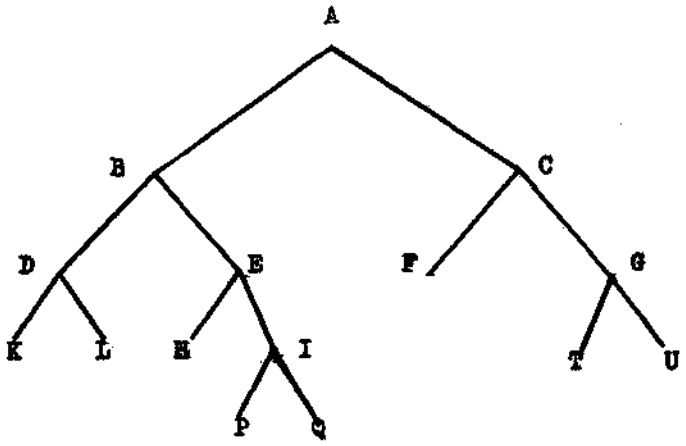
The set of substituents to be bracketted is read into the working store; all possible sets of consecutive substituents are examined in turn and the set with the highest bracketting number selected. If two sets sharing the highest bracketting number are found, the longer is chosen. If they are of the same length, the later one is selected. The set selected is bracketted (by Section V) and the bracket replaces its members in the sentence. The procedure is then repeated.

The way in which all possible consecutive sets in a sequence are tried is extremely simple. The diagram below shows the order in which the sets would be scanned in a sequence of six items:



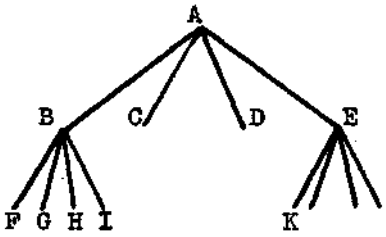
It will be seen that the order of scanning is to try first all consecutive sets beginning with the first word. Then all consecutive sets beginning with the second word and so on.

Sometimes the kind of bracket group selected in accordance with the priority list leads to an incorrect result. If this happens the program restarts with instructions to try in turn lower priority brackettings. The device used in these instructions is a "choice tree". We represent the choice tree as follows:



In this diagram, each line represents an operation performed by the program, and each of the points A, B, ...T, U a situation in which a choice must be made. At each of these points the operation represented by the left-hand line is in some sense to be preferred. With each point of the tree is associated a switch which determines the line to be followed. Initially, all switches are set to the most preferred choice. If a route through the tree leads to failure, the last switch on the route is changed to the next choice in the preference order. If there is no next choice, the switch before that is similarly changed and so on. The tree is then re-entered from the beginning, and should failure again occur, the appropriate switches are changed in the same way as before. Thus, let us suppose that in the tree above the points K, L, H, P, Q, & F are failure points. The first route through the tree is ABDK, ending in failure. Next time, the switch would have altered the initial conditions, and the program would try the operations represented by ABDL. With the conditions specified above, it would then try in turn ABEH, falling (since in effect the point D now represents failure, as all ways through D fail later), ABEIP, failing, ABEIQ, failing, ACF, failing (since now also B represents failure), and finally ACGT, which succeeds. In this way the switching mechanism, of whatever sort it may in fact be, ensures that the program does not cycle, but tries every possibility in turn.

In our program there are four choices at each point which may be performed. These are arranged in a preference order. The following choice tree with failures at points F, G, H, I, C and D is of the sort used in the program.



The program would try in turn the routes ABF, ABG, ABH, ABI, AC, AD, all of which fail, and finally AEK, which succeeds.

In our program, the choice is based on the preference order of kinds of bracket group. There are four choices which are tried in turn. When the set with the highest bracketting number has been found as described above, then the successive choices are:

i) to form a bracket of the highest ranking kind.
ii) to form a bracket of the second ranking kind.
iii) to form a bracket of the third ranking kind.
iv) not to bracket the sequence at all.

Should any of these choices lead to a failure to bracket the sentence completely, then the remaining choices are tried. The operation of the choice tree assures that, in the end, either the sentence will have been bracketted successfully, or all possible ways will have been tried without success. In the case of complete failure the machine will stop for further orders.

It would perhaps be better (in the sense that less attempts would usually be needed) for the second choice to be not the second highest use of the

best bracket, but the highest use of the second-best bracket. Unfortunately there is not room in EDSAC to store more than the bracketting number and address of the best bracket, so that no record of the second-best bracket can be kept. However, unless one of the actual alternatives chosen leads to success, in which case there is no problem, the second-highest bracket will in fact eventually be chosen as the highest bracket of a different sequence.

