UNIVERSITY OF LONDON

IMPERIAL COLLEGE OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTING AND CONTROL

Heuristic Methods in the Analysis of Program Statements

by

Derek Partridge

A Thesis Submitted for the Degree of Doctor of Philosophy

August 1972

Abstract

A generalised system for language analysis and local modification is described.The analysis can be in terms of any a priori unit of the language;such as a statement in the FORTRAN language.

A description of the language is provided as input data from which the system constructs a tree-structured hierarchy. This hierarchy then directs and controls the subsequent analysis and possible modification of particular units in the given language.At the same time the system collects detailed information on the structure of all units analysed.

Periodically,the normal process of analysis is suspended, and the analytic information stored within the tree-structured hierarchy is utilised by mechanisms designed on heuristic principles to optimise the subsequent processing.The optimisation derives both from modifications to the basic tree-structured hierarchy and also insertion of extra control information within this hierarchy.

This results in an increased processing rate for the units analysed and an improved ability to deal with errors,including 'errors' which are definitionally correct but very 'unlikely'.

The analyser is applied as a recogniser for statements written in the FORTRAN language by programmers working in industrial and university enviroments,comprising a 'normal' computer workload in those enviroments,and the results of the analysis are presented and discussed.

CONTENTS

Acknowledgements

I wish to thank my supervisor, Mr. E.B. James for his constant interest, encouragement and guidance during the last four years.

I am also indebted to my wife, Azar, for relieving me of all domestic chores whilst this thesis was being written.

Finally, I acknowledge the financial support of the Science Research Council, and wish to thank the trustees of the IBM Support Fund for the financial assistance provided during the last year.

Note on thesis structure

Corresponding sections and sub-sections within each chapter cover corresponding facets of the overall project from the viewpoint implied by the chapter heading.Thus section numbers carry semantic implications which will facilitate a search for discussion of any particular topic from different viewpoints. For example,an introduction to the 'strategy tree' is found in 1.22,a detailed description is given in 4.22 and the results of its usage in 5.22.

We have aimed to provide each discussion of a particular feature with a reference back to the original detailed description by means of the appropriate section or sub-section number within parentheses.In chapter 4 (and occasionally elsewhere) reference is made to the system implementation details within the Appendices.These references consist of the word APPENDIX (or abbreviated to APP.) followed by the particular Appendix number and perhaps a section number.So a reference such as (APP.III,ii),refers to section ii) within Appendix III.

References to figures within the same chapter are given as the figure number only.References to figures outside the current chapter are given as the figure number preceded by the appropriate chapter or Appendix number.

1.0 Introduction and outline description of the system.

The following section contains a general description of the problem selected and the method of attack.A detailed description of the implementation of the following ideas will be found in the equivalent sub-sections of chapter 4.

1.01 An introduction to adaptive analysis

The advent of high level languages in computing has moved the possibility of computerised problem solving far beyond the specialist domain of computer science,and paved the way for 'non-professionals' to utilise this very powerful tool. The professional programmer has no difficulty in writing programs in a language such as FORTRAN which can be successfully executed after very few correction runs.The error indications from current compilers provide a perfectly adequate service to enable the rapid correction of programming mistakes at least of the syntactic type.

However,in Universities and other research centres there has been an explosive growth in the number of computer users who do not wish to regard the use of the computer as an end in itself.They wish to use computers for limited periods on specific problems and wish to do this with a minimum of learning effort.Also,particularly in Universities,the users always include a high proportion who are in the process of learning to write programs rather than performing specific calculations.

Thus we envisage the problem as an attempt to make computerised problem solving easier for large numbers of comparatively inexperienced users,for whom a computer run which goes into execution is a welcome rarity.The direction of enquiry is to see how far the automatic analysis of program statements may be developed to exploit the 'pattern of language usage'.

The method of attack is to adapt and optimise the future processing of statements by making use of the experience gained from past processing.

1.02 Current methods of analysis and the experience tree

The normal process of syntactic analysis involves the determination of the statement structure by using a processing algorithm which expresses the total structure of all possible statements in the language.The initial objective of such a system is usually to divide the input statements into one of two mutually exclusive classes,correct and incorrect.Correct are assigned a unique structure whilst incorrect statements will usually result in an 'error exit' of some type.

Usually the structure of the language is expressed more or less explicitly as a tree structure.This is provided once and for all as part of the algorithm.The rigid nature of this tree structure means that the process of analysis remains unaltered and so every statement is analysed as if it were the first.Important things such as the order in which alternative possible analyses are attempted remain fixed.The error exits are also specified in advance.The actual rigid structure specified is usually based upon previous experience of the relative likelihood of the various possible sub-structures and errors.

We propose to extend this method in several directions. The general principle is that the tree structure is no longer static but dynamic.An initial structure is provided and is altered in many different ways as a result of the continuing process of matching the incoming statements.At a particular point in time therefore the structure of the tree reflects the total experience of the process of analysis

that has been carried out on all incoming statements.Thus it is convenient to call it 'the experience tree'.

1.10 The pattern of use of a programming language and its exploitation in language analysis

The basis for analysis embraces a consideration of the gross structure of all the actual statements previously analysed as well as the defined structure of the language provided ab initio in the statement analyser.We aim to use information which arises from programs submitted by all users of the system whether these programs were right or wrong by conventional standards.These features we group under the heading 'pattern of use'.

1.11 The particular features of the usage pattern that are exploited

The particular features of the usage pattern that we concentrate upon can be usefully divided into three sections: first,the fact that only a small proportion of the 'possible' structures in a language are commonly used;so during the analysis,a search for a match is conducted principally within a small subset of specific instances of these common structures, and backed up only when necessary by a search through structures representing the rest (the majority) of the language,condensed into some rather general form of representation which needs to be examined on infrequent occasions only.

Secondly,there is an overall stability in the relative proportions with which the various possible structures occur; and thus the likelihood of future occurrence of the various types of program statement can be predicted from the frequency of past occurrence with a high degree of accuracy.This also means that as soon as the experience tree has been altered sufficiently to mirror the current particular usage pattern,

subsequent alterations that may be necessary will detract little from the overall processing efficiency.

Thirdly,redundancy;the incoming statements contain a certain degree of redundant information which can be ignored if we are aiming to speed the analysis of correct and 'trivially' incorrect statements,or utilised if required to assist in the interpretation of incorrect statements.

1.12 Improving the experience tree structure

A fundamental requirement of the analyser is that the experience tree is automatically restructured in the light of information gained during the analysis of all previous incoming statements.This automatic restructuring is aimed at continually minimising the complexity of the total matching process,hence increasing its efficiency.

1.121 The 'branch swopping' technique

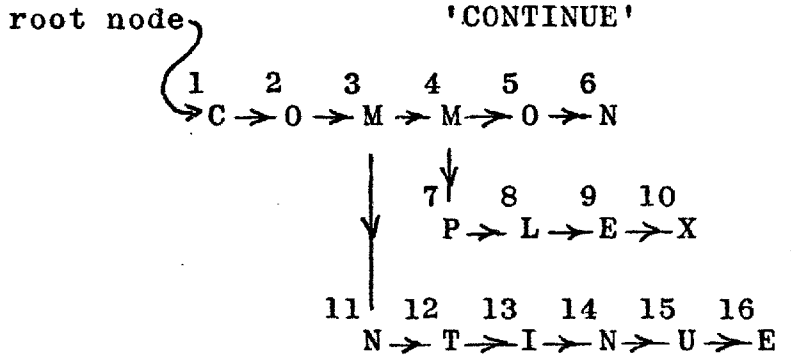
A natural assumption is that the commonest structures should be dealt with most efficiency at the expense of those which occur less frequently.This can be realised by the use of a 'branch swopping' technique.The set of alternative nodes at any point in the tree which represents the set of all possible alternative structures present at a certain point in the input statement,can be continually rearranged so that those most frequently required are nearest to the root of the tree and therefore are reached most quickly.Naturally,when a set of alternative nodes are reordered in this way,their attachments to subsequent nodes remain undisturbed.

This branch swopping rearrangement of nodes can be continuous in the sense that a node is moved to a position of higher precedence where possible at every successful match: or an examination of successful match frequencies and subsequent reordering of alternative nodes can be carried out periodically,say,at intervals of 10,000 thousand statements

processed.

As an example of the branch swopping mechanism;consider the partial syntax tree in figure I,composed of the FORTRAN keywords,'COMMON','CONTINUE' and 'COMPLEX'.The particular structure illustrated in figure I,gives precedence to 'COMMON' with respect to 'CONTINUE' (node 3 is in a higher precedence position than the alternative node 11),and also with respect to 'COMPLEX' (node 4 is in a higher precedence position than the alternative node 7).

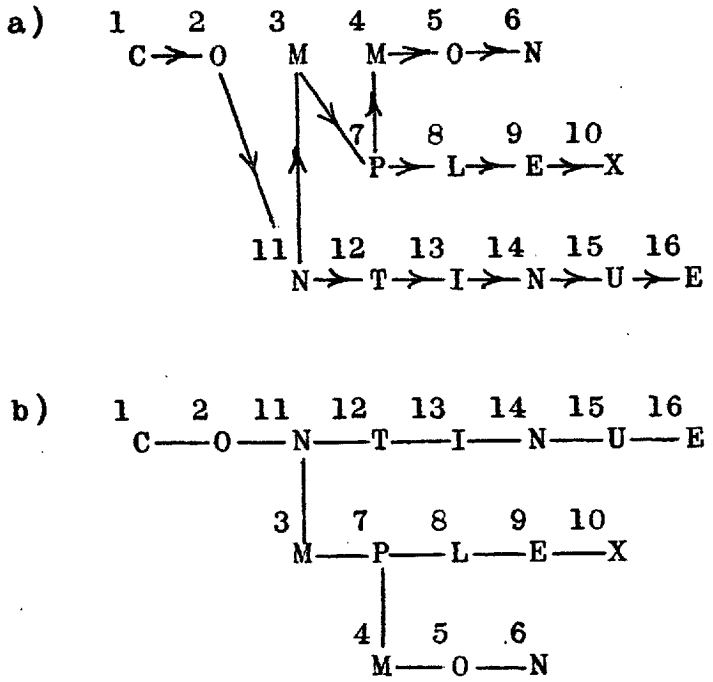
Fig. I - a partial syntax tree of the FORTRAN keywords,'COMMON','COMPLEX' and 'CONTINUE'



note: see Fig. III, 4.04 for details

Then if subsequent match frequencies indicate that the statement 'CONTINUE' occurs most frequently,followed by 'COMPLEX' and finally 'COMMON';the link from node 2 to node 3 would be 'swopped' to link node 2 to node 11 and the link between the alternatives,nodes 3 and 11,would be reversed to link from node 11 to node 3.Similar action is taken with nodes 3,4 and 7.As a result of the action of the branch swopping mechanism on figure I we obtain figure IIa,which we would normally represent as in figure IIb to conform to our convention that horizontal lines link from left to right and vertical lines link from top to bottom.

Fig. II - fig.I after 'branch swopping'



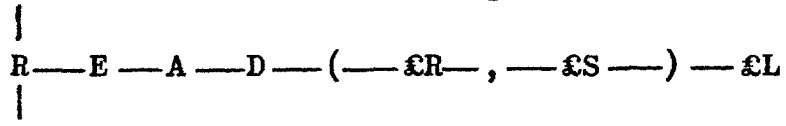
1.122 The automatic removal of unwanted generalisation - the 'promotion' technique

A useful and particularly interesting extension of the branch swopping technique stems from the necessity of utilising subroutines within the experience tree.By 'subroutines' we mean the removal of common sub-structures,specifying them in one place only and then referencing these subroutines or subtrees,from the appropriate positions within the total structure.The experience tree is thus a hierarchical structure of subtrees or subroutines referencing and cross-referencing each other to an arbitrary depth.Clearly the use of subroutines runs contrary to our aim of particularising and hence increasing the efficiency of the total matching process;but,it is unavoidable if we are to specify the tsructure of any real computer language within a reasonable or even finite,space.

Examples of convenient subroutines within a structure that specifies the language FORTRAN;that section of the experience

tree,S,which can match a FORTRAN statement number;that which can match a FORTRAN variable list,L;and that section which can match the set of numbers that can correspond to the possible reading sources,R.All these three subtrees might occur in the FORTRAN input statement in figure III.

Fig. III - a FORTRAN input statement referencing three subtrees



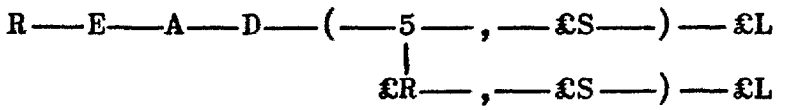
In this example the tree item illustrated from the experience tree consists of the immediate symbols,'R','E','A','D','(',',' and ')':and the subtree references '£R','£S' and '£L',which of course,reference the subtrees labelled 'R','S' and 'L', respectively.

Now clearly the branch swopping technique (1.121) will be useful for moving the frequently required nodes to positions of higher precedence,where possible,within every tree and subtree.But there is the further possibility of simplifying the matching process even more by 'promoting' the very common subtree sub-structures out of their particular subtree and inserting them at each position within the total experience tree structure from which the subtree is referenced:but in a position of slightly higher precedence than the original subtree referencing node.

As an example of this promotion technique,consider the subtree R,referenced from within the FORTRAN statement illustrated in figure III,that is READ(£R,£S)£L.If the normal input source (given immediately after the left bracket) is '5',then the total subtree R may consist of say,the numbers 1 to 20 with '5' being cited in the overwhelming proportion of cases analysed.Thus the action of the promotion mechanism

might result in the structure illustrated in figure IV.

Fig. IV - the FORTRAN input statement after the promotion of the substructure '5'



where '5' is an immediate symbol and the alternative '£R' references the subtree R which is to be examined if 5 does not occur within any particular statement analysed.

The structure illustrated in figure IV will produce a simplification of the total matching process,for in the majority of cases the immediate symbol 5 will be the one required and hence we do not have to obtain the subtree via the reference node before it can be examined for particular immediate symbols.Thus the inefficient generalisation is removed for the majority of analyses.But of course,in general the simplification that results from the promotion technique is gained at the cost of the extra space required to specify the resulting expanded structure.So the final structure must represent a compromise between the simplification of the total matching process and the total size of the experience tree.

The success of the two restructuring techniques outlined depends critically upon the extent to which the pattern of usage of the statements processed exhibits the first and to a lesser extent the second of the features described earlier (1.11).Clearly these techniques would not be useful if the distribution of a particular input feature was uniform over a range of possibilities.

1.13 Levels of confidence and control of statement analysis

The improving techniques described above (1.12) imply that there is available a count of the previously successful match frequencies;an obvious place to store and maintain these 'frequency counts' is within the experience tree itself. Thus frequency counts are stored throughout the experience tree,and a frequency count is stored and associated with each item in every tree and subtree that comprises the total hierarchy.

Apart from use in the'improving' strategies there is an important further use of the frequency counts.They are utilised during the matching of incoming statements to provide a measure of the relative likelihood of success with any particular sub-structure - or from another viewpoint,a measure of 'confidence' that any particular sub-structure will prove to be correct.Thus there is available at all times during the processing of a statement,some value representing the confidence in an overall successful match which can provide a criterion for continuing or abandoning the current matching process.

Consideration of these confidence levels has a useful,if not essential part to play in the production of an efficient system which can 'learn' to analyse statements with an arbitrary number of errors,given sufficient learning and processing time.There will be statements which resist a satisfactory analysis because they differ too much from any structure in the experience tree.It is important to detect these statements as soon as possible so that they do not waste a significant time in the matching process.

1.14 Non-exhaustive analysis and processing efficiency

The fact that the distribution of the actual features of a language that are commonly encountered is far from uniform

amongst all the possible features,suggests the possibility of simplifying the total analysis by tentatively neglecting the possibility of occurrence of the very unlikely features. Clearly such a procedure could be fraught with danger and adequate safeguards must be developed concurrently.For example,we must be able to accommodate a sudden surge in the popularity of a previously uncommon feature.

We are investigating the possibility of a compromise between increasing the efficiency of the analyser and decreasing the number of mistakes that result.The usual stand taken,is that the efficiency of a system may be increased as far as possible whilst maintaining the theoretical number of mistakes that the system can make,at zero;that is, no 'allowable error' can be entertained.

We plan to relax this stipulation and move the position of compromise into the area of positive proportions of allowable error,in order to assess the practical feasibility of such a move.

The admission of a proportion of allowable error might be expected to increase the efficiency of the matching process in several different ways.The general idea behind them all is that while the matching process is continuing successfully, it makes use of a language description containing only the common features,and only brings the rest of the language structure into consideration when the level of confidence drops due to mis-matches.

The particular method by which allowable error can be introduced into the system can be considered as an exploitation of one type of redundancy,that is statistical.Substructures within the experience tree are,to a first approximation, ignored when the defined set of alternatives in the language structure which they represent have been found to occur

in statistically insignificant proportions.In practice for a particular application,'statistically insignificant' will mean substructures associated with frequency counts below a certain value.