The state of the choice tree is represented by a number called the choice tree switch number. This consists of quartal digits with values 0, 1, 2 or 3. The first digit refers to the first bracket made, the second to the second bracket made, and so on. The value 0 indicates that the highest-ranking kind of bracket is to be chosen, the value 1 that the second highest is to be chosen, the value 2 that the third highest is to be chosen, and value 3 that the sequence whose switch number this is is to be left unbracketted at this stage. Every time the program fails in its attempt to form a bracket, this switch number is increased by 1 in the appropriate place: that is, if the program fails when trying to form the sixth bracket, say, then 1 is added in the fifth place, i.e. at the place representing the last successful bracketting. Since the number has digits 0, 1, 2 and 3 only, the addition of 1 to a 3 is interpreted as making the 3 into 0, and carrying one to the previous digit; thus 1212 + 0020 = 1222; 1213 + 0001 = 1220; 1233 + 0001 = 1300. This operation is simple addition in the scale of 4. The effect of the number is that the program on failing to bracket will not cycle, but will try again choosing the next preferred route, through the choice tree, as described above. For example, we may use this method to follow the program through the choice tree shown above. Initially the switch number

is 0000, so the program tries route ABF. It fails at F, so the last successful choice is the second. So we add 1 to the second place of the switching number, making it 0100. This directs the machine to follow path ABG. There is again failure after the second choice, so we again add 0100. The switch number is now 0200, so next time the program tries ABH. Further addition of 0100 makes the switch number 0300, and we try ABI. When this again fails, adding 0100 makes the switch number 1000, which means we try the second alternative at the first choice, that is, we try AC. This fails after the first choice, so we add 1000 to the switching number, making it 2000. This now means we try the path AD. Failure again adds 1000 to the switch number, making it 3000. This directs us down the path AEK, which succeeds.

It will be seen that the use of a switch number in this way will eventually lead to failure only if the switch number becomes 3333 before any success is met. This kind of technique has been called "backtrack" (S.W. Golomb, 'A Mathematical Theory of Discrete Classification', 4th London Symposium on Information Theory, 1960).

Section 3 of the program does the major part of the bracketting of any sentence. It is itself subdivided into four parts. The first of these sorts the incoming words into types according to habitat, and does preliminary work such as opening appropriate working stores in the computer, and bracketting together such simple pairs as a prefix and the word to which it is prefixed. The second parts deal with conjunctions. The third is the part which does the actual bracketting (the united reading is formed in Section 5) and includes the mechanism by which the appropriate bracket is selected, and also the operations of the choice

tree and the choice tree switch number. The fourth part is a sub-routine needed to calculate bracketting numbers for the third part.

3.1.0 The first substituent is examined. (3A)

3.1.1 If it has a zero habitat (see Section 5) then a mistake has been made, and control returns to the master program. (3B)

3.1.2 If the substituent has an initial habitat, then the program opens two working stores, one starting from the substituent in question, the other starting from the following substituent. Thus, if the substituent u in the sequence u v w x y z is a preposition, which has an initial habitat, then on encountering it the program opens two working stores, one starting u v w x... and the other v w x.. After this the next word is picked up and examined in turn. (3C)

3.1.3 If the substituent has a multinitial habitat, then a working store is needed which starts with this substituent. There may be such a store already if the substituent follows one with an initial habitat, but if there is not then the program starts one. The next word is then examined. (3D)

3.1.4 If the word has a special initial habitat R, (i.e. it is an article), then the previous substituent is also examined. If this has a multinitial habitat, then the two substituents are examined together to see if they can be bracketted. If this is possible then the bracket is made, and the next substituent has not a multinitial habitat, then the word is dealt with as in 3.1.3. (3E)

3.1.5 If the substituent has a special initial habitat K, (i.e. it is a relative pronoun), then again the previous (3F)

substituent is examined. If the previous substituent has an initial or multinitial habitat, then again a bracket is formed if possible as in 3.1.4. Otherwise the substituent is dealt with as in 3.1.3.

3.1.6 If the substituent has a special initial habitat U, then again, if the previous substituent has a special initial habitat U also, bracketting is attempted as in 3.1.4., and otherwise the substituent is dealt with as in 3.1.3. (3G)

3.1.7 If the substituent has a medial habitat, (i.e. it is a conjunction), then its position in the sentence is noted in a store, and the next substituent examined. (3H)

3.1.8 If a word with an L habitat is encountered, then the next substituent is examined straight away. This situation should not, however, arise as all such words should have been replaced by Section *2.* (3I)

3.1.9 If the substituent has a final habitat (for example, if it is a full stop), then a return is made to the previous substituent, and Section 3.2 entered. (3J)

3.1.10 If the word has a multifinal habitat, then Section 3.2 is entered. (3K)

3.1.11 If the word has a prefix habitat, then an attempt is made to bracket it to the following word. If this is possible the bracket is formed and the next substituent examined; if not, the program reports, i.e. stops with an error indication. (3L)

3.1.12 If the word has a suffix habitat, then a similar attempt is made to bracket it with the previous word. (3M)

3.1.13 If the substituent has an unmarked habitat, then the next substituent is examined immediately. (3N)

Sub-Section 3.2

This sub-section looks through the current working store to see if there is a substituent with a medial habitat (conjunction) present. If not, then sub-section 3.3 is entered. If there is, then the address of the last such substituent is put in the conjunction store, and the master program re-entered.

Sub-Sections 3.5 and 3.4

Sub-section 3.3 selects the bracket to be made. Sub-section 3.4 is entirely concerned with calculating bracketting numbers for use in sub-section 3.3. Initially a sequence of substituents is read into the working store. It is checked to ensure that in fact more than one substituent is present. (3W)

3.3.1 The next set is selected from the sequence. As described above the first set will consist of the first two substituents. (3X)

3.3.2 The bracketting number of the set is calculated. (3Y)

3.3.3 If this bracketting number is higher than any previous bracketting number in the sequence, then it is stored as a new highest, and the corresponding set noted. If all the sets have not been tried, begin at 3.3.1 with the next. (3O)

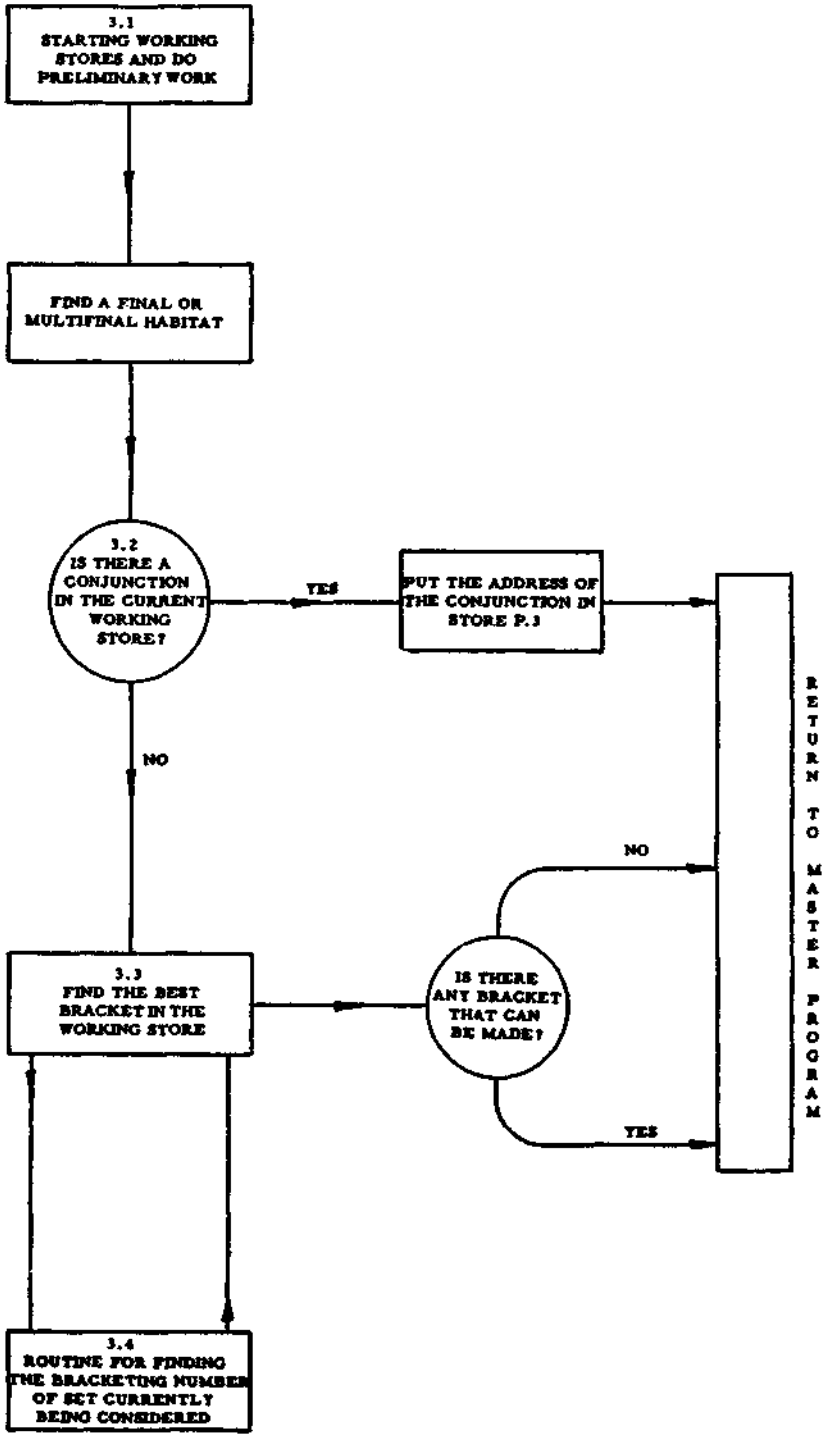
3.3.4 If this bracketting number is less than some previous bracketting number then, if all the sets have not been tried, begin again at 3.3.1 with the next. (3P)