The increase in efficiency stems from the effective reduction in size of the working definition of the language to be analysed,which results from 'removing' the very infrequent features for at least a first analysis,which implies the introduction of allowable error in failing to recognise these features if they actually do occur.

1.20 The restoration of statements which contain errors of specification

The analyser has the ability to process and thus effectively restore statements that contain errors of specification. Several of the techniques described above will be reconsidered in the context of processing incorrect input statements,to demonstrate how they are expected to both increase the speed and improve the quality of error 'correction'.Errors can be categorised by the method by which they are processed into two types,trivial and non-trivial,and will be described separately below (1.21 and 1.22 respectively).

1.21 Structural redundancy and efficient processing of trivial errors

The statements of a program presented to a computer for processing are not always perfect,especially within the learning enviroment that we are considering.It is a fundamental assumption of the work described here that we do not expect perfect specification of the incoming statements.Prime concern is with extracting the 'meaning' of the statements analysed which implies that within the matching process the fit is not necessarily or even usually,perfect.The fact that statements

can be 'made sense of' (matched) without a perfect fit means that there must be a certain amount of redundancy within them.

As a result of our approximate matching techniques, error 'correction' does not have quite the usual meaning within our analyser. We are not satisfied merely with classifying the incoming statements into one of the two mutually exclusive classes, correct or incorrect, which then invoke normal or error processing techniques, respectively.

Trivially incorrect statements can be processed, that is, the critical features (the ones that convey the essential meaning) can be extracted with no loss in efficiency over the processing of completely correct statements. A logical consequence of the high efficiency of this technique is that we do not indicate the occurrence of trivial mistakes in the incoming statements.

The realisation of this approximate yet descriminating matching is seen in our 'confidence jump' technique. This mechanism quite simply exploits the structural redundancy of the particular language definition that comprises the experience tree. This tree structure definition makes any structurally redundant features of the language conspicuous as nodes that specify immediate symbols and are not associated with any alternative structure.

Thus when analysing the incoming statement we have at any particular point in the process, a current input statement character to be matched against an immediate symbol in, or referenced from a current tree node or one of its alternatives. Now if the current tree node contains an immediate symbol and has no alternative nodes associated with it, then it can be assumed that the match has been successful and control transferred to the next tree node and input statement character. Clearly this mechanism extends easily and more

efficiently to longer sequences of structurally redundant symbols.

As an example,consider the partial syntax tree illustrated in figure I.The nodes 1,2,5,6,8,9,10,12,13,14,15 and 16 are all structurally redundant.Thus if the first four successive characters of an input statement are 'C','O','M' and 'P' which are matched successfully with the tree nodes 1,2,3 and 7;then the statement if corredt or only trivially incorrect (that is,the next three characters are not 'L','E' and 'X'),must be 'COMPLEX' and there is no need to examine the final three input statement characters.

1.22 Non-trivial errors and the strategy tree

The method of analysis described has two components:the experience tree and an algorithm which matches the incoming statements against this tree structure.The structure of the experience tree is continually optimised (1.12) and so it seems reasonable to attempt a similar optimisation of the matching algorithm.

So far the matching algorithm described has been assumed to proceed smoothly,but when it does not and thus we have a non-trivially incorrect statement to be analysed,we do not immediately divert this statement to the'nearest' error exit,but attempt to 'force' a match against the experience tree by modifying the matching algorithm.The various modifications or strategies will in general be different for dealing with different types of error.We can arrange these different strategies in a hierarchy similar to that of the experience tree and there seems no reason why the 'strategy tree'* should not be manipulated in a similar manner. Thus successful strategies can be promoted so that they are tried more often,and unsuccessful strategies can be demoted or deleted.

*the 'strategy tree', so named for historical reasons, is a simple list in the present realisation

General strategies that might prove useful are,'assume one character has been omitted from the input statement',or 'assume two successive characters have been altered within the input statement'.There is at least one further feature that would seem to be necessary in the modification of a matching strategy,and that is the position within the input statement and the experience tree from which the force match should be initiated.For it is by no means necessary that the error occurs at the position in which it is first detected; thus we include within any strategy a parameter to accommodate this factor.

The confidence levels (1.13) also have a significant contribution to make to the process of error correction when using the force matching technique.The matching strategies are arranged and will therefore be tried in order of decreasing likelihood and hence also in decreasing confidence that they can effect a useful correction;the confidence of the matching process at any particular point would seem to be some function of the confidence level of the particular strategy and of the particular experience tree sub-structure.Thus an attempted matching could be abandoned when some function of these two levels diminishes to some preset threshold value and a match has not been obtained. Then,although the force match will be continued with a less likely strategy,the chance that a fit might be obtain on a high likelihood sub-structure of the experience tree could result in an overall higher confidence correction than would have been obtained by pursuing the previously abandoned matching.

By using the confidence levels in this manner we expect to produce high likelihood corrections which are superior to the initial results of the system,when it has not learned

the pattern of language use.

1.30 The end product

The analysis of an input statement involves approximately matching it to the most likely sub-structures of the experience tree.A special case of the approximate match will be a perfect fit.Once the matching process has been completed satisfactorily we assume that the matched sub-structures of the experience tree define what was 'meant' by the input statement.

This information is normally used to direct the conversion of the statement analysed into a form more directly executable by the machine.In the present project we define 'output' as a string of characters which represent the original statement in some standard form directly acceptable to an existing translating mechanism.

During the matching process the matched sub-structures direct the extraction of the salient features from within the input statement.Thus on completion of the matching we have a set of features extracted from the input statement and various experience tree sub-structures distinguished as the ones that were matched.The frequency counts within each of these matched sub-structures is incremented to record this last successful match and the string of characters to be output is controlled by a 'pattern' which is associated with the main matched sub-structure:a main sub-structure is in effect a skeleton of each main type of statement within the language.

The pattern directs exactly what is output,in the form of specific characters and the values of features extracted during processing.The particular main sub-structure that is matched directs the analyser to an associated pattern.

Thus the processing that the system performs can be radically altered by simply changing the patterns associated with each main sub-structure.For syntax checking,each pattern will consist of an exact copy of the associated experience tree sub-structure and will direct output of the extracted features in positions equivalent to those from which the substructure directs the extraction.But obviously the patterns can be used to output innumerable different strings of characters containing possibly multiple occurrences of some or all of the extracted features,if any.That is we can convert into a 'different' language.

Furthermore,a 'second' end product is obtained as a by-product of the analysis of incoming statements.By virtue of the frequnecy counts contained throughout the experience tree we have available for further analysis a detailed description of the pattern of usage of the language.

2.0 Survey and discussion of related work.

The problem and method of attack that we have chosen does not seem to fit easily into any single area of computer science.Various parts of our problem have close associations with parts of many problems represented within previously published work,but there appears to be little 'wholesale' correspondence between our work and any previous publications of which we are aware.Consequently we have decided to discuss each facet of our system and its relation to previously published work independently.

Related work with a similar basic approach and underlying principles is discussed in chapter 3.Work with much more specific and detailed similarity is cited and discussed at the relevant points in all of the subsequent chapters.

2.01 Binary tree searching algorithms

Most of the work known to us is concerned with purely theoretical procedures for minimising search times and consequently authors tend to base their results upon the assumption that the distribution of the possible items to be searched for is uniform.One author (STANFEL 1970) even goes so far as to elicit support for this uniform distribution assumption by suggesting that this is the most important case; whereas to us it would seem to be the most unlikely case and only probable in the sense that it is usually among the first simplifying assumptions to be incorporated in any theoretical system.

Given the above assumption,SUSSENGUTH (1963) only considers trees in which the terminal nodes are all at the same level and varies filial set sizes (our sets of alternative nodes are filial sets) to minimise search times.

PATT(1969) removes Sussenguth's restriction on the position of terminal nodes and derives minimum search times for two cases; first,if the filial set of each node is specified a priori then they should be ordered according to the number of terminal nodes reachable from each;second,where the entire structure of the tree is changeable.

STANFEL (1970) again extends Sussenguth's work and considers the variation of the node "key" (our data characters correspond roughly to the node "key") sizes to obtain minimum search times.He obtains a result for the special case of equally probable key requests.A similar type of optimisation is described by MARTIN & NESS(1972).

The point to note in all the above work is that the tree structures are manipulated to achieve a minimum search time when there is a uniform distribution of input to be matched;whereas in our system the initial definition of the language and the information from previous processing determine the current tree structure,and one,if not the,basic premise upon which our system stands is,that the distribution of incoming statement types to be processed is typically far from uniform.

Descriptions of systems in which the tree structure is constrained by the items which are used to create it are found in ARORA & DENT (1969) and RIJSBERGEN (1971).The former authors again consider the search assuming random input and thus the results do not bear much relation to our work. Rijsbergen's paper describes the automatic construction and restructuring of trees and subtrees,by clustering the items from given values of "dissimilarity coefficients".He then considers "simple" search strategies as opposed to "complex" which involve consideration of previous decisions and possibly backtracking,and says,

"the distinction [between "simple" and "complex"] purely indicates the amount of memory involved",a statement which, from our experience with "complex" search strategies,is not borne out by practical work.Finally,he states that,"moderate success was achieved" with 200 documents in a document retrieval system.Clearly there is not much opportunity for comment or comparison in terms of the practical application.

We are aware of one paper which aims at exploiting non-uniform distributions of items.COFFMAN & BRUNO (1970) consider methods for the structuring of a file so that initially unknown but non-uniform access frequencies are exploited in a way that reduces mean search times.The total file is maintained in two distinct parts - a structured part containing items already accessed at least once and an unstructured part which contains all the items that have not been used.They test the algorithm by using the results of a Monte Carlo simulation for a family of "exponential decay" distributions,defined by the decay parameter ε , where the probability of the i+1 ranked item,$h_{i+1} = h_i \varepsilon$. When ε =5/6 they demonstrate a mean saving of 1/5 of the access time for an unstructured file.One difference from our structured file (the experience tree) is that Coffman's file cannot be restructured and thus they are considering only non-uniform and <u>stable</u> distributions.Now although our system will be more efficient if the distribution of items to be accessed is stable it is by no means a necessary prerequisite and the dynamic nature of our file means that it can be automatically adjusted to accommodate any changes in the distribution of items to be processed.

Many tree searching algorithms involve an examination,to

various depths,of some or all of the possible future alternatives in an attempt to select a future path that is in some sense optimum.SLAGLE (1971) has recently proposed a new version of this idea - the gamma procedure - which he explains as a generalisation of the earlier analogues,"the dynamic programming approach" and "the branch-and-bounds method" (both explained and referenced in his paper).A particularly important point with respect to our system is the reduction in the number of nodes that have to be evaluated as a result of the "dynamic ordering" of possible alternatives;a feature that we make use of to select the optimum path.In effect the 'look-ahead' is performed by our restructuring process (1.12) which reorders the alternatives such that the optimum path is encountered first during the subsequent processing.

As for dynamic reordering of alternatives,the system closest to ours is that of LONGUET-HIGGINS & ORTONY (1968), which was the work that originally motivated our particular approach to this problem,especially the branch swopping mechanism (1.121).They demonstrate a saving in search length over a rigid equivalent structure when nodes within a dynamic dictionary are moved to a higher precedence position each time they are accessed.The dictionary items are words and natural language text is used as input.

NOTLEY (1970) uses a system of frequency counts associated with strings of characters to control the contents of his "cumulative recurrence library",the searching of which is not explicitly explained,but would presumably be most efficient as some sequential binary choice structure.The strings contained in the library serve to parse an input text by selecting the longest complete string that may be

found at the front of the input text.Each time any string is parsed,all of the string counts are reduced by one and then the count of the string actually matched is increased by m, which is the number of strings in the library.Strings are concatenated when their counts reach an upper limit and deleted at a lower limit.Clearly this system has close associations with our utilisation of frequency counts (1.13);although we do not actually decrement our counts the system overall produces a similar effect.The difference is that sudden drastic changes in the file structure are less likely to be produced by our system.The significance of this is that we are more concerned with maintaining a genrally stable structure whilst Notley wants his library to quickly reflect the sequential features of its input.He suggests ways of increasing the power of his system.One amounts to our hierarchical structure of subtrees.It is stated that such a system would be able to parse any syntax that it is possible to define in BACKUS NAUR form (BACKUS 1959).

2.02 Parse algorithms

The first work that we are aware of that bears a close resemblance to our analyser is "the predictive analyser" of KUNO (1965,1966).Kuno's system will convert any 'context free grammar' automatically into the equivalent 'standard form grammar',and then for a given sentence the analyser will produce all of the possible syntactic interpretations.This is somewhat different from our system in that we only produce the 'most likely' interpretation.The value of producing multiple interpretations would seem relevant only where these are subject to some'higher authority'(e.g. human judgement), to make a final decision as to the 'best one'.Kuno's system pays the penalty in its explosive storage requirements.

VIGOR,URQUHART & WILKINSON (1969) describe a system which

will parse a subset of English into a crude dependency structure which can then be used to output sentences using different words which are obtained from a dictionary.They use a 'learning' mechanism which attaches properties to words.This mechanism learns by trial and error but because of the wastefulness of this process in terms of the number of bad sentences output they have to introduce "meta-linguistic determiners".This feature allows the operator to perform a limited amount of manipulation of the dictionary tree.The dictionary is a hierarchical structure of trees which contain two types of nodes,"or-exclusive nodes" and "and nodes",whose logical function seems to be realised in another way by our links to alternative and successor nodes (see 4.01).

The description of "the use of symbol-state tables" from DAY (1970) is a closely analogous method of parsing but using symbol-state tables instead of our experience tree(1.02).Day presents a detailed parsing of FORTRAN.A further point of affinity is that his program is written in FORTRAN,a procedure which he justifies in terms of the machine independence and generality gained in a sphere in which non-standard specialisation is one of the major problems:these sentiments we heartily endorse,for our system is written in FORTRAN for much the same reasons.Day makes no mention of the possibilities of adaptively improving or incorporating error correction into his system.

WOODS (1970) presents a lengthy and detailed theoretical paper in which he discusses many previous parsing systems and works towards the use of augmented transition network grammars for natural language analysis.He says that a transition network graph that can access other graphs (or itself) via non-terminal nodes,which he calls a recursive transition network (RTN),is "equivalent in generative power to that of a context-free

grammar or pushdown store automaton,but ... allows for greater efficiency of expression,more efficient parsing algorithms, and natural extension by 'augmentation' to more powerful models which allow various degrees of context dependence and more flexible structure building during parsing".Our basic hierarchy of trees and subtrees would appear to provide such a structure.Woods further states that an "augmented recursive transition network(ARTN)" is capable of performing the equivalent of "transformational recognition"(see CHOMSKY 1964,1965) with several advantages over the transformational approach. Clearly our system is theoretically powerful and capable of very powerful extension.In discussing the advantages of the ATN model Woods makes a point of considerable importance,since it typifies many of the current approaches in this area.Woods states that one of the advantages of the ATN model is the efficiency of representation over the usual context-free model; more explicitly,common parts of many context-free rules can be merged using the ATN model which "not only permits a more compact representation but also eliminates the necessity of redundant processing".We find this procedure commendable,and use it extensively,but only in conjunction with a consideration of the pattern of use of the language that we are describing, for "general" features may be used much more in one context than another.This means that,dependent upon the relative usage of particular instances of some generalised feature, there may be important gains in terms of processing time to be made by maintaining the most frequently used features as specific instances within the structure.This is the use of our promotion mechanism (1.122) and can be viewed as an overall increase in efficiency at the cost of the introduction of a little redundancy in terms of the structural representation.

and in the processing of unlikely features.Thus we are concerned with the removal of unwanted generalisation rather than trying to generalise and remove redundancy at all costs.

A further important advantage cited by Woods is the flexibility of the ATN model and hence its potential usefulness in linguistic research.Flexibility is an attribute that we consider important in our system although it is this 'open-endedness' which causes difficulties when trying to describe in useful terms exactly what our system is capable of.Finally Woods suggests that "in applications where natural language is to be used as a medium of communication between a man and a machine,it is more important to select the 'most likely' parsing in a given context" than to continue simultaneously all possible parses,although some means of retrieving the 'less likely' parses might be provided. Clearly our frequency counts and the resulting restructuring and confidence levels (1.13) are an approach to this problem.

Error correcting parse algorithms are described in 2.21 below,and linguistic pattern recognition in 2.12.

2.10 Dynamic systems with similar features

We divide our discussion of similar dynamic systems into, first,systems which 'learn' in a similar way to our system; and second,systems in which the 'learning' or adaptation has a similar objective.

2.11 Learning programs

In HUESMAN's (1970) study of heuristic learning methods he concentrates on,"the collection and use of stochastic information on past performance in order to improve performance",clearly a task which considerably overlaps ours.The

gross logical structure of his system bears a close resemblence to that of our system,in that he uses "a general problem solving strategy","task specific strategies" and strategies are selected on the basis of a count of past successes.These features appear to be paralleled in,our strategy tree (1.22), the experience tree (1.02) which directs control via strategy codes (4.111),and the frequency counts (1.13),respectively. Huesman considers the use of only ordinal feedback and finds that only a small number of positive reinforcements were required to produce significant improvements whilst negative reinforcement or error corrective feedback was shown to be a hindrance to learning!In our system ordinal feedback would be sufficient for the branch swopping mechanism (1.121), but not for the promotion mechanism (1.122) or any other system of re-grouping items,for then the degree of similarity or dissimilarity would seem to be indispensable.