3.3.5 If this bracketting number is equal to the old maximum, then we compare the lengths of the corresponding sets. If the new one is longer we proceed as at 3.3.3, if shorter, as at 3.3.4. If they are the same length, proceed as at 3.3.3. (3R)

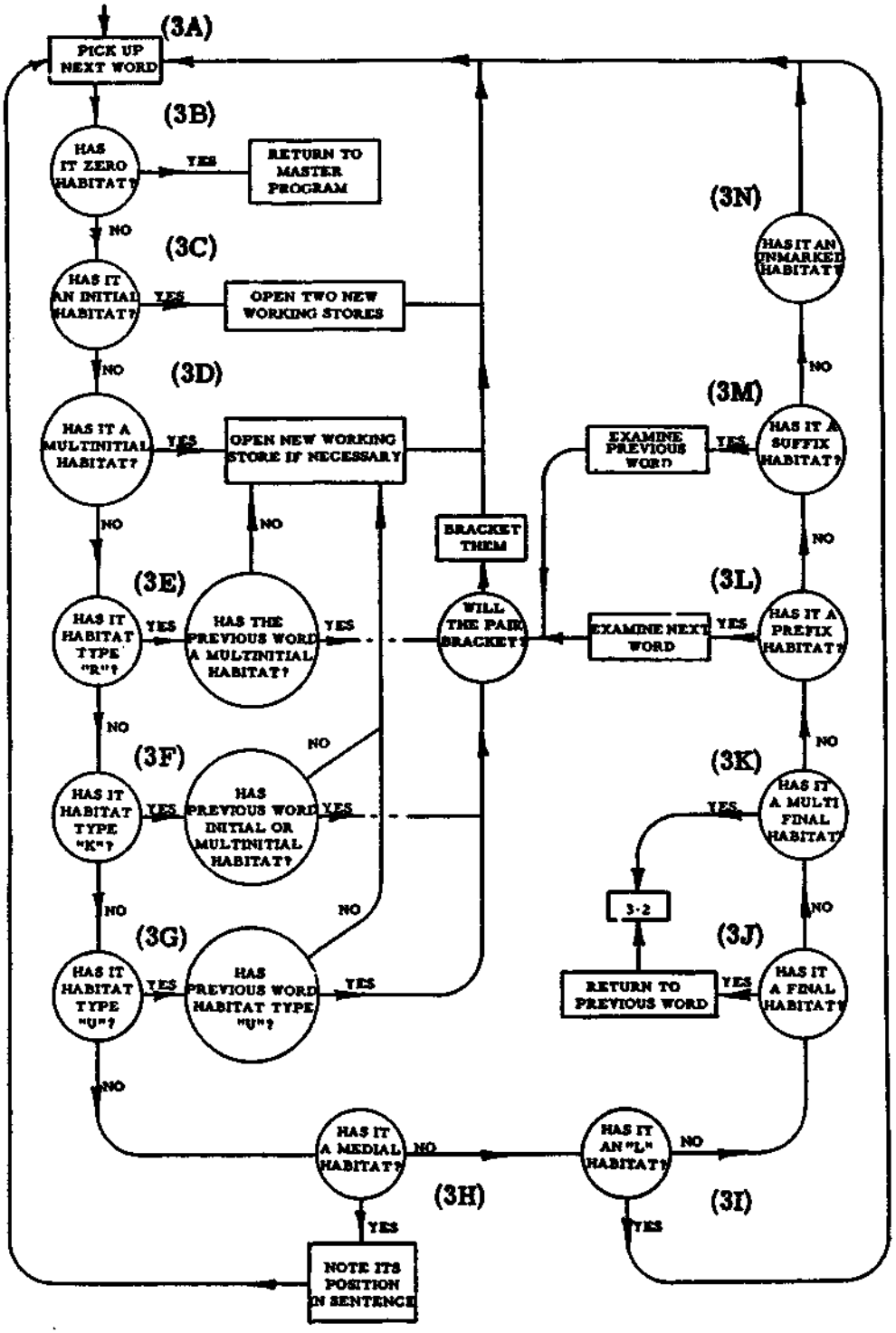
3.3.6 When all the sets have been tried, in the order described above, the best bracket is noted. (This may mean that no bracket is made). It is taken to be of the kind determined by its bracketting number and the choice tree switch number. The bracket is replaced in the sentence by a single entity with the appropriate participation class. (3S)

3.3.7 We now begin bracketting again on the modified sentence. This process is repeated until eventually the sentence is reduced to one bracket followed by a full stop. (3T)

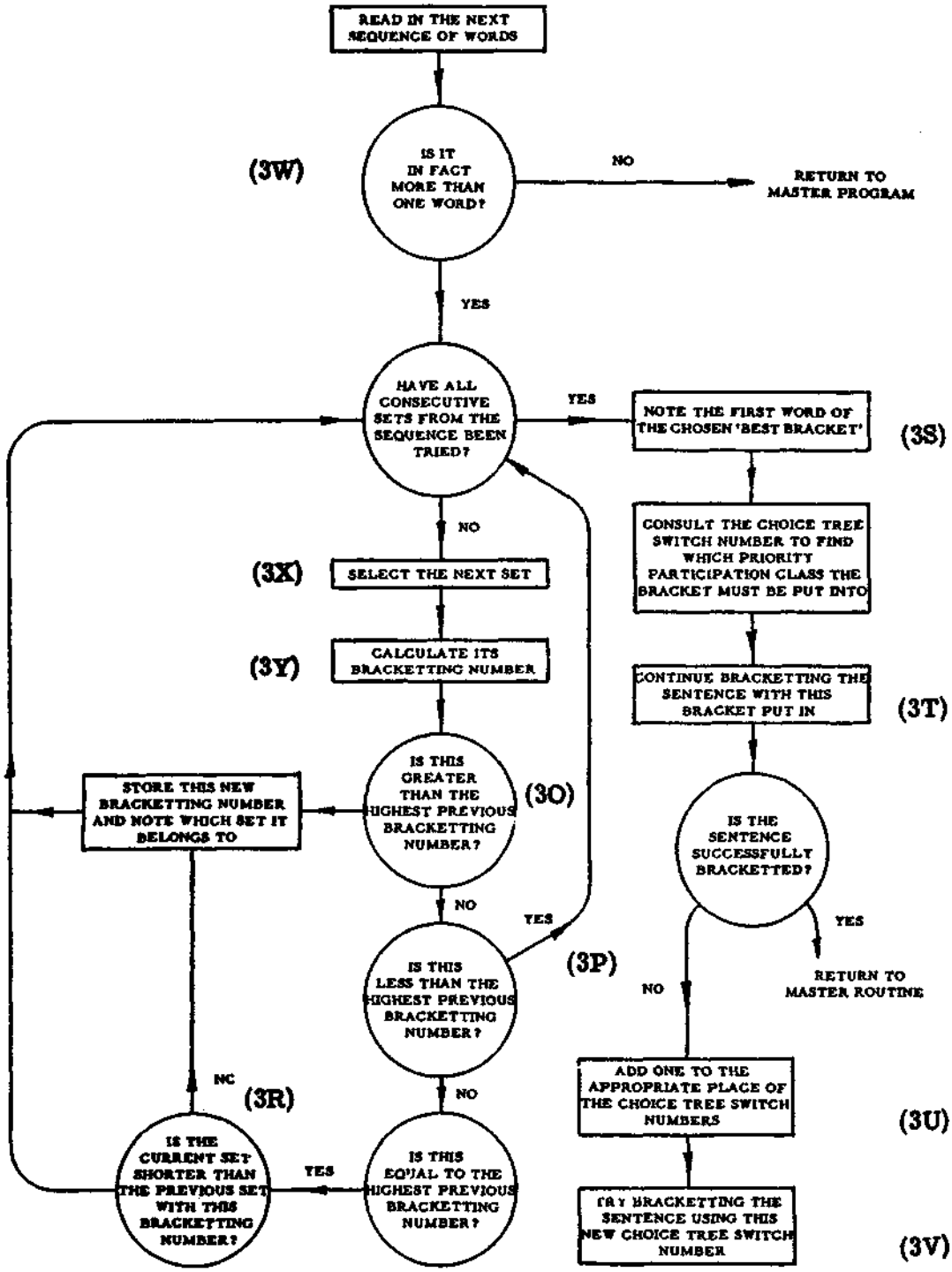
3.3.8 Should a wrong choice of bracket be made at some stage, then the program will be unable to continue bracketting until the situation of one bracket and a full stop is reached. In this case 1 is added to the choice tree switch number at the stage representing the last successful bracket. An entirely new attempt is then made to bracket the sentence, exactly as before, but using the new choice tree switch number* Eventually, either the sentence will be successfully bracketted, or all the possibilities will be tried without success. (3U) (3V)



FLOW CHART OF PROGRAM SECTION 3



FLOW CHART OF PROGRAM SECTION 3-1.