KISS (1969) describes "a computer model of certain classes of verbal behaviour",in which the two essential components of a word retrieval system are;"evolution of the word store in time" and,"a decision stage in which a random choice is made according to the relative levels of cumulative activity". Our experience tree is very similar to Kiss' word store from the point of view of a stochastic information retrieval system which he claims can qualitatively cover learning mechanisms in a natural manner.Learning is effected by adjustment of the link values between words in the store, and which we perform with our branch swopping mechanism (1.121) in an analogous manner.

The branch swopping technique of LONGUET-HIGGINS & ORTONY (1968) which as we have mentioned earlier (2.01) underlies our branch swopping mechanism,was used by MICHIE & ROSS (1970) to reorder the operators used by their "adaptive graph

traverser".The method they used was to promote an operator whenever it was selected and to demote it in the ordering whenever it could have been selected but was not selected. The point that they stress about their work is that the method of improvement does not depend upon there being any successful searches in the program's past experience of a given problem.Experiments were performed using the "Eight-puzzle",in which 8 sliding blocks must be arranged in a bounded 3x3 array so as to form a specified goal configuration. A sample of 48 randomly ordered operators was trained on 50 randomly generated samples of the puzzle,then the operator order was frozen and the operators were split into the top 24 and the bottom 24.Duplicate re-runs were then performed using the two reduced operator sets with no reordering.The results led the authors to conclude that the operators had been sorted into a "good" and a "bad" set.

The above described project falls into the second of MICHIE's (1970a) three categories of machine learning which are:one,"rote learning,whereby previously computed results are automatically memorised for subsequent use",

two,"parameter learning,whereby a problem-solving system automatically tunes itself to improve performance by trial-and-error adjustment of numerical parameters",

and three,"inductive generalisation,whereby new descriptions are automatically generated and extended to fit the growing store of experience."Thus the result of the use of the branch swopping mechanism (1.121) could be classed as parameter learning.Now we come to the question of our promotion mechanism (1.122),a mechanism which first became apparent as a natural extension of the branch swopping mechanism,but later we came to view it as a generalisation of MICHIE's (1967,1968) "memo" functions.

The memo function idea is that a programmer should be able to attach to a given routine a small private memory,or "rote",to which the results of successive calls to the routine are automatically written.When this "memo-ised" routine is referenced,immediate evaluation is first attempted by look-up from the rote.If this fails,the body of the routine is entered and evaluation proceeds by calculation.In the latter case a new entry is made in the rote on exit,the value of this new entry being,of course,that of the particular result just calculated."Methods are incorporated for automatically 'forgetting' and 'refreshing' the memorised argument-result pairs according to the frequency of occurrence of the arguments".(MICHIE 1970a).The subtrees within our total tree hierarchy are subroutines (1.122) - generalisations of similar syntactic features.The promotion mechanism selects the very infrequently occurring items,if any, from each subtree and places them within the appropriate referencing trees (or subtrees) in a position of slightly higher precedence than the particular referencing node. Thus when matching an incoming statement the contents of this promoted 'rote' are examined first and if the current input feature is not found then the particular subtree is entered via the subtree referencing node and is then searched. For example,consider the FORTRAN input statement illustrated in 1.122,figure IV.The rote consists of the single item '5' and the generalised function is the subtree 'R',which is referenced via the node illustrated as '£R'.The contents of these promoted rotes are automatically adjusted according to the dictates of the frequency counts.We deviate slightly from Michie's above description of memo functions in that rote contents are altered periodically rather than continually, but the difference is clearly a minor point of implementation

rather than logical structure.

The justification for our contention that we have a generalisation of the memo function is based upon the fact that the features which we promote into the rote can be immediate symbols (such as '5' in 1.122,fig.IV),or subroutine references which may reference further subroutines to an arbitrary depth,or a combination of both.This powerful feature thus encompasses MICHIE's (1970a) proposed extension of the memo function concept,which is for the rote items to represent sets rather than individual items.

A further learning program which we will discuss,"Pandemonium:a paradigm for learning"(SELFRIDGE 1959,1970),was first published in 1959 and reprinted as part of a collection entitled,"Perceptual learning and adaptation".The interest from our point of view is that an attempt is made to process a natural information source - Morse code sent by a human - but not as a conceptual transcriber,that is, not stipulating that dash=3 x dot.The system is organised in a series of hierarchical levels in which each level performs its processing and passes the results up to the next level which processes more generally,thus one level might receive information from the level below which it processes and outputs,say,a dot or a dash (or the relative probabilities) which is received by a unit in the next higher level along with dots and dashes from other lower units:this higher level unit then outputs a character,etc.The type of information used and the refusal to incorporate the conceptual formulation of its structure(i.e. dash=3xdot) but rather to try and "transcribe actual code the way people use it",is somewhat akin to our approach.Unfortunately the promise of results (1959) has failed,to our knowledge,to materialise (but note extension

of this project McELWAIN 1962,discussed in 2.21),this can perhaps to some extent be accounted for by an apparently small but possibly crucial difference in our basic design principles.That is,although we build up from the bottom of our hierarchical structure we are at any point prepared to 'guess' on a higher level and proceed with the consequences of the 'guess' unless an inconsistency occurs,we accept that this procedure may give rise to errors which the formal bottom-to-top procedure would not.But,as stated by MILLER (1962) and NORMAN (1969) there is no logical reason why the formal (that is,no 'guessing') procedure should be able to process natural information at all,since all available evidence points to the conclusion that humans could not possibly be utilising this formal procedure and thus we cannot be sure that sufficient information is present.

2.12 Approximate and adaptive pattern recognition

The main applications of approximate and adaptive pattern recognition techniques have been in the field of natural information processing,for example,the identification of hand written characters.The incentive for this work is to be found in the potential saving in terms of human time and effort and the propect of the many computer projects that would become viable if the vast amount of time that has to be spent producing the high quality data that computers cu customarily demand,could be reduced.

The early work on approximate pattern recognition used the 'template' method in which the matching was in fact exact but the specific variations of each pattern were stored as different templates.The obvious problems with this method,such as which variations are most useful,etc., are discussed by SELFRIDGE & NEISSER (1960),where they

suggest the next improvement should be the construction of a system which can automatically select and alter the pattern features for which it is searching,as a result of experience.They cite Selfridge's program for automatic Morse decoding as an attempt to automatically generate its own features (a program already discussed above 2.11).They also stress the need for the recogniser to utilise the 'context' of the pattern;a point which we will take up later in this section.

The usual method of 'learning' to recognise involves the optimisation of a set of parameters by 'training' the system with specially selected sets of input data,but the intiial features (essentially the parameters) have to be selected beforehand by the human operator,see SLAGLE (1971a), UTTLEY (1970) and MUNSON (1968) for recent examples of this approach.

The last author,Munson,also gives a survey of previous work on automatic recognition of machine-printed and hand-printed characters,and concludes that the greatest problem area is "the generation and selection of features for pattern classification".

CARBONELL (1970) describes an application of artificial intelligence techniques to the design of a computer-assisted instruction system called "SCHOLAR".The system aims at evaluating student's answers as not only correct and incorrect, but also approximately correct,and there are plans for the system to be able to accommodate many types of student error. But the approximate pattern matching technique seems,so far, to extend only to approximate numerical answers which can be dealt with quite simply by the use of some numerical margin of error.Again there is no explicit description of

how and what errors can be corrected,and the only example is of one simple spelling correction.

Concluding our discussion of the above types of pattern recognition we will consider a recent paper from a hitherto unmentioned but important area of approximate and adaptive pattern recognition;that is,automatic speech recognition. The automatic word recognition system of CLAPPER (1971) uses an adaptive electronic template to recognise discrete words.On trials with 16 different words the system had an overall accuracy of 94.2% for 13 individual speakers who had,of course,supplied preliminary training data.The most interesting point from our present position,is in the author's concluding remarks,he states that this method will not be good enough for more comprehensive and practically useful systems.

As an alternative to the above techniques which can be loosely grouped as the optimisation of mathematical expressions is the possibility of employing a more or less ad hoc collection of intuitive heuristic techniques;the possible advantages of which have recently been expounded by IVAKHNENKO (1970).He suggests that many problems such as the automatic synchronous translation from one language to another,are impossible to solve by deterministic methods. Thus although heuristics are mathematically groundless they can give us results that are good enough in practice and are often much better than those which can be obtained from a formalised approach.This approach is largely in accord with our own (see chapter 3).Ivakhnenko terms his method "heuristic self-organisation" which is described as the generation of random hypotheses.He demonstrates the "prediction of random processes" in which "the accuracy is

quite fantastic".Although he specifically opposes his method to the formal mathematical type of approach and says,"for the present we cannot give an exact mathematical definition of 'self-organisation' ",it appears to us to be a collection of specialised mathematical techniques which are complex in relation to those used in our adaptive analyser.

MINSKY (1970) discusses the work and problems involved in formally demonstrating the equivalence of algorithms and schemata,he then cites SLAGLE (1963) as representative of the practical rather than formal approach.Slagle's system for symbolic integration is composed of a set of heuristic pattern recognition techniques built on top of a set of formal transformations,and this is a structure and approach that is simialr to both Ivakhnenko's and our system.

A further survey of pattern recognition or as the authors term it,"picture processing",is particularly relevant to us as it is a survey of the "linguistic methods" used.They (MILLER W.F. & SHAW 1968) first discuss the receptor/categorizer model (RCM),essentially the model we described earlier but viewed as a receptor which extracts the features from the picture or pattern;then a categorizer which,on the basis of these features,classifies the picture as one of a finite number of classes.Within the field that our analyser is currently working in,that is the parsing of computer languages,the above model is invariably the one used.The finite number of classes on the first level being two,'correct' and 'incorrect'.Our analyser parses incoming statements without this initial categorisation.The term "linguistic method" is used to describe methods in which the processing is directed by the structure that is used to describe the

picture;thus we consider our system to fall within this grouping,the experience tree being the description of the picture and the strategy codes (4.111) within it direct the processing.The authors state that the RCM model fails to be useful for analysing complex patterns "where the structure and interrelationships among the picture components are the important factors."This is a point that is also stressed by FORSTER (1970),who states that in the study of human beings (and we might add,in the processing of natural information sources),"the conceptualisation of a variable as the weighted sum of other variables is methodologically unhealthy ... it is precisely the complex interaction effects among variables which are of the most theoretical interest". He introduces a system of man-machine interaction for the discovery of what he calls "high level patterns",which are the partitions "of a set of individuals into subsets within which a common set of linear,additive,non-interactive relationships holds".Similarly our experience tree (1.01) maintains the complex interrelationships between the functionally simple entities,the frequency counts.

In his paper "an introduction to linguistic pattern recognition",ANDERSON (1970) draws attention to the fact that a programming language compiler is a pattern recognition device.And that programming language compilers have been developed to produce very efficient systems,at least when the idea of error correction is not entertained in any general sense.He makes the point that such linguistic analysis produces a structural description of the picture (in our case statement) analysed and not just a classification into one of n categories.This is precisely the reason we can automatically 'translate' any analysed statement.

Anderson describes the extension by means of

the generalisation of the linguistic methods to the recognition of two-dimensional pictures,that is," a two-dimensional compiler".He then surveys the work in linguistic picture analysis applications.The work most relevant to our system would seem to be that of SHAW (1970) who like Kuno (see 2.02) uses a system which recognises patterns "in a predictive way".But all that "predictive" means is that once a feature has been recognised the grammar is used to generate the set of valid successive features and the system then proceeds to attempt to identify the actual feature as one of this subset, rather than attempt to classify it amongst the complete set of all possible features.Our system represents an advance on this in that not only do we 'generate' the subset of valid successive features but also rank and examine this subset in order of their relative likelihoods.Anderson also states that "there have been relatively few applications of linguistic picture analysis to 'real-world' problems",and points to omissions within the papers he surveys of how 'noise' problems are dealt with.A fundamental feature of our system is that we place no a priori limitation upon the degree of 'noise' that can be tolerated in the 'pictures' to be processed (1.2).Finally he cites WATERMAN (1970) as an example of the investigation of a fundamental type of flexibility that would be exploited in the syntax-directed approach to pattern recognition:adaptive behaviour.

WATERMAN (1970) presents a system in which heuristics are represented as production rules,which have the advantage of separating individual heuristics from the program,clarifying their interrelationships and being compatible with generalisation schemes.The system improves its performance by modifying existing heuristics and hypothesizing new ones.

He mentions "the credit assignment problem" which hinges upon how credit should be distributed amongst a number of steps that all contributed to the successful completion of some task.We have come up against the reverse of this problem in attempting to produce to highest likelihood analysis of incoming statements.Waterman's techniques (as Anderson says) lead to the possibility of the analyser handling special cases as these cases arise,rather than attempting to anticipate all patterns of interest initially. A feature which is paralleled in the particularisation ability of our system (see the promotion mechanism 1.12). A final interesting feature from our point of view is the need that Waterman finds to guard against over-generalisation and to maintain redundant rules so that they may lead to non-redundant generalisations.

The further extension of our system in approximate pattern recognition of natural information sources concerns the inclusion of non-exhaustive matching (1.14);we are not aware of any relevant previous work,that is,work that is pertinent to our particular problem (except possibly BLOOM 1970,whose work is discussed in chapter 3).

The final aspect of approximate pattern recognition that needs to be developed and exploited is the use of context, that is the structure within which the pattern is found and can be anything from the character that,say,precedes the character that we are currently considering,to a complete and up to date knowledge of the world.Clearly there is a wide margin here within which contextual considerations can be used.This is by no means a new idea but one which, to our knowledge,has only been tackled on a very limited or ad hoc basis.Clearly in the parsing of a formal language

when it is only necessary,and hitherto considered sufficient, to categorise input as correct and incorrect there is no need to consider context (except of course the very limited formal sort imposed by for instance,'type' statements).But any attempt to extend the problem into the realms of approximate recognition would benefit greatly from the introduction of broader contextual considerations.Current formal conceptual systems do not yet seem to cope with context above statement level and consider only 'context free grammars'.At what may be conveniently considered the next stage of generalisation,program context,the practical compilers at Cornell University are possibly the best example (CONWAY 1963,1971 discussed in 2.2),and appear to be a collection of special techniques that the system designers implemented,rather than the result of any general theory.At the next higher level the context implies a knowledge of the real world which in turn implies artificial intelligence.

Our analyser currently only works with the first level of context,partly because that presents us with sufficient problems for the moment and partly because we wish to resist the addition of a collection of very specialised techniques that higher level contextual processing would necessarily involve.But the basic system is in no way limited to this first level of context and in any practical application we envisage the inclusion of specialised techniques (for example,basic program logic) although as a research tool we feel it is currently more useful in its present uncluttered form.

2.2 Error recovery.

The most striking result of our review of the litrature

on error recovery is that there are very few attempts to 'learn' about the errors corrected with a view to automatic improvement or even manual alteration of the system to effect the improvement.Typically severe limitations are placed upon the types of errors that can be corrected.Our analyser embodies neither of these restrictions.

2.21 Error correcting algorithms.

The methods of error correction can be divided into three groups.The three methods are;a 'matching function' and a stored pattern,a dictionary of stored errors,and a fixed set of local manipulations.We will discuss the previous work in roughly chronological order within each of these three groupings.

To begin,we will consider the first of the above described groups.The earliest paper is that of GLANTZ (1957),a "matching factor" is computed from the number of matched characters in equivalent positions in two strings,divided by the length of the longer string.

BLAIR (1960) presents a "Program for Correcting Spelling Errors",he uses abbreviations which retain the "meaningful kernal of the word" and the criterion of similarity is that the abbreviations of the two words are identical.The scheme for constructing abbreviations is to assign to each letter of the word a value which reflects its importance (this value is arrived at by consideration of the letter itself, its position in the word,and frequency of occurrence of errors) then letters are deleted until the four most important letters remain.Blair also mentions an improvement procedure with failures,which is to add the abbreviation of the uncorrected word to the dictionary,as a new word or tagged to the standard abbreviation for that word.

DAVIDSON (1962) describes a further abbreviation technique which he uses to recognise and retrieve the few names that most closely resemble the given misspelt name in the context of an airline reservation inventory.He forms the abbreviations by deleting all vowels,the letters H,W and Y (unless it is the first letter),and elimination of all but one instance in any contiguous repetition of a letter.If no exact match is found the program returns a list of rated possibilities for final selection by a human operator.

THORELLI (1962) discusses in a general manner various systems for automatic error correction.The particular problems that he suggests formal methods for correcting are not realistic,for instance,the first examples assume that the 'word' structure is preserved.He cites the then recent work of Blair (discussed above) and that Blair's abbreviation technique is most suitable for the elimination of errors induced by the 'human factor'.When the problem is extended and the word structure constraint is removed (i.e. most real cases) then the mathematical techniques give rise to a combinatorial explosion of the number of necessary applications of the 'matching function'.There is an interesting conclusion in which he suggests that it may be better to allow the machine to process incorrect text rather than correct it first as although certain passages could be misinterpreted this "is a phenomenon which could not be fully avoided anyway".