FLOW CHART OF PROGRAM SECTIONS 3.3 & 3.4

SECTION 4

This section processes conjunctions picked up in Section 3. Conjunctions may be qualified by adverbs, as in "men and occasionally women enjoy cricket". Apart from these qualifiers, however, conjunctions lie between bracket groups of the same kind. Two substituents are said to be 'parallel' if they can both occur in one of the English word classes listed in Part III a). A substituent which can only be a governor in a particular kind of group cannot be parallel with a substituent which can only be a dependent in that kind of group. 'Parallel sequences' occur when corresponding substituents from two ordered sets of substituents are parallel.

In the program it is assumed that a conjunction, or a conjunction with its qualifying adverbial group, occurs between two parallel sequences. A pair of parallel substituents is treated as if the substituents were sequences. It may be the case that words are omitted from one of the sequences. Thus, for example, "apples were given to the boys and pears were given to the girls" is usually shortened as "apples were given to the boys and pears to the girls". In such cases the program will replace the latter by the former. Sentences such as "boys and girls went to the dance", on the other hand, do not require expansion.

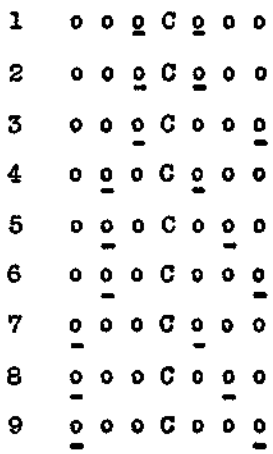
This section has not been mechanically tested, but it is apparent from rule of thumb experiments that it has not sufficient scope to deal with a large variety of conjunctive sentences. It may fail partly because in attempting to find parallel sequences it may select the wrong ones from a number of alternatives. The correct sequences must be bracketted by the

normal procedure. For example, "the kitchen sinks" could be a clause, though it is more often a nominal group. If it is found as a parallel to another clause, the normal usage may be overlooked, and the program may form a wrong bracket. The program may also fail if the working store does not contain all the substituents in the sequences: for the first substituent in the store will be one with an initial habitat, and this may not be the first substituent in the sequence preceding the conjunction.

Another procedure which is designed to overcome these difficulties is at present being tested by rule of thumb. If satisfactory results are obtained the program described below will be modified.

The program representing the first version of the procedure is as follows:

4.1.0 The substituents in the working store on either side of the conjunction found in Section 5 are scanned, a pair at a time, to find a parallel pair. The scanning order used is illustrated below on a seven member sequence:



When a parallel pair is found Section 4.2.0 is entered.

4.1.1 If no pair of parallel substituents is found a new working store starting with the word following the conjunction is opened and Section 3 re-entered.

4.2.0 If two parallel substituents, say x and y, are found, the substituents in the working store are examined to see if x and y are the last members of two parallel sequences: x - 1 and y - 1 are examined to see if they are parallel, x - *2* and y - 2 and so on backwards until one of the following points is reached: either the first word in the store as x - n, or the conjunction as y - n, or a non-parallel pair. The sequences found are recorded.

4.2.1 If the first member of the sequence dependent on y does not immediately follow the conjunction the items between it and the conjunction are examined to see whether they can form an adverbial group. If so, they are bracketted with the conjunction. If not, the bracketting has failed and Section 3 is re-entered.

4.2.2 It may be the case that there are parallel sequences but words have been omitted from the second sequence. These cases are dealt with as follows: when a non-parallel pair is reached in 4.2.0 x is moved back until a substituent corresponding to the y - n under consideration, say x - n, is found. Copies of the items between this x - n and the last paired substituent x - m are inserted between y - n and the last prepared substituent.

In this way a sequence dependent on y is produced which is properly parallel to the dependent on x.

4.3.0 The sequence dependent on x is examined to see if it can be bracketted. If not, the attempt to bracket round the conjunction

is abandoned. A new working store is started from the substituent following the conjunction and Section 3 is re-entered. If the sequence can form a bracket group, the bracketting is carried out.

4.3.1 If the sequence dependent on x can be bracketted, the corresponding sequence dependent on y can also be bracketted. Finally the bracket containing the sequence in front of the conjunction, the conjunction itself, and the bracket containing the sequence after the conjunction are all bracketted together. This bracket is assigned the same participation class and habitat as the sequences.

4.4.0 When this has been done, Section 3 of the program is re-entered.

SECTION 5

This section of the program forms the united reading in the united reading store. This consists of a computer 'word' representing each original text word, and a computer word representing each bracket group that has been formed. The bracket groups are subsequently treated as if they were text words.