McELWAIN & EVANS (1962) investigated the possibility of improving the output from Selfridge's Morse translator (SELFRIDGE 1959,discussed in 2.11) by using certain known linguistic contraints.They used a dictionary of digrams and trigrams and the technique involves a mathematical scoring

function to estimate the 'match'.Only a very limited set of error types are corrected but the authors claim the program's output is genuinely much more intelligible and useful than its input."In other words,for errors caused by noisy communications circuits,no enormous linguistic or semantic analysis is needed to provide a sensible degree of correction".

ALBERGA (1967) gives a review of the litrature of "string similarity and misspellings" and evaluates a number of algorithms by testing on a collection of misspellings"made by students at various grade levels.The methods evaluated are characterised by their mathematical rather than their intuitive or heuristic content and thus will not be discussed further.

The last of the abbreviation techniques described is that of JACKSON (1967),a series of precise rules for forming the stored 'perfect' abbreviations was derived from the designer's insight and experiment (no indication is given of the relative proportions);the actual rules are given.Matching is on a letter-by-letter scan and comparison basis and a 'matching factor'.This matching factor is calculated from; one,the number of letters that match,two,the number of unmatched vowels in the input,and three,the number of unmatched characters in the 'perfect' abbreviation.He states that six letter abbreviations gave almost perfect results and the most important point is that the system is used satisfactorily in a commercially orientated system.

CORNEW (1968) uses a dictionary of letter digrams and their relative frequencies in English,he uses a large language dictionary and failure to match the input to an entry signals an error.The frequencies of the digrams occurring in the incorrect word are examined and the least likely letter is replaced by the most likely alternative.

He tests a wide range of words,the 1000 most common English words,but only one type of error,a single randomly substituted character.He obtained 94% corrections and concludes that more complex errors would involve a combinatorial explosion of possibilities to be tested and thus "the spelling correction problem might at this point be viewed as giving way to syntactic and semantic considerations".

There are two publications in which the error correction is performed by identifying the incorrect item in a dictionary of stored errors and then obtaining the associated correction.

GALLI & YAMANDA (1967) describe "An automatic dictionary and the verification of machine-readable text",which is part of a broader study dealing with the conversion of documents into machine-readable form.The problem was attacked by constructing a dictionary that contains enough words to keep the number of errors due to absence from the dictionary small,plus standard variations and common errors.To minimise the size of the dictionary entries were automatically compounded from generally four fields which have functions such as,after a match modifying the search for the next entry.The resulting dictionary could cope with approximately 56,000 words and included 2,000 entries for standard variants and 2,000 entries for error correction.The entries for error correction were obtained from actual document file spelling errors,errors made by secretaries in typing and five other contributing sources are referenced.The main test of the experimental dictionary involved 130 documents containing about 19,000 words;of all words processed,89% were properly verified,of the remaining 11%,6% were not in the dictionary and the non-corrected errors were some part of the final 5%.

The idea of adding common errors to the dictionary is one that we have toyed with but not implemented because we feel

that the dictionary should be automatic in the sense that the error entries should be added to the dictionary as they occur and then be deleted if they are not common;this is a complex problem when dealing with a dictionary structure of trees and subtrees nested to an arbitrary depth thus we have temporarily abandoned the idea in favour of more important ones.Galli particularly mentions the missing-space type of error as occurring fairly frequently in the key-punched documents,e.g. 'keptat' and 'threecomponent'.This is an example of the missing boundary or 'fuzzy edges' problem.

DARDEN (1969) describes work 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;he considers particularly,formal languages.The input is an ordered list of alternatives (an L-list) in which each alternative has an associated confidence which is proportional to the logarithm of the probability that the particular character is,in fact,the correct character.Substrings are delimited by spaces (which are treated as error free characters),in the matching process a substring of length n is matched,using a 'distance' measure against dictionary entries length n,n-1,... up to n-k,until a satisfactory value of the distance measure is found.He points out that with k=2 no erroneous matches in the processing of LISP, ALGOL or FORTRAN were found.His approach to the problem of collecting poor quality input is fairly close to ours in that he is particularly interested in exploiting the redundancy of a language and he also utilises presumably intuitive ideas of how a human analyser would tackle the problem of garbled text,in order to formulate his method

of attack.One such idea that he implements,and with which we are in complete agreement although we have not initially utilised it,is to give the parser a global view of the structure before the detailed processing.To return to redundancy and the exploitation of,Darden distinguishes three types;"vocabulary redundancy,results from the grammatical specification of a set of strings (basic symbols) which have a high a priori probability of occurrence",two, "statistical redundancy results from the high probability of multiple occurrence of members of the name-space of such languages",and three,"structural redundancy results from the grammatical constraints on concatenation of well-formed strings".We exploit these three categories of redundancy in the following ways.Redundancy of the first category is exploited by virtue of the fact that the basic symbols, keywords,are placed specifically in the main tree at the highest level of our experience tree (1.02).We exploit statistical redundancy in Darden's sense by the use of our promotion mechanism (1.122) which elevates specific names as a result of their multiple occurrence,to a position of higher precedence than all the possible (by definition) instances of that name.Finally the confidence jump mechanism (1.14) was designed to exploit the structural redundancy which becomes immediately apparent from the tree structure definition that we employ.A structurally redundant symbol will occupy a tree node that contains no link-left address (APPENDIX III) or to put it another way,each structurally redundant symbol occupies a node which is the only member of its filial set.

The earliest paper that falls into the group that we have loosely circumscribed as, correcting errors using a fixed set of special manipulation,is that of CONWAY & MAXWELL(1963) in

which they describe CORC and its implemention.Conway's 1963 paper,CORC is a language designed at Cornell with special consideration given to increasing the chances of succesfully processing an incorrect program.For example, extra redundancy was added to the assignment statement when usually the emphasis is on eliminating redundancy to save computer time.One of the (if not the) highest priorities with the CORC compiler was that processed programs should always execute in the hope that the programmer can gain further information about the correctness of his program.The error correcting ability of the CORC compiler rests upon a collection of fixed procedures, such as,"add + for a missing operator except when two adjacent operands then add *",which were presumably based upon some prior survey but which make no provision for possible improvement.The latest of the tolerant compilers from Cornell, "A diagnostic compiler for PL/1" called PL/C (CONWAY & WILCOX 1971a),has "local repair" facilities described as "variations upon four basic tactics" which are;one"delete the next symbol", two,"insert a synthetic symbol",three,"replace the next symbol", and four,"delete the previous symbol".The compiler does not have a systematic algorithm to specify which repair tactic is to be employed under particular circumstances,generally implementers of a particular module judged what would constitute the most plausible repair.

Once again proceeding chronologically after Conway's CORC compiler came IRONS (1963) with "An error correcting parse algorithm" based upon the justification,"It is common knowledge that most of the ALGOL or FORTRAN 'programs' which a compiler sees are syntactically incorrect".The algorithm will parse any string describable in BNF (BACKUS 1959),when more than one parse is possible he continues all possible parses in parallel.An error is detected when no

parse can be continued further and then he makes the somewhat ambiguous statement that the error is at the point where the parsing failed or shortly before.Errors are corrected by making local insertions,substitutions or deletions until the string can be parsed again.There is no attempt at a 'best' correction in any sense and no mention of improvement.

DAMARAU (1964) uses only four error correcting techniques which he says account for over 80% of mistakes,the four errors are;one,one letter missing,two,one letter inserted, three,one letter wrong,and four,two letters transposed.On a trial with nearly a thousand words containing spelling mistakes Damarau's technique corrected 96% of those that fell into the above classes whilst a test run concurrently using Blair's abbreviation technique (described above) gave poor results.Only 15% of the total errors fell outside the above four classes and the distribution within the four classes is given.

As a result of a translation project at the National Physical Laboratory from 1960 to 1967 in which the difficulty of processing large quatities of data containing,inevitably, trivial mistakes became apparent,SZANSER (1968,1971) has developed a novel method of error correction to be applied to natural language text.To obviate the difficulties caused by trivial errors,say when the initial letter of a word is missing the remaining letters are all misplaced relative to the correct word,Szanser assigns specific characters to fixed positions in a pre-determined sequence.The letters of a word are then spread over these positions always keeping the order unchanged.The text is then divided into segments called "lines",a new line is begun when the current character occurs earlier in this set sequence than the last character. The resulting lines are then matched against the lines corresponding to the correct words.As an example,the fixed

sequence 'azbf' segments 'aazabazfbbfza' into the lines 'a', 'az','ab','azf','b','bf','z' and 'a'.Then the input lines are "elastically" matched against the dictionary lines checking only for the four classes of error described by Damarau above.Examples of the corrections performed are given but no indication as to frequencies of errors or types of errors.

MORGAN (1970) describes several specialised techniques for efficiently incorporating spelling correction algorithms into compilers and operating systems.Once more only the four special classes of error are considered (see Damarau above). Keywords are corredted during the syntactic pass of a translator and suspiciously used variables during the semantic phase.The correction is performed by a pairwise comparison of the incorrect string against all dictionary entries differing at most by one in length from the incorrect string. This is again a severely limited mathematical approach which does not bear much relation to our method.

In the work on syntax-directed error recovery described by LAFRANCE (1970,1971) the detection of a syntactic error causes all possible parsing paths at the point at which the error occurred to be examined again and the path chosen is controlled by a set of intuitive and predetermined rules. The major difference with this work and our system is that the order of probable error occurrences is built into the compiler as opposed to being dynamically determined on the basis of frequency of use during previous processing.Working with a small subset of ALGOL-60 the system processed six programs written by novice programmers and one with deliberate errors,49 out of the 55 actual syntactic errors were corrected "exactly as the programmer would have done". A summary of the errors corredted indicates that just over

80% involved the omission of syntactic features,and 40% involved a syntactic feature rather than a single character.

In LEVY's (1971) thesis,"Automatic correction of syntax errors in programming languages",we find an explicit distinction between "error recovery"- the process of determining how to continue analyzing a source program when an error is detected",and "error correction - the process which,given an incorrect program,transforms it into a correct one".Thus previously we have mostly been discussing "error recovery" not "error correction".We do anticipate some difficulty in deciding whether we have "corrected" an error or "recovered" it,for we cannot accept Levy's criterion of a "correct" program as "what the programmer meant".Levy criticised previous work on the grounds that it is heuristic and most often is only "error recovery" and states that,"It (his) is the first model,which is both formal and fairly realistic ..".The work is mainly concerned with finding the set of possible interpretations of an incorrect string and then in the conclusion refers to chapter VI as supplying "a conceptual framework for developing,in a systematic and orderly way,heuristic assumptions and restrictions which lead to,what he terms "good" error corrections.The "systematic and orderly" framework is described in the chapter heading as "miscellaneous practical examples" and is just that plus the statement that "some learning process" must be used.We feel that this falls short of "formal correction" and that the criticism of previous work as "heuristic recovery" is perhaps a little out of place.

2.30 Systems with similar aims.

ALPIAR(1971) describes "double syntax oriented processing" in which he has two blocks of production rules,one to parse the input and the second to construct the output from the parsed information,clearly this has similarities with our analyser where the syntax tree parses the input and the stored pattern constructs the output from the parsed input.He envisages similar uses such as;the modification of programs to conform to a 'local dialect' of the original programming language;the execution of programs in a temporarily altered form,either as a debugging aid,or to test special features; and the translation of programs into machine code,that is, compiling.He stresses the symmetry of his system and thus input can be recovered from output by simply exchanging the two blocks of production rules,this is not a feature that our system exhibits and we feel that as soon as the system is extended to "ramified" rather than just "simple" processing -the author distinguishes the former as processing which allows context sensitive features to be handled- thedegree of symmetry than can be exploited in this manner will be considerably reduced.The author makes no mention of the possibilities of improvement,error correction and no concrete practical results.

Alpiar does cite the "compiler-compiler" of BROOKER,MacCALLUM, MORRIS & ROHL(1963) a project that we feel is relevant only in the very general sense that our analyser will process any language which can be specified in a form suitable for the tree building process,while Brooker's system will record the definition of any phrase structure language and then translate any program written in the language.

HEINDEL & ROBERTO (1972) are currently developing a retrieval process language for hierarchical data bases

similar to the "LANG-PAK"(HEINDEL & ROBERTO 1972a) language which they hope to make as self-correcting as possible.LANG-PAK which is "an interactive incremental compiler-compiler" is written in ANSI FORTRAN for machine independence reasons the same as ours,and the system allows the designer to alter the syntactic and semantic specifications of the language in an interactive mode.

3.0 Basic design principles.

In this chapter we describe the principles which have guided the development of our analyser.Initially we consider the principles behind the exploitation of the pattern of use (3.1),and then discuss the principle of error recovery (3.2).Finally we describe the application of these principles in four different areas (3.3).

3.10 Exploitation of a stable pattern of language use.

The main objective can be summarized as:how can we exploit knowledge of a pattern of use in the analysis of a programming language?We are searching for some sort of stability in the usage of computer languages which can then be exploited by the use of heuristic methods of analysis.

The exploitation of this purely empirical stability in the use of certain language features can be contrasted with the majority of current theoretical investigations which nearly always assume a random distribution of statements and more detailed syntactic features;or at most, utilise a rough a priori distribution based largely upon intuition rather than detailed measurement.

The basic approach is to improve the processing of frequently occurring features at the cost of dealing inefficiently with unusual events;which can be contrasted with ensuring that a moderately satisfactory solution is provided for every possible contingency.

3.11 Frequency stability of the language feature usage.

The stability of the usage pattern is investigated by the collection of usage frequencies within a spatially well-defined structure - the experience tree (1.02).The

sort of stability expected is that which occurs within natural languages as described by HOCKETT (1968),"some features are replaced very rapidly;others persist for centuries or millenia"(perhaps months or years would be more accurate in our context).For example,the keywords and basic statement structure of a language will probably persist for the lifetime of the particular language,but small changes and additions may be introduced from time to time by systems programmers,which are either adopted or ignored by the majority of users.

Each possible language feature is represented by an item at the appropriate place(s) within the experience tree.Each item has an associated frequency count which is incremented by one each time the particular tree item is successfully matched against an input feature.This procedure was chosen as the simplest in the absence of any evidence to justify a more complex measure,but is,of course,by no means the only method (see,for instance the more elaborate method of NOTLEY 1970).

We have chosen to use measures which are crude by sophisticated mathematical standards in the sense that the manipulation of the tree structures is purely a result of the simple comparison of the frequency counts.

3.12 Mechanisms for restructuring the experience tree.

Simplification and hence optimisation of the analsis of incoming statements is based largely upon the restructuring of the experience tree.The restructuring mechanisms are in turn based upon two principles which exploit the usage pattern.

First,precedence is given to the most commonly occurring items within any set of alternatives within the experience tree.This ranking of alternatives is accomplished by the

branch swopping mechanism (1.121).

Second,any unwanted generalisation is removed by giving precedence to the high likelihood particular instances of a generalised feature,over the feature itself.This removal of unwanted generalisation is accomplished by means of the promotion mechanism (1.122).

The basic principle that underlies both of these mechanisms is that within our particular,limited enviroment,the past closely resembles the future - or,that there is a general stability in the pattern of language usage.

We recognise and accept that this principle which amounts to the prediction of future events,is unprovable and logically unsupportable and that the only justification for it can be the practical utility that it may exhibit.

We use the word 'likelihood' to convey the inference from past experience to the prediction of future events rather than the more obvious word 'probability' which entails a number of conclusions derived from the various'theories of probability' which are certainly debatable (COHEN & CHRISTENSEN 1970,or POPPER 1968,for example) and which we do not require.

3.13 Further exploitation of usage information.

The frequency counts are utilised further as the confidence levels (1.13) within the experience tree that are used in the control of the matching of input features.

In the sense that any means of comparison of two structures involves a 'distance' function,then we employ such a function. But there is a distinction that might usefully be drawn between the type of distance function which utilises fairly complex mathematical manipulation to condense the structure into a single number,and the type which concentrates on preserving the interrelationships between the entities

that comprise the total structure.We utilise this latter type which involves a complex of simple comparisons rather than a simple comparison after a complex manipulation of the structure.Examples of the former type can be found within the pattern matching techniques collected by FU (1968).

3.14 Non-exhaustive analysis - the principle of allowable error.

The feature that we have described (1.14) and have termed 'allowable error' is another facet of our aim to give precedence to high likelihood features.The use of allowable error is a necessary function of economy of time and/or space for systems so large and complex that they would not function adequately whilst processing with zero allowable error.It can be viewed as a substantial gain in the overall performance of a system in exchange for a small proportion of mistakes.

BLOOM (1970) in his paper "Space/time tradeoffs in hash coding with allowable errors"(and from where incidently we originally obtained the term 'allowable error') advances a hash coding system in which the size of the hashing space can be reduced in exchange for a proportion of allowable error.He suggests that a secondary perhaps time consuming test might be used to 'catch' the small fraction of errors associated with the new methods.Bloom sees his system being used when a large amount of data is being used and the hash area using conventional methods would be too large to be feasible.

RANDELL (1971) implies that allowable error techniques might be necessary in large and complex systems.The paper is a report prepared for the SRC on the reliability of complex computing systems.Randell says,"A result that is 'guaranteed' correct,produced after all need for it has

passed,may well be less valuable than a timely result which has some (hopefully small,and known) probability of being incorrect!".

An example of a possible use of allowable error within our analyser might be:when frequency counts indicate that certain features of a language are not being used,the processing system could,as a first approximation,delete these non-used features from the language.Thus processing speed would rise due to the now more compact definition of the language,but at the cost of occasionally dealing inefficiently with a program using one of the deleted features. Allowable error techniques can be particularly useful when the information to be processed is less than 100% accurate but the 'correct' form can be recovered because of redundancy in the specification.

3.20 Tolerance to inaccuracy of specification.

In this section we consider the possible applications of tolerant processing and meet the objections that it commonly evokes.

Our system is tolerant in that redundant features of the language being processed need not necessarily be included, and errors of specification can pass 'unnoticed' if trivial, or else be 'corrected' by the utilisation of any redundancy that the programmer has included,in conjunction with 'knowledge' gained from past experience.This approach is open to three main types of criticism:one,the complexity of such a system and its consequent slowness,two,that it encourages 'sloppiness' in programmers,and three,that users will become dependent upon a degree and type of assistence that they will not receive from other systems.

We consider that the increase in complexity and the concomitant decrease in speed is inevitable with any move to construct computer systems which are designed to actively assist the programmer rather than to display the customary indifference to inexperienced users.MINSKY's (1970) Turing Lecture was based upon the statement that,"An excessive preoccupation with formalism is impeding the development of computer science".He sees an increase in the complexity of computer systems as inevitable and maintains that the function of a compiler is currently considered to be "to translate from one language to another",and should become "to produce an algorithm,given a description of its effect". Further,Minsky states that,"A heuristic compiler system will eventually need more general knowledge and common sense ..." Now,although our system by no means meets these suggestions it is nevertheless a step in this direction,albeit a small one.

A system designed and implemented in accord with Minsky's suggestions,called "the programmer's assistant" is described by TEITELMAN (1972).The basic idea behind this system is to enable the programmer to say to the computer "do what I mean" instead of "do what I say".This system is tolerant to errors and generally attempts to provide a cooperative and helpful enviroment for the programmer.In answer to the suggestion that the system would promote sloppy working habits,Teitelman says that freeing the user from many of the more trivial tasks "results in his being able to pay more attention to the conceptual difficulties of the problem he is trying to solve".

The tolerant computer systems from Cornell University (the CORC compiler CONWAY & MAXWELL 1963 and the PL/C compiler CONWAY & WILCOX 1971a) have been functioning

satisfactorily in an ever increasing number of installations over the last eight years.Thus the basic philosophy of these compilers that every program submitted should execute,which would be regarded as very rash in some quarters,has been soundly endorsed in practice.This philosophy is based upon the supposition that "by briefly prolonging the life of an obviously moribund program it often yields additional diagnostic information which helps to reduce the number of submissions required to achieve satisfactory execution". (CONWAY et al. 1971,p.35).This approach meets with the three types of criticism described above:the reply (CONWAY & WILCOX 1971a),suggests that the increase in compilation time is offset by the reduction in the number of debugging runs needed,the dependence charge is as much a criticism of other systems for not providing sufficient help to the programmer and finally,eight years of satisfactory use of these systems at Cornell and elsewhere is a testimony to its practicability. University College,London is currently evaluating the PL/C compiler (see Computer Centre NEWSLETTER no.65,1st May 1972) and after preliminary tests consider that the basic philosophy (i.e. forcing execution) is sound and the only drawback is concerned with the subset of PL/1 that it deals with (private communication 9 June 1972).

We will preface our discussion of the 'sloppiness' charge with some evidence to support our claim the current computer systems are not all that they could be.Three recent papers (BARRON 1971,EVERSHED & RIPPON 1971 and WALWYN 1971) give numerous examples which suggest that many computer systems can be too rigorously pedantic,and consequently some relaxation of the constraints that are typically imposed upon the programmer might be defensible.

EVERSHED & RIPPON (1971) suggest that compiler efficiency is too sacred and that some could be sacrificed to reduce human errors.Then after stating that changes to humanise computer systems could not be expected unless there were strong economic reasons for doing so,they continue and adduce some possible reasons.They expose some inefficiences that go unrecognised,such as,one compilation error causes a doubling of computer time;and then they focus attention onto lost opportunities such as, the number of 'non-professional' potential programmers who are frightened off by the inordinately time consuming and frustrating task of producing a workable program.

We note in passing that "computing costs per basic operation in the last twenty years,have halved about every eighteen months"(GOOD 1969,p.1),a trend which should lessen the pressure upon the hitherto overriding need to save computer time.

Professor BARRON (1971) concludes his list of frustrated attempts to communicate with a selection of computers with, "... why should computer users be expected to put up with such systems?Either the designers have never actually used a computer as a problem-solving tool,or they have a vested interest in maintaining the mystique of programming.Neither situation should be tolerated in a responsible professional community".

The charge that tolerant computer systems encourage 'sloppy' programming is usually extended to point out that 'sloppiness' is not conducive to good scientific work and therefore should not be countenanced within a scientific enviroment.Whilst we admit that there could well be some merit to this viewpoint;we believe that in view of the rigorously pedantic philosophy which seems to underlie many current computer

systems,the degree of relaxation which we advocate hardly justifies the description 'sloppy' and its concomitant undesirable connotations.We emphasise that our position is not to let programmers 'run riot' with the system.A well designed tolerant system will naturally provide incentives to encourage precise programming in that 'correct',that is, expected program statements will be processed faster and the inclusion of any redundancy will increase the chances of having errors 'correctly' recovered.

A further and less controversial use of a computer system that is tolerant to errors in the specification of its input data,is as a preprocessor which will accept and extract the essential features from large quantities of data which inevitably contain many trivial mistakes.An example of this type of usage is found in SZANSER (1971) who states that the Machine Translation project of the National Physical Laboratory constantly ran up against the problem of coping with errors in the input,"even if the error was trivial by human standards (such as to pass unnoticed - obvious evidence of the existence of 'automatic error correction' in the brain)".Clearly this type of correction is one of the basic features of our system (the confidence jump mechanism,1.21) which although its merits are questionable in the current form of implementation (we have chosen to implement the confidence jump mechanism such that trivially incorrect input is processed just as fast as completely correct input but this entails an inability to signal modifications as they are 'not noticed');it is perhaps a case of coming to terms with features such as imprecision of specification and the resulting action or imposing a severe block upon the type of problem that can be solved.

3.30 Areas of application.

It will become clear that the analyser as currently implemented can be locally optimised at many places.We have based the system upon a few simple techniques and largely resisted the temptation to incorporate many obvious features which would produce an immediate local optimisation in time and space for execution.This is because we are still at the research stage and thus wish to maintain a general and uncluttered system so as to more easily evaluate the few techniques involved.The next stage would be to particularise and thus optimise the system in any direction that the previous evaluation stage may have indicated as potentially fruitful.

3.31 Language processing in computer systems.

As stated above we have made no attempt to optimise the analysis system at a detailed level to make it suitable for immediate practical use.A similar situation arises in an information retrieval application where MOYNE (1969) discusses his natural language information retrieval system in comparison with the much faster 'keyword' systems.The crux of the comparison is that conventional systems leave little room for improvement or extension,whereas the more general and dynamic system is amenable to,and can even indicate numerous avenues of improvement and extension.For example,we consider (chapter 6) the extension of the hierarchical structure that underlies our system down to the level of character recognition and up to the level of program structure.The important point is that these enlargements can be accomodated as natural extensions of the original structure as opposed to extra features 'tacked on'.Considerations such as these do belong more to a research

enviroment than to an immediate well-defined application in which we cannot afford the luxury of experimentation;but it is important that such a system is finalised and optimised as a result of performance within a 'real' enviroment or else we run the risk of making grossly incorrect assumptions about the nature of such an enviroment. An example of the importance of working within a 'real' enviroment concerns the distribution of input features.We have found that various means of optimising tree searching algorithms demonstrated,assuming a random distribution of input would be swamped by the actual distributions within the enviroments that we have investigated (chapter 5).

The practical utility of a syntax checking program written in a high level language,which is the application of our system that we demonstrate in depth (chapter 5) has been stated by DAY (1970) as follows,"Compilers do not do all the syntax checking which may be desired as they are almost always biased towards some manufacturer's version of the language,and hardly ever check for an internationally agreed standard.Consequently there arises a need for syntax checkers written in a high level language (in order to be machine independent) ... and flag every statement which does not conform to the ASI standard".To this can be added that our system is constructed to enable easy alteration to meet the demands of specific non-standard implementations of a language.The alteration is accomplished by simply adding and/or deleting items from the experience tree and hence changing the language definition.

3.32 A system for language modification.

Further immediate potential uses of our analyser which we might term peripheral,as distinct from a complete computer system based upon its central tenets,cover all the

applications described by ALPIAR (1971) of his "Double syntax oriented processing".These applications are described below.

3.321 Modification of a program to conform to a 'local' dialect.

The analyser can be easily altered manually (by changing the stored pattern,1.3) to cause it to modify or modify differently some language feature(s) and it can then be used to effect this modification throughout all programs processed.

For example the particular numbers associated with the standard input and output devices will vary from one installation to another and thus are a feature of local dialect.

3.322 The temporary alteration of a program as a debugging aid or to test special features.

Again the stored pattern can be temporarily adjusted to add,say,an output statement or an output subroutine reference statement following particular language features, as a debugging aid.

3.323 Translation of non-standard programs back to a standard form.

As an example of this application;FORTRAN variable names which are limited to six characters in the standard versions, are allowed more in some non-standard versions.Our analyser could just flag this feature (i.e. variables of length greater than six) every time it arises or modify,say,truncate the actual variables involved.

Alpiar emphasises the provisional nature of the above applications and suggests that on numerous occasions only

one modifying run is required to perform alterations that are slight but occur repeatedly."Under such circumstances it becomes economical to employ an automatic processor even when the latter is somewhat inefficient by reason of its generality".

3.33 Applications in the general context of 'artificial intelligence' research.

The basic design principles of this project include techniques that are already found in the 'artificial intelligence' field.In this section we first show how our system might usefully be considered as an artificial intelligence project.We then cite a selection of publications which display one or more of the major facets of our basic principles.

Artificial intelligence projects can be roughly divided into two classes:the first concerns man's highly formal intellectual pursuits,such as playing chess or theorem proving;the second concerns much more informal,ill-defined activities that can be said to necessitate intelligence, such as man's very powerful communicative ability which includes reading,writing,talking,listening,etc.

Our project can be viewed as an approach to artificial intelligence within the latter of the above two classes. In the fairly well-defined situation of a human attempting to communicate with a machine using a programming language there exists the possibility of exploiting the informality of the problem;a task that is at present to big to attempt without gross over-simplifications when natural language is the medium of communication.The further important point is that the situation is sufficiently restricted to enable our system to function within a completely 'real' enviroment

and have realistic criteria for assessing the quality of the results.The results of such an investigation could be used to develop a system that displayed 'common sense' within this enviroment.

The ability to process poor quality data,which can contain a considerable degree of redundancy,in a reasonable time is fundamental to the solution of ill-defined problems.Our system utilises approximate matching techniques and 'calculated guesses' in an attempt to reproduce this ability. Another avenue of approach to the same situation is the incorporation of a degree of allowable error in return for a reasonable processing efficiency in terms of speed and conciseness versus the number of incorrect or unsatisfactory results.We have argued elsewhere (JAMES & PARTRIDGE 1972) that the human ability to process incomplete data and make guesses that occasionally give rise to mistakes,is perhaps fundamental to intelligent behaviour.A similar idea was recently voiced at the Infotech State of the Art Lecture entitled "1980","there may be a limit to general intelligence. Humans are characterised by being very intelligent but also unreliable in the results they produce.Machines tend to exhibit the same characteristics of unreliability as they become more intelligent."(PALME 1972).

Such a philosophy appears in many models of intelligent behaviour but very rarely in papers on the design or implementation of computer software (but see exceptions BLOOM 1970 and RANDELL 1971,discussed in 3.14).Most examples of allowable error implicitly or explicitly are to be found within the more psychologically orientated artificial intelligence papers.

Thus we will now cite and extract the relevant detail from

a selection of such papers which under our interpeetation contain a philosophy akin to ours.

KISS (1969a) in his paper,"Steps towards a model of word selection",says,"that probabilistic considerations should dominate the discussion of the word store,will occasion no surprise for anyone who ever felt frustrated at his apparent inability to recover a familiar word or name from his store on some occasions.We all accept the fact that some rare items may not be available when needed ... occasionally even quite well-known words are 'lost' in the system ... an information retrieval system will provide that information with a probability which can be less than one,and sometimes may be zero".

In GOOD's (1965) typically flamboyant and interesting article,"Speculations concerning the first ultraintelligent machine",he suggests that a big step towards this somewhat ambitious goal would be the simulation of the "magic" of human recall.He continues,to suggest that a weighted or statistical system would be used and retrieval proceeds in order of descending probability,stopping the process at some threshold value.Within our more limited context this describes our experience tree (1.02) and the method by which we scan it.The concern with probabilistic considerations in both of the above papers stresses the necessity for allowable error,which probability implies.Our system,as human speech perception according to GOOD (1965),involves probabilistic regeneration.In our system the incoming statement is processed and replaced by a construction from our stored patterns and lower level tree items,both of which have associated confidence levels (1.13).It is precisely this regeneration which enables us to correct

trivial errors with no loss in processing efficiency.

A model of the human brain proposed by SPARKES (1969) contains three basic features:a short-term memory,a long-term memory which is associative and to which permanent information can be added and, a "similarity function store".All three are basic features of our system in the form of:the input buffer, the experience tree and,the strategy tree,respectively. Sparkes also stresses the need to "jump to a conclusion" which may then be reinforced or destroyed.

In a paper by NOTON (1970) we find one of the few attempts to explain "how the patterns are recognised under unfavourable real-world conditions".He suggests that the internal representation of the particular pattern directs the processing,"avoiding other patterns and background noise ... distorted patterns are handled by accepting a certain degree of inaccuracy at each step of the matching process ..." Such a system of recognition is well-known within linguistic methods of pattern recognition in which the internal representation of the pattern directs the recognition process (2.12).The usual systems will only attempt the approximate match,if at all,when the exact match has failed. But Noton proceeeds to suggest the feature that we have termed the confidence jump (1.21),"... recognition of a well-known pattern may gradually become possible without completing the scanpath and verifying all features ... there is a gradual build-up of certainty as each feature is verified".We feel that it is an important function of economy for any system aspiring to intelligent behaviour that approximate matching is a general rule rather than a perturbation of the exact match situation that is introduced only after all else has failed.

GUZMAN's (1968) program "SEE",attempts to perceive three-dimensional bodies in a two-dimensional scene.He demonstrates the problems raised by curves,shadows and noise (which have analogies in our problem) and points to the need to "take a chance" in ambiguous situations.

We have instanced many examples to demonstrate that tolerant processing is now considered by many to be fundamental to the development of artificial intelligence. Thus we consider the tolerant processing ability of our analyser to be important in its development towards a system which exhibits 'common sense'.

3.34 An important side effect - the availability of a detailed pattern of use.

The first immediate practical application of our analyser might be to provide information to help future language and systems designers,who are beginning to appreciate the need for detailed knowledge of language usage (as stated,for instance,by GOOD 1969a and KNUTH 1970).Knuth considers information of two types;dynamic,concerning the frequency with which control is passed to each statement during execution of a program,and static,concerning the frequency with which each statement occurs in a program,as written. We are only concerned,at present,with statistics of the latter type.

The collection of extensive results from static analysis is only a by product of the need to utilise the data collected to improve both speed and accuracy of analysis.

4.0 Description of the implemented system.

Within this chapter the details of the implementation of the analyser are described. Extensive reference is made to the Appendices which contain a complete description of the finer implementation details.

Section 4.0 introduces our terminology and then describes how the experience tree hierarchy is constructed from input data to provide the initial language definition.

In section 4.1 we describe how the analysis of incoming statements is controlled by the experience tree and how this process is optimised.

In section 4.2 we describe the analyser's tolerance to errors of specification in the incoming statements.

Finally we describe the use of the stored pattern to 'translate' analysed statements and the collection of usage frequency statistics.