Suppose for example that the four words a b c d have been bracketted together. The united reading will contain the computer words representing a b c d and also (abcd5) representing the bracket group. The computer word representing the bracket group also gives its habitat and participation class and indicates the number of words in the group. This count includes the bracket group itself.

Next suppose that in the sequence a b c d e d and e were first bracketted, that this group and c were combined and finally that a and b were bracketted with the previously found group. The united reading at each stage of the bracketting would be as follows:

Sentence Store	United Reading Store
(i) abcde	
(ii) abc(de)	d e (de3)
(iii) ab(c(de)))	d e (de3) c (cbded5)
(iv) (ab(c(de)))	de (de3) c (c(de)5) ab (ab(c(d e)8)

Again every bracket group contains a number indicating the total number of substituents including itself. Thus (c(de)) is marked 5, since the group contains the substituents, d, e, (de), c, and itself, (c(de)).

As a second example, suppose that in the sequence a b c d e, a and b will bracket, so will d and e, and that the entire sequence will then bracket. The final united reading will be

c d (cd3) a b (ab3) e ((cd) (ab)e8).

The count in the last bracket group is 8 since it includes c, d, (cd), a, b, (ab), e and ((cd)(ab)e).

The habitat of a bracket group is obtained from the table given below. The habitat of a two-word bracket, for example, is found in the row containing the habitat of the first member and the column containing the second.

	Second word's habitat										
First word's habitat	A	E	F	H	K	M	O	R	S	U	Z
A	A	-	-	-	-	-	-	-	A	-	A
F	-	E	-	-	K	-	E	-	F	-	F
E	E	E	-	E	E	-	E	-	E	E	E
H	-	-	-	E	-	-	-	-	H	-	H
K	-	E	-	-	-	-	K	-	K	E	-
M	A	E	F	H	-	M	O	-	-	U	-
O	E	E	-	E	K	-	E	R	O	E	E
R	E*	E	-	-	E*	-	E*	-	R	E*	-
S	A	E	F	H	K	-	O	R	-	U	-
U	E	E	-	U	-	-	-	-	U	U	H
Z	-	-	-	-	-	-	-	-	S	-	-

"-" indicates an excluded pair of entities.

E* indicates that not only can we ascertain the habitat E but we can

also ascertain the function of a noun attached to it.

Thus, suppose for example we have a bracket group (ab), where a has habitat M and b habitat U; as can be seen from the table above, the group has habitat U. Similarly, if c has habitat U, and d habitat Z, the bracket (cd) has habitat H.

In a three or more member bracket group the combined habitat of the first two members is found. This is then taken to combine with the habitat of the third member. For example, suppose that in the four member bracket group (abcd) a has habitat U, b habitat Z, c habitat H and d habitat S, a and b have combined habitat H; when this is combined with the third member's habitat H a combined habitat E is found and finally habitat E combines with the last member's habitat S to give habitat E. The combined habitat for the bracket will thus be E.

Certain combinations of habitats are marked with a dash "-". This indicates that the items cannot in fact be correctly bracketted together. Section 3.1 1 is designed to discover and rectify such errors.

According to its kind every bracket group formed is assigned a participation class. This is obtained by look-up from the table below.

Kind of bracket group	participation class
2	130 02000 0000
3	100 00200 2212
4	100 00200 2202
5	100 00200 2212
6	201 01031 0101
7	201 01031 0101
8	100 00200 2202
9	100 20010 1000
10	130 10010 1000
11	130 10000 0000
12	100 00000 0010

Thus, suppose for example that a and b form the kind of bracket group marked Z; this group can occur as a governor in other groups of kind 5, as a dependent in kind 1, or as either a governor or dependent in kind 2. As in Section 3 we give it the participation class 130020000000, where each position represents a kind of bracket group, and the digits 0, 1, 2 and 3 indicate respectively whether it cannot occur in the group or whether it occurs as a dependent, as a governor or as either. As a further example suppose that c and d form the kind of bracket group marked 12. This occurs as a dependent in kind 1 or kind 11 and nowhere else. It has been found unnecessary to indicate a participation class for bracket groups of kind 1 as these are special cases and are dealt with elsewhere.

To sum up, the united reading consists of the original text words plus each bracket group, treated as a word. The latter will have habitats

according to the habitats of the components, participation classes according to their kinds and counts recording the number of their substituents.

It will have been noted that the united reading tends to turn the sentence back to front. The program has been deliberately designed to do this, since it has been shown that in any language bracket groups are more often formed from the back forwards. This is to say that in any sentence a b c d e f g for example, it is more likely that the sentence will bracket as (a(bc(d(d(fg))))) than as ((((((ab)c)d)e)f)g).

The design chosen, therefore, while no better logically than one which does not change the order, is generally quicker in practice.