4.01 Terminology and definitions.

The terminology used to describe the data structures is based upon that of SUSSENGUTH (1963) and can be defined as follows:

GRAPH - a set of nodes with unilateral associations between them

BRANCH - an association between two nodes

PATH - a sequence of branches in which the terminal node of each branch coincides with the initial node of the succeeding branch

CIRCUIT - a path in which the initial node coincides with the terminal node

TREE - a graph which contains no circuits, has at most one branch entering each node *and one root*

Examples of two possible trees are given in Fig.I

In this project, we employ only binary trees, in which the nodes comprising filial sets are ordered in terms of precedence

ROOT - a node with no branches entering it.
Each tree must have one,and only one root,as illustrated in Fig.I

LINK-LEFT BRANCH - the association contained in the LINK-LEFT ADDRESS field of a tree node (see MINISLIP; APPENDIX I for node structure details) and represented diagrammatically as a vertical line between two nodes;see Fig.I for examples,node x is associated to node 1 via a link-left branch

LINK-RIGHT BRANCH - the association contained in the LINK-RIGHT ADDRESS field of a node (again see APPENDIX I) and represented diagrammatically as a horizontal line between two nodes;in Fig.I node 1 is associated to node i via a link-right branch

TERMINAL NODE - a node with no link-right branch leaving it.
Thus the main tree illustrated in Fig. I contains five terminal nodes,which are nodes j,k,m,n and o

LENGTH of a path - is the number of link-right branches between the first and last node of the path
As an example,in Fig.I,node o is at the end of a path length 3 from the tree root

j^{th} LEVEL - nodes which lie at the end of a path of length j-1 from the tree root are on the j^{th} level.

FILIAL SET - is a set of nodes lying on the same level and associated via link-left branches.
In Fig.I,nodes 1,x and m comprise a filial set.

PARENT node - each node is the parent node of the filial set (which may be only one) to which it is associated via a link-right branch on the next higher level.Thus the tree root is the only node without a parent and terminal nodes are the only nodes that are not parents.

REACHABLE - a node j is reachable from node i if there is a path from i to j.

Two special cases of 'reachable' may usefully be distinguished

DIRECTLY LEFT - a node j is directly left of node i if the path is composed solely of link-left branches. Thus a node can only be directly left of nodes within the same filial set.

DIRECTLY RIGHT - a node j is directly right of a node i if the path from i to j is composed solely of link-right branches. In Fig.I, a node j is in fact directly right of node i.

ITEM - a tree item is the representation within a tree of a language feature. Each item is composed of one and only one node from within each level of the tree, where the node in the n^{th} level is REACHABLE from the nodes in each of the lower levels. Any non-terminal node may form part of more than one item.

In Fig.I, the main tree is composed of five items, node x is part of three items and the item terminated by node o is composed of nodes q, x, p and o.

SUBTREE - a tree J is a subtree with respect to a tree (or subtree) I, if a node of tree I contains a reference to tree J.

In Fig.I, the tree J is a subtree with respect to tree I if a node of tree I, say, node p contains a reference to tree J.

ODDITY - (following LONGUET-HIGGINS & ORTONY 1968), the oddity of a path is the number of link-left branches within it. Thus the nodes of a filial set all have the same path length from the tree root but are ordered in terms of increasing oddity.

SEARCH LENGTH - is the number of nodes that have to be queried to find a particular node, starting at the tree root. Thus the search length to any node is equal to, 'path length+oddity+1'. In Fig.I, the length of the search necesary to find the item terminated by node k, is 3+1+1 or 5; the search length for the item terminated by node o, which is on the same level as k, is 3+3+1 or 7.

SUCCESSOR NODE - if and only if node i is the successor of node j, then i is directly right of j at the end of a path length one. Thus only the lowest oddity member of a filial set is the successor of the set's parent; terminal nodes are the only nodes that do not have successors. In Fig.I, node 1 is the successor of the root node q.

ALTERNATIVE NODE - all nodes in a filial set are alternatives to each other, as filial sets are ordered in terms of oddity so are alternatives.

In Fig.I, node i has no alternatives, whilst nodes 1, x and m are alternatives to each other.

PRECEDENCE - the nodes of a filial set are ordered in terms of precedence. The highest precedence node being the successor of the parent node and the precedence of the rest of the set decreases as the oddity increases. Thus in Fig.I, nodes 1, x and m are ordered in that precedence and node 1 has a precedence of one greater than node x.

MAIN TREE - the main tree during any processing is the tree originally accessed; all other trees entered (which may include the original tree again) are only entered as a result of subtree reference nodes and are thus subtrees with respect to the current processing.

Fig. I - diagrammatic representation of trees

We have adopted the following representation of our tree structures, which are a departure from previous conventions (see SUSSENGUTH 1963 for example). The convention has been chosen as the most perspicuous in that the items that we process are most naturally displayed horizontally from left to right; and also this form is most suggestive of our implementation of these structures within the machine (see APPENDIX I).

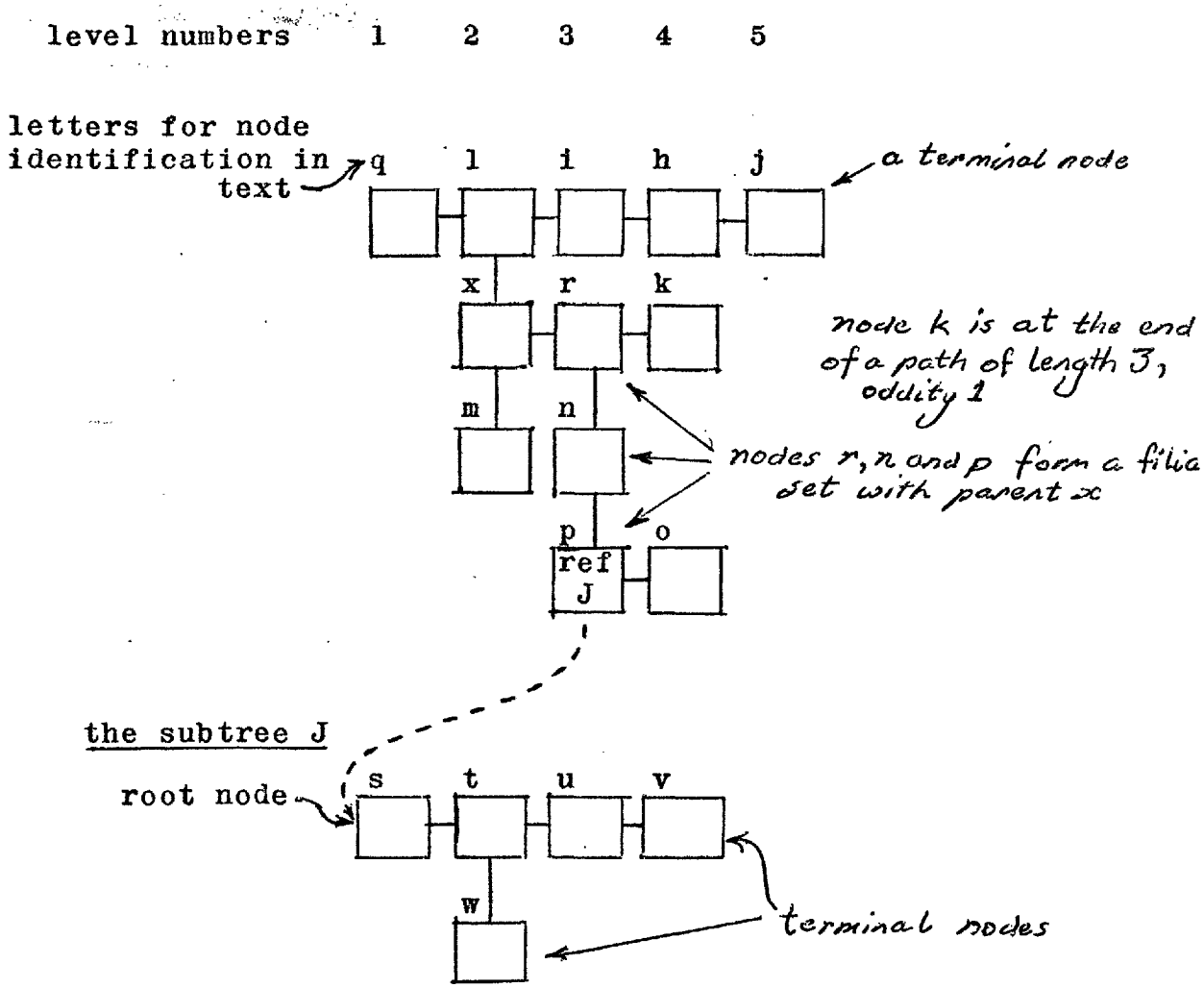
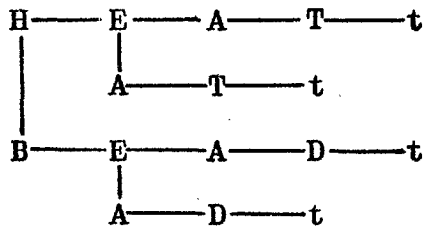
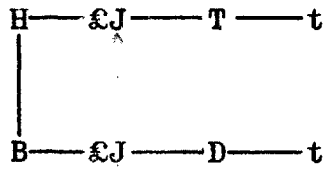


Fig. II - the allocation of the four language features, HEAT, BEAD, HAT and BAD

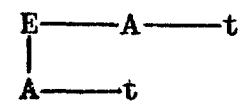
IIa - as a single tree



IIb - as a tree and subtree J



the subtree J



key: a single upper case character represents an immediate symbol,

a subtree reference is represented as £X, where X is the name of the subtree referenced,

a terminal node is represented as t

4.02 The initial language description and the construction of the tree hierarchy - the experience tree

The items processed are strings of characters in which we allocate one character to a tree node, together with a termina node. Thus although one or more items may (and usually do) share one or more nodes in common, there is a one-to-one correspondence between terminal nodes and items.

The first character of an item occupies a level one node in the tree, not necessarily the root node, and successive characters occupy a node in each successive level.

Each node also contains an identity number (ID) or 'strategy code' which specifies the interpretation of the node contents and also indicates the node's function, because it directs the processing of the node. For example, the strategy code 1 indicates that the node contains a immediate symbol and is to be processed accordingly, whilst the strategy code 7 indicates that the node character refers to a tree name and hence we have a subtree reference node.

Fig.II is an illustration of the allocation of the four language features, BEAD, HEAT, BAD and HAT; in Fig.IIa as four items in a single tree and in Fig.IIb as four items and using a subtree J.

Note that the use of a subtree results in the saving of three nodes over the single tree allocation. The question of tree and subtree optimum allocation is discussed below (4.03).

4.021 The 'stored pattern' for translation of the analysed statements.

The main tree allocation is extended to include a 'pattern' of each item. The pattern is a sequence of nodes following the allocation described above. The pattern is a device which controls what is output after the

statement has been analysed.

Items within the main tree of the total experience tree hierarchy are 'patterned' either explicitly item by item using routines ADD or SETUP (APPENDIX II,I/07 and I/01),or the complete main tree can be automatically patterned using a routine which transforms the tree items into the desired patterns.For example,routine PATERN (APP.II,RP2) will automatically add the patterns necessary for the analyser to function as a syntax checking system for any language.

Patterns are removed with routine PATOUT (APP.II,RP3) or with routine DELETE (APP.II,I/08) which also removes the associated tree item.

Thus by simply altering the patterns associated with various tree items we can translate analysed statements in many different ways(see 4.3 for details).

4.022 The insertion of the 'confidence jump' nodes for the efficient processing of trivially incorrect statements.

The 'jump nodes' which initiate and control the 'confidence jump' mechanism are inserted in the experience tree automatically with routine JUMPIN (APP.II,RP4) and automatically deleted by means of routine JMPOUT (APP.II,RP5). The value of a parameter passed to the insertion routine (RP4) dictates the degree of statistical redundancy,if any, to be added to the structural redundancy that the confidence jump mechanism exploits(1.21).

4.03 Trees and subtrees - generalisation versus efficiency.

It is usually impractical if not impossible within a finite space to enumerate specifically,all the grammatical statements of any practically useful language.Thus in the definition of such a language it becomes necessary to condense the description by extracting collections of

commonly occurring similar features;enumerating the contents of each collection only once,then referencing hierarchical structures of these collections from the appropriate positions within a main structure.These collections are logically equivalent to subroutines.

From the point of view of having to describe a language for our analyser to use,the first question that we are confronted with is,'what is a useful subroutine?' - a leading question to say the least!One important aspect of this problem hinges upon the criterion of similarity used,and we pursue this point with respect to the possibility of automatic subroutine discovery in chapter 6.

The initial description of the language FORTRAN that we have used to obtain the detailed results of our analyser that are described in chapter 5,contains subtrees of syntactically similar features(detailed illustration in APPENDIX VIII).

The initial division of the language definition into the hierarchical structure of cross referencing subtrees is specified by the input to routine SETUP (APP.II,I/01). Generally the more subtrees used the less space will the total hierarchy occupy.But also the more complex and thus time consuming will be the process of analysis by reason of the increasing number of subtree references that will have to be processed before immediate symbols are encountered.

Thus the initial experience tree hierarchy contructed should represent some position of compromise between space saving generality and time saving particularisation.The promotion mechanism (1.122 and 4.122 for details) will automatically remove some unwanted generalisation and thus alter the initial experience tree hierarchy to increase processing efficiency.

4.04 The tree building process.

The tree building process is the construction of the hierarchical definition,of main trees and subtrees,of the language whose statements are to be processed.This initial structure directs the processing and develops into the experience tree (1.02) by virtue of the continual analysis of incoming statements and the collection of analytical information.

Information for the initial construction of this tree hierarchy is coded (as described in detail with examples in APPENDIX III,ii) and input by means of routine SETUP (APP.II, I/01).

First the names of all the trees and subtrees to be constructed is input.Then the tree items are input one by one,and each preceded by the name of the tree in which it is to be inserted.The order in which items are inserted in any tree does effect the initial tree structure (it effects the order of precedence of nodes in filial sets) but as the branch swopping mechanism (1.121) will change this ordering to produce the current optimum,the initial ordering is not of prime importance.

As each coded item is input,the tree building process inserts a sequence of nodes within the named tree.Now the number of new nodes added is not necessarily equal to (and is usually much less than) the number of nodes specified by the input tree item;because before a new node is inserted in the tree the possible position of insertion is examined, and if an instance of the proposed node already exists in an equivalent position the processor discards the current data features and considers the next node features input with respect to the next position within the tree.The result

is that equivalent nodes are not found in equivalent positions within the tree and that any number of items can have any number of nodes in common within a tree.

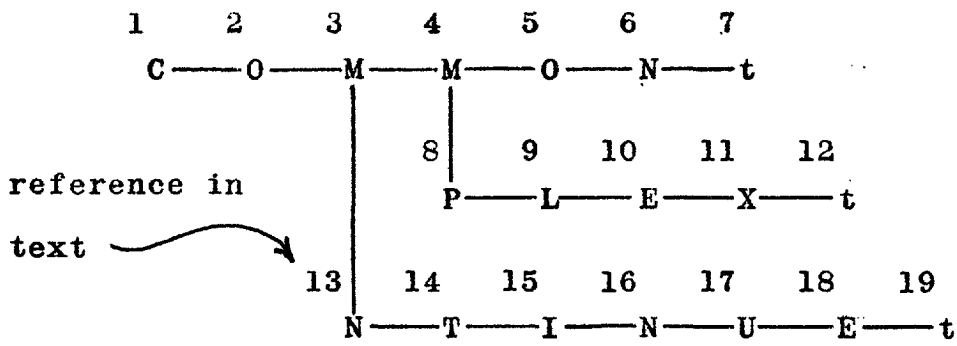
As an example of the tree building process;if we assume that the character 'F' has been input as a tree name,then the following data items,in the sequence illustrated would cause the construction of the partial syntax tree of Fig.III. The items input are:

F 1C 10 1M 1M 10 1N 2

F 1C 10 1M 1P 1L 1E 1X 2

F 1C 10 1N 1T 1I 1N 1U 1E 2

Fig. III - a partial syntax tree of the FORTRAN keywords,COMMON,COMPLEX and CONTINUE, tree F



Note,that the order in which the second and third items are input could be reversed and it would make no difference to the initial tree structure(for they do not have any non-identical nodes in the same filial set).

When the basic tree hierarchy has been constructed,patterns appropriate to the desired translation of analysed statements must be added to the main trees as described above (4.021).

Nodes to initiate and control the confidence junp mechanism may be added as described above(4.022).

Finally items may be deleted by means of routine DELETE (APP.II,I/08).

4.10 The analysis of incoming statements and methods of optimisation.

In this section we describe the analysis of incoming statements and the consequent building up of 'experience' within the basic tree hierarchy described above (4.04). The total tree hierarchy plus analytical information is 'the experience tree'.

In sub-section 4.12 we describe the use of this analytical information in the restructuring of the experience tree and optimisation of the subsequent analysis of further incoming statements.

The details of translation of analysed statements is dealt with in section 4.3,and the use of the confidence jump mechanism during analysis is described in section 4.2.

Within this section we only consider the analysis of 'correct' statements and thus although the analysis is generally controlled by the interaction of a general strategy from the strategy tree (1.22) and the strategy codes within the experience tree,we assume that the simplest general strategy,which is essentially 'obey the strategy codes', is in force.

4.11 The analysis and collection of analytical information.

First we will consider the analysis of the FORTRAN statement,'CONTINUE' as directed by the tree illustrated in Fig.III.

The first input statement character,'C',matches the tree root node data character,'C'(node 1 in Fig,III).We obtain the next input statement character,'O',and proceed to the successor node (node 2) via the link-right branch of node 1, as directed by the strategy code for immediate symbols (see APP.III,1 for ID=1).This match is also successful so

the process is repeated.

The next attempted match fails;the input statement character is 'N',and the current node (node 3) contains the immediate symbol 'M'.Therefore the strategy code of the current tree node (again it is 1) directs control to the first (and in this case only) alternative node via the link-left branch.Thus the current tree node is now node 13,and the analysis proceeds as directed by the new strategy code (which is again 1,as the node contains an immediate symbol).

The current input statement character is still 'N' and the current tree node data character is the immediate symbol 'N',thus we have a successful match.The next input statement character 'T' is obtain and control is passed to the successor of the current tree node,that is to node 14.

Now successive input statement characters will match the immediate symbols contained in successive nodes 15,16,17 and 18.

Now the current tree node is node 19,a terminal node and its strategy code (which is 2,see APP.III,i) assumes control. Successful completion of processing with the current tree item is tentatively assumed and then confirmed by the facts that the input statement is exhausted and the tree item is on a main tree.(If it had not been confirmed the node 19, would have been examined for alternatives).Thus a total successful match is reported.

Notice that successive tree nodes define successive matching processes in time.But,these processes do <u>not</u> necessarily involve matching successive characters of the input statement in a left-to-right manner.However,we use a left-to-right manner for most of our matching processes (but see strategy code 6,APP.III,i for an exception), because it appears to be the most natural method of

processing languages which are written from left to right.

4.111 The tree directed analysis - the strategy codes.

As emphasized in the above example (4.11),the analysis of incoming statements is directed by the experience tree. Each node in the experience tree contains a 'strategy code' as well as other data items.The strategy code specifies how the rest of the node contents are to be interpreted (for instance,data character can be an immediate symbol or a subtree reference),and also directs the processing by means of associated routines.The strategy codes currently implemented are described and illustrated in APPENDIX III,i and the associated routines are in section MP of APPENDIX II.

The set of strategy codes with the associated interpreter can in a more general sense be considered as a tree processing language.The language is flexible in that new codes for particular language features can be added independently of the existing codes (as was the code,6,to deal with the 'Hollerith count' feature of FORTRAN).

4.112 The usage frequency counts.

An outline description and the underlying principles of the frequency counts have been presented earlier (1.13 and 3.11 respectively).The frequency counts are the means by which analytical information is collected and stored within the experience tree.

The frequency count associated with each tree item is contained within the terminal node of that item(detailed terminal node structure in APP.III,i).The count location within each terminal node successfully processed is incremented by one when a total successful analysis of the current input statement is reported.

Thus after any period of analysis we have available within the experience tree a count of how many times each tree

item has been successfully matched against the statements analysed.It is this analytical information that is utilised to restructure the experience tree as described below(4.12).

4.12 The restructuring process - heuristic techniques.

The experience tree is restructured periodically on the basis of the frequency counts in the terminal nodes. There are two distinct mechanisms for restructuring the hierarchy of trees,both are intended to simplify and hence optimise the subsequent process of analysis.

These two distinct mechanisms are described in detail below.

4.121 The branch swopping mechanism.

The basic idea for the branch swopping mechanism was proposed by LONGUET-HIGGINS & ORTONY (1963);they maintain that natural information sources tend to generate sequences in which the probability of occurrence of a particular member is strongly dependent upon the members that have already occurred.They demonstrate a branch swopping algorithm which moves a sequence within a tree to a higher precedence position every time it is used.Their results show that the average number of nodes queried to match a word of natural language text is substantially less when using the flexible rather than the inflexible tree.

We have adopted and extended this branch swopping idea;its relevance to our analysis process can be described as follows.

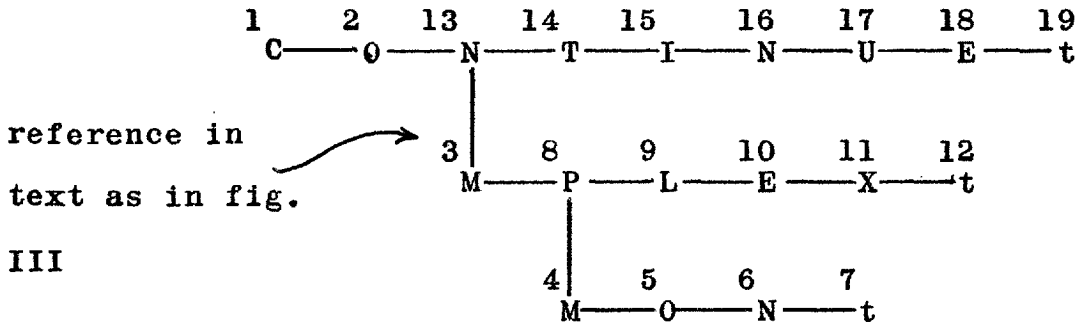
Given that each character match occupies a certain time whether it is successful or unsuccessful,it is clearly important to arrange the trees so that the number of nodes queried per successful complete statement match is minimised. The commonest statements must therefore involve as few alternative nodes as possible.The branch swopping mechanism alters,if necessary,the order of the alternative nodes in

each filial set,placing the nodes with the highest frequency counts in the highest precedence positions.

As an example of the precedence during the analysis process implied by the tree structure and action of the branch swopping mechanism,consider the tree illustrated in Fig.III. This tree is constructed from the FORTRAN statements 'COMMON', 'COMPLEX' and 'CONTINUE'.The statement 'COMMON' is in the highest precedence position as no alternative matches are involved.The other two statements are of equal precedence as one alternative match is involved in the matching of each complete statement.

Figure IV represents the same partial syntax tree after it has been restructured by the branch swopping mechanism to allow for 'CONTINUE' being encountered most frequently and 'COMPLEX' more frequently than 'COMMON'.

Fig.IV - the partial syntax tree of fig.III after the action of the branch swopping mechanism.



In figure IV the three statements have a definite order of precedence,'CONTINUE','COMPLEX' and 'COMMON'.The precedence of items can also be viewed in terms of the oddity (4.01) within the search path,thus the above statements have oddities 0,1 and 2 respectively.

Notice that the apparently drastic rearrangement that

produces figure IV from figure III,is in fact simply the 'swopping' of two link-right branches (node 2 to 3,and node 3 to 4,now 2 to 13 and 3 to 8,respectively) and the reversal of two link-left branches (node 3 to 13 and node 4 to 8,now 13 to 3 and 8 to 4,respectively).Thus in this case the implementation of the branch swopping mechanism amounts to altering the links (addresses) in four nodes (nodes 2,13,3 and 8).

4.122 The promotion mechanism.

This mechanism became apparent from a consideration of the branch swopping mechanism in conjunction with our hierarchical structure of trees and nested subtrees (i.e. the experience tree).To minimise the search time for any particular statement we not only have to consider the number of alternatives involved,but also the number of subtree reference nodes that have to be processed before the immediate symbols are encountered.

This mechanism is introduced and exampled earlier (1.122), within this sub-section we will pursue the implementation details.The use of subtrees and subtree reference nodes is not a logical necessity but a result of purely practical constraints (see 4.03).Thus we are presented with the possibility of minimising the average search time by minimising the number of subtree reference nodes that are processed when matching the incoming statements.

In practical terms this means that we must include specific representations of the language features in the main tree or as near to the top of the nested structures as possible. Although this procedure will simplify the analysis of statements (by eliminating nested subroutine references) and thus decrease search time,it will also give rise to an

increase in the size of the total data structure,trees and subtrees.So,in contrast to the branch swopping mechanism, the extent to which this promotion technique can be exploited must represent a compromise between efficiency of analysis and size of the experience tree.

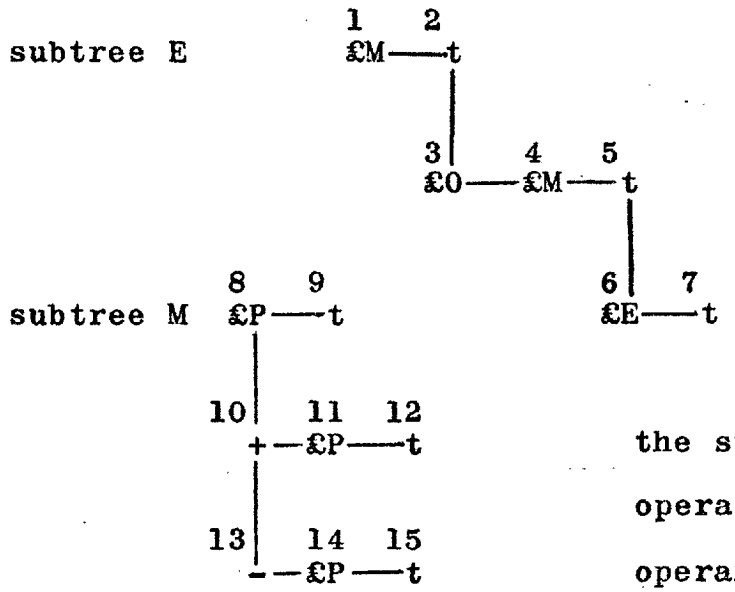
The logical structure of the implemented mechanism is as follows:first,each tree (subtree) is examined to determine from an examination of the frequency counts,if a small proportion of its items have been used on the vast majority of entries to the subtree.That is,each tree is examined to determine if the frequency with which items have been accessed is sufficiently non-uniform (as specified by the value of three arguments passed to the promotion routine PROMOT,APP.II,RP13).

Second,as soon as a sufficiently non-uniform distribution of frequency counts is found in any particular tree,all the trees (and subtrees) are examined to determine how many times the particular tree is referenced.Then if the increase in the total hierarchy size that this 'promotion' would cause can be accomodated the promotion proceeds;else it is abandoned.

Third,the item(s) to be promoted are effectively deleted from the subtree (they are in fact demoted to the lowest precedence position in the subtree to simplify the reverse mechanism,see below),and inserted in all trees in which the subtree reference nodes are found,but with one higher precedence(or one less oddity) than the actual subtree reference nodes.

As a further example (the previous one is in 1.122), consider the subtree for dealing with arithmetic expressions, E,and the subtree for dealing with prefix operators,M;both illustrated in figure V.

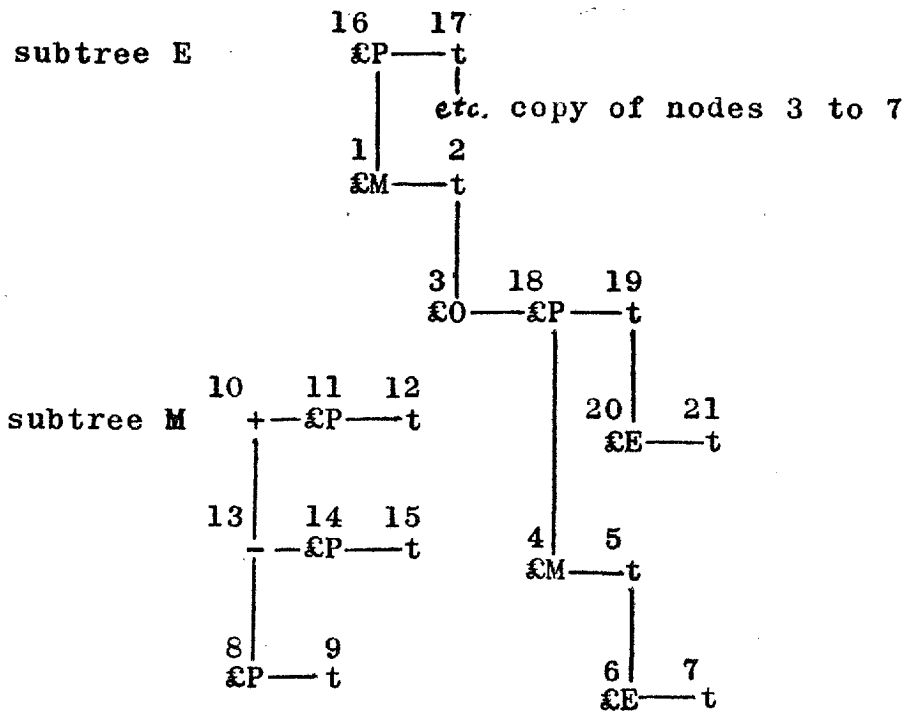
Fig.V - the subtrees E and M



the subtree O,deals with operators and subtree P with operands

Now if the frequency count information in subtree M indicates that the item '£P t' has been accessed on,say,95% of references to subtree M,and the only referencing tree is subtree E;then this item would be promoted to give the subtrees illustrated in figure VI.

Fig.VI - the subtrees E and M of figure V after 'promotion'



Thus at the cost of six extra nodes,we would expect that in the future analysis of arithmetic expressions that in the majority of cases (i.e. approximately 95%),the time required for one subtree reference will be saved for every operand analysed.

This mechanism requires a complementary mechanism to accomodate changes in the usage of items (a previously very popular item might suddenly fall from favour) and reduce the overall tree hierarchy size.Thus to reduce the total hierarchy size (but not,note,to necessarily increase analysis efficiency) we should delete all specific instances of a rarely accessed item and insert it in a subtree that is referenced from positions equivalent to all the previous specific instances of the item.Thus this complementary mechanism is tantamount to a 'subroutine discovery' procedure which represents a formidable problem even in the limited context of our system (see discussion chapter 6).We bypass this problem at the expense of having redundant items within the trees,for example,the lowest precedence item in subtree M of figure VI is redundant.

But the fact that we maintain these redundant items after the action of the promotion mechanism means that the simple complementary mechanism which 'demotes' items that have 'fallen from favour' also increases the particularisation power of the promotion mechanism.

For example,if after a period of analysis with the subtrees of figure VI as part of the experience tree,we find that the frequency count in node 19 of subtree E indicates that most (again,say 95%) of arithmetic expressions are simply one operand;then the structure composed of nodes 18,19,20 and 21 is 'wasted space' and is automatically deleted.

Thus the promotion mechanism has been particularised by two simple steps (initial promotion to all referencing positions, and later deletion of any under-used structures) to increasing the efficiency of analysis of single operand expressions only (which presumably occur most of the time).It is because of the redundant items after the initial promotion that under-used items can simply be deleted in the second step, for they are 'covered' by these previously redundant items. In chapter 5 we describe and illustrate a somewhat different result of this further particularisation of the promotion mechanism.

The promotion mechanism,which can be described in more general terms as 'the removal of unwanted generalisation', appears to be a generalisation of the "memo" function concept due to MICHIE (and discussed in 2.11) which was in turn a development from SAMUEL's(1959) investigation of machine learning using the game of checkers.

4.13 Further utilisation of frequency counts to control the process of analysis.

As described earlier (1.13),the frequency counts are utilised to provide confidence levels throughout the experience tree.Thus we have available at all times during the process of analysis a measure of the relative likelihood of occurrence of all items within the experience tree and thus all features of the language.

As described above (4.112) a usage frequency count is maintained within the terminal node of every tree item.It turns out that these frequency counts are sufficient to specify the relative frequency of past occurrence of every node in each tree.

Thus we have a routine (SETJNC,APP.II,RP10) which will

scan a tree and insert the correct relative frequencies,as dictated by the current frequency counts,in each node (APP. III,i for node structure details) of every filial set with more than one member,plus the tree root node (which may or may not be a member of a filial set of greater than one). It is not necessary to insert relative frequencies within nodes with no alternatives (i.e. filial sets of one),because the correct value is simply that of the preceding node.

It is these relative frequencies that are used as confidence levels in the control of the analysis process (see below 4.14 and 4.2).

4.14 The introduction of allowable error as a trade-off against processing efficiency.

Allowable error has been described and discussed earlier (1.14 and 3.14).In this sub-section we describe exactly how allowable error can be introduced to increase processing efficiency in our analyser.

We introduce allowable error by means of the confidence levels described above(4.13).The process of analysis is generalised so as to treat a tree node that has no alternative as the limiting case of nodes with successively less likely alternatives.Thus when functioning with no allowable error the full experience tree is used.Then when functioning with allowable error the analyser simply treats alternative nodes that have confidence levels of less than some specified value,as if they did not exist.

The process of analysis becomes more efficient because of the effectively more compact experience tree - fewer wrong alternatives are examined.This extra processing efficiency is bought at the cost of the possibility of a small percentage of processing errors.This proportion of errors of commission

may be acceptable in comparison of overall system reliability or,'caught' by the next lower precedence strategy in the strategy tree (1.22),operating with no allowable error.This 'catching' of the small proportion of errors is again an example of increasing overall efficiency (by the introduction of allowable error) at the cost of occasional inefficiences (when an 'error' is processed by the second general strategy).

The degree of allowable error introduced is given by the value of the relative frequency below which the analyser ignores alternative nodes and is specified by the general strategy in force (1.22 and 4.22,below,for details).

Allowable error may also be exploited by the confidence jump mechanism (see 4.21 below).

4.20 The recovery of errors of specification.

In this sub-section we describe in detail the action of the confidence jump mechanism for the efficient processing of trivially incorrect input (4.21).Then we describe the interaction of the general strategies and strategy codes in the analysis of non-trivial errors of specification(4.22).