The operations are carried out in the following order:

5.1.1 The working space in the computer is cleared. This section is entered from Section 3, where the items to be bracketted were selected. A first check is made to verify that at least two items are present. The kind of bracket group to be made is ascertained from the bracketting number. (5A) (5B)

5.1.2 The length of the bracket is calculated from the number of components. (5C)

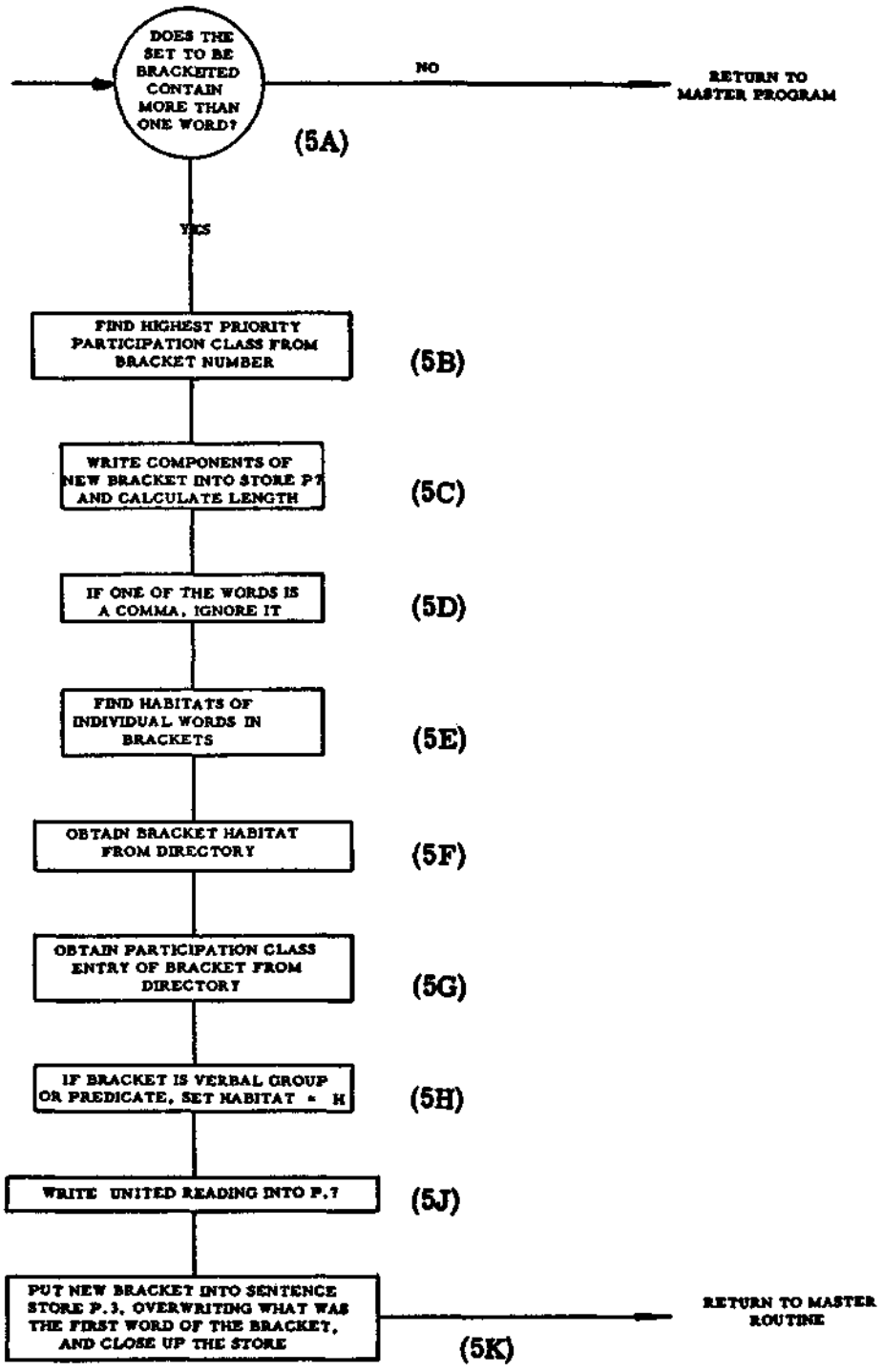
5.2.1 Commas are ignored. (5D)

5.2.2 The habitats of the substituents are found and the habitat of the bracket group found from the directory as described above. (5E) (5F)

5.2.3 The participation class of the bracket is found from the appropriate directory. If the bracket group is a verbal group of a predicate, the habitat is set at H, whatever the habitat calculated in 5.2.2. (5G) (5H)

5.3.1 Section 5.1 is now re-entered, and the united reading recorded. (5J)

5.3.2 Return to the master program. (5K)



FLOW CHART OF PROGRAM SECTION 5