4.21 The confidence jump mechanism for trivial errors.

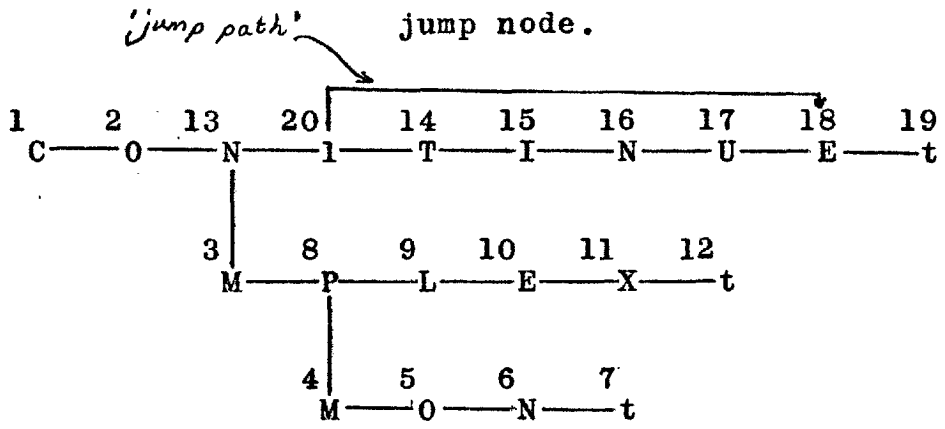
The jump nodes which initiate and control the confidence jump mechanism are inserted in the experience tree as described above (4.022).

During the process of analysis the confidence jump mechanism is initiated if control is passed to a jump node (ID or strategy code 8,APP.III,i) and the confidence level of the process is sufficiently high;else the jump node simply transfers control to its successor and the analysis procedds as normal.

As an example of the confidence jump mechanism,consider

the insertion of a jump node in the partial syntax tree of figure IV,as illustrated below in figure VII.

Fig. VII - the partial syntax of figure III after the insertion of a jump node.



Consider the analysis of the FORTRAN statement 'CONTINUE'. The first three successive input statement characters,'C','O' and 'N' are matched against the first three successive tree node data characters (in nodes 1,2 and 13).Then the current input statement character to be matched is 'T' and the current tree node is node 20,a jump node.

Assuming the confidence of the matching so far is sufficiently high the jump mechanism is initiated;else the strategy code 8(node 20) simply transfers control to its successor, node 14 and matching continues as normal.

The jump mechanism first involves noting the immediate symbol at the end of the jump path in this case 'E' in node 18 (note,jump nodes are the single exception to the rule that link-left branches from a node are to the next alternative) Then control scans down the input statement until the i^{th} occurrence of the character 'E',where i is the integer contained in the jump node,in this case i=1.Now the current input statement character is 'E'.

The jump node strategy code then causes control to be

transferred along the 'jump path' to node 18.Thus node 18 now assumes control and matches the current input statement character and its own data character (both 'E's) and transfers control to its successor,node 19.

The terminal node 19,now assumes control and as it is on a main tree and the input statement is exhausted,a total successful match is reported.

Notice that the input statement between characters 'N' and 'E' werenot checked against the tree nodes 14,15,16 and 17,and thus processing time would be saved in comparison with an exact character by character match.Also that any or no characters could have occurred between the 'N' and final 'E' in the input statement.The only character that could not have occurred was another 'E' for the strategy code,in this case was checking for the first occurrence of an 'E' in the input statement.

So 'trivially incorrect' means that any or no characters can occur in the statement between the 'N' and the final 'E', except another 'E'.Similarly,'trivially incorrect' has a well-defined but very specific meaning for each statement in the language.

The confidence jump mechanism extends naturally to encompass the allowable error technique described above (4.14).Jump nodes are inserted to direct jumps not only over structurally redundant nodes but also statistically redundant ones.Nodes whose first alternative contains a relative frequency of less than some specified value (the allowable error,which is passed to the insertion routine JUMPIN,APP.II,RP4) are considered redundant and can thus be 'jumped' over by the mechanism.

Thus if the keywords 'COMPLEX' and 'COMMON' had occurred

in statistically insignificant proportions,as indicated by the relative frequency value in node 3 of figure VII,then the jump node (node 20) would have been inserted between nodes 2 and 13.Thus any statement beginning with 'CO' and ending in 'E' would be analysed as 'CONTINUE'.

4.22 The strategy tree and force matching of non-trivial errors.

As described earlier (1.22) the matching of an input statement is not,in general, directed by the experience tree alone,but by a combination of the experience tree and a general strategy.In the analyses described so far (4.11 and 4.21) we have assumed that the general strategy used is the one that can be described as,'obey the strategy codes'.

If the statement to be analysed is more than trivially incorrect then the above described matching process will not suffice;it is in this situation that another general strategy would be invoked to interact with the strategy codes to 'force' a match between the incorrect statement and the experience tree.

The general strategies are maintained in a linear tree, the 'strategy tree',and thus have a definite order of precedence.Thus when the current general strategy fails to force a successful match with an input statement,the next general strategy is used.

A general strategy contains data to modify the overall analysis process to accommodate likely errors.A useful general strategy might be,'assume one character is missing from the input statement'.

Clearly any incorrect statement can be modified to a correct form (as defined by the current experience tree) by some combination of the three general strategies;

'n characters omitted',

'n characters inserted' and

'n characters wrong'.

Where n is a positive integer.Useful values of n would probably be 1,2 and 3.

There is a further feature that must be included within the general strategies and that is,at what point in the input statement the error occurrs;for it is by no means necessarily at the point where it becomes apparent that we are dealing with a non-trivially incorrect statement.

Thus there is a need for the general strategy to dictate not only which type of error to search for,as exemplified by the three general cases above,but also at which point in the input statement the force matching should start.Generally we must assume that the error occurs m characters earlier than detected, so the three generalised strategies must be amended to;

'backspace m characters,and search assuming n character omitted'

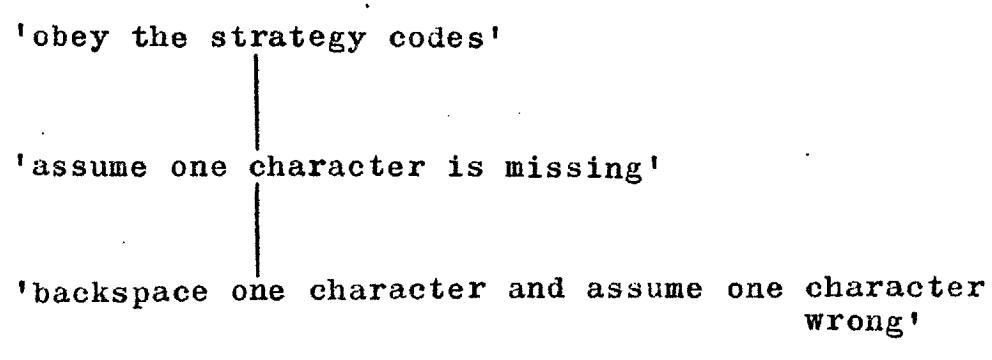
'backspace m characters,and search assuming n characters inserted' and

'backspace m characters,and search assuming n characters are wrong'.

Several strategy trees are illustrated in APPENDIX IX.

As an example of force matching consider the analysis of the FORTRAN statement 'COMTINUE' as directed by the experience tree of figure III and the strategy tree of figure VIII.

Fig. VIII - a strategy tree



The first two successive input statement characters,'C' and 'O' are successfully matched against the immediate symbols contained in the two successive nodes from the tree root (nodes 1 and 2),using the first general strategy. Similarly the third input statement character matches the data character of the now cuurent tree node (node 3);control is transferred to the successor node (node 4) and the next input statement character 'T',is obtained.

The current tree node again contains an immediate symbol and so the strategy code attempts a match with the current input statement character 'T';the match fails.The strategy code then transfers control to the alternative node (node 8). Node 8 again contains an immediate symbol and thus attempts a match with the current input statement character,still 'T'; the match again fails,but this time there is no alternative node to transfer control to.

Thus we have an incorrect statement and the first alternative general strategy,'assume one character is missing',is obtained.

This general strategy attempts to match the current input character 'T' with the immediate symbol in node 5 (assuming an 'M' is missing) but the match fails.Then the 'T' is matched against the immediate symbol in node 9 (assuming an 'P' is missing) but again the match fails;there are no more alternatives for this general strategy so the next general strategy is obtained.

Now the general strategy in force is 'backspace one character and assume one character wrong',the current input statement symbol is still 'T' and the current tree node is node 4 (the most likely node that failed to match originally). The backspace feature of the strategy causes the current

input statement character to be reset back one character and is thus 'M' and the current tree node is reset as node 3.

The second half of the general strategy determines which type of corrective search is to be performed and it is 'assume one character is wrong'.Thus the now current input statement character is assumed to be wrong,that is 'M',and thus we do not attempt to match it against the first immediate symbol encountered,but attempt to match the next input statement character,'T' against the second immediate symbol encountered.

It is tentatively assumed that 'M' should be the immediate symbol within the current tree node (that is 'M' in node 3) and control is transferred to its successor,node 4.Node 4 contains an immediate symbol and thus its strategy code attempts a match,which fails (input character 'T' and node character 'M'),control is transferred to its alternative,node 8.The current tree node again contains an immediate symbol and a match is attempted,which again fails.

Now the current tree node (node 8) has no alternative to be examined but the node which was assumed to contain the correct input statement character (node 3),does have an alternative.

Thus control is transferred to this alternative,node 13, and the incorrect input statement character 'M' is now tentatively assumed to be 'N' (i.e. the immediate symbol in node 13).Again the next input statement character 'T' becomes the current character and control is transferred to the successor of node 13,which is node 14.

The now current tree node (node 14) again contains an immediate symbol so the strategy code attempts a match; which is successful (both 'T's).

Now as the tentative assumption as to which character was wrong and which character it should be,has been confirmed by the character match,the first general strategy is restored to control the analysis.The process of analysis will now proceed smoothly and match the remainder of the input statement characters 'I','N','U' and 'E' against the immediate symbols within the nodes 15,16,17 and 18.Control is then transferred to the terminal node (node 19) which suggests a successful match.This is confirmed by the fact that the input statement is exhausted and that the terminal node is on a main tree.Thus a total successful match is reported.

When we consider the many different possible general strategies that might reasonably be tried,compounded with the considerable complexity due to the experience tree structure containing subtrees nested to an arbitrary depth, it is not surprising that the force matching can prove to be a very time consuming process.So we have developed several techniques for increasing the efficiency of the force matching process.

First a frequency count is associated with each strategy in the strategy tree and is maintained in exactly the same way as those in the experience tree (4.112).That is,each time a general strategy is used successfully its associated count is incremented by one.So the strategy tree can be restructured by the branch swopping mechanism (4.121) in the same way as the experience tree so as to move the more useful general strategies to higher precedence positions.

We would expect the process of analysis and restructuring of the strategy tree to produce faster and 'better' corrections.

A further feature introduced to optimise the force match-

ing process is the use of the confidence levels within the experience tree (4.13),by means of the allowable error technique described above (4.14).The general strategies contain one further feature which controls the analysis by setting a threshold value and items within the experience tree with confidence levels below this value are ignored. Thus we prevent unlikely strategies from utilising very unlikely items of the experience tree and save time by avoiding searches in which success is 'doubly' unlikely.

A complete failure to deal with an input statement is reported when each general strategy in the tree has been unsuccessfully used and there are no further alternatives in the strategy tree.

4.30 The system output.

Within this section we describe the output from our analyser,firstly,the translation of analysed statements (4.31) and secondly,an important by-product,the usage frequency statistics that accumulate within the experience tree (4.32).

4.31 The translation of analysed statements.

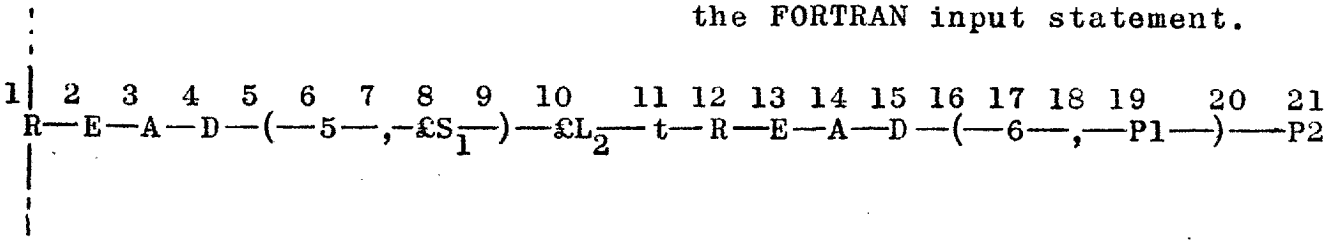
So far we have only described the analysis of incoming statements and,how and where the pattern is added to the experience tree (4.021).In this sub-section the detailed structure and function of the pattern is described.

A pattern is a linear sequence of nodes with strategy codes 3 or 4 (see APPENDIX III,i for node structure details), each main tree item is associated with one such pattern.The first node of a pattern is the successor of the main tree item terminal node (thus main tree terminal nodes are the single exception to the rule that terminal nodes have no successors in 4.01).

During the analysis of a statement various parameters are extracted (there may be none) and stored with an associated 'parameter number' for identification purposes,as directed by the experience tree.When the complete statement has been successfully analysed and the main tree item terminal node is the current tree node,control is transferred to its successor which is the first node of the associated pattern. This pattern node then directs the output of some feature, the value of a parameter or some immediate symbols,and then transfers control to its successor,another pattern node,and so on;until the final pattern node has completed its task and total successful processing of the current input statement is reported.

As an example,consider the main tree item required to translate the input device number 5 to say 6,in all FORTRAN statements in which it occurs.Figure IX illustrates such an item.

Fig. IX - the main tree item required for translating the device number in the FORTRAN input statement.



Note:pattern nodes are distinguished from analysis nodes, since they follow a terminal node,such as node 11 in figure IX.The two types of pattern node are;

1. outputting the parameter numbered following a 'P',

and 2. outputting the immediate symbol illustrated otherwise. Thus node 18 simply outputs a ','.

Also the parameter number to be associated with the feature extracted from the analysed statement must be inserted in the required subtree reference nodes of the main tree item. Thus the node referencing the subtree S (node 8) contains also a 1,and the reference to subtree L (node 10) contains a 2 (see APP.III,i for node structure details).

During the matching of,say,the statement 'READ(5,99) ANAME',the symbols '99' are extracted as the value of parameter 1 and 'ANAME' as the value of parameter 2.On successful completion of the analysis at the terminal node 11,control is passed to its successor node 12.

The node is a pattern node and its strategy code (which is 3) directs the immediate symbol 'R' to be output and passes control to its successor,node 13.Similarly the immediate symbols 'EAD(6,' are output and control is passed to the node 19.

The strategy code of node 19 (which is 4) directs a copy of the parameter number 1 to be output,that is '99',and control is passed to its successor,node 20.

Node 20 outputs the immediate symbol ')' and transfers control to node 21,which directs a copy of parameter number 2 to be output,that is 'ANAME'.

Node 20 has no successor, so total successful processing of the statement 'READ(5,99)ANAME' is reported and the statement output is 'READ(6,99)ANAME'.

The power of this system as has been emphasized above(3.32) is due to the ease with which the modifications can be altered by simply altering the patterns associated with particular items.

As an example,if the pattern is deleted from the item in figure IX and the pattern;

---I—N—P—U—T—P1— ,—P2

added,then the previous input statement 'READ(5,99)ANAME' would be translated to 'INPUT99,ANAME' which might be the corresponding statement in another language.

If we then substitute;

-----R—E—A—D—P2

for the previous pattern,the FORTRAN statement 'READ(5,99) ANAME' would be translated to the BASIC statement 'READ ANAME'.

The pattern technique supplies the means by which we can output 'correct' statements after only approximately matching the incoming statement,and also the means by which the analyser can process trivially incorrect statements and output the 'corrected' form with no loss in efficiency(4.21).

4.32 The usage frequency statistics.

From the results of prolonged analysis,punctuated by periodic restructuring,the experience tree will come to mirror the structure of the language statements processed. The information contained within the experience tree extends down to the level of occurrence frequencies for every feature in the language.

The strategy tree will contain information relating to a different facet of the processed statements, that is, the frequencies of the different types of error.

Thus the total system,experience and strategy trees,will have available for possible further use,extensive statistics concerning the structure of all the previously analysed statements.In the context of our particular investigation, the processing of the normal FORTRAN workload of a computer (see chapter 5 for details),these usage frequency statistics will reflect the 'normal' usage of the language FORTRAN.

5.0 Analysis and critical assessment of system performance.

Within this chapter we present detailed results of our system, checking and when necessary, attempting to correct statements written in the language FORTRAN. The statements processed comprise programs which were drawn from five different enviroments; the five data sets are described below (5.01).

We then describe the results of the analysis of each of the data sets, drawing attention to general and then particular features of stability and unstability (5.11). The second part of this section (i.e. sub-section 5.12) contains an assessment of the utility of the optimising techniques. The effects of introducing allowable error are described in 5.13.

In section 5.2 we describe the frequency and types of error found in the analysed data sets and the results of our methods of dealing with them.

Finally we give an overview and summary of the results obtained (5.3).

5.01 The raw data for analysis.

The raw data analysed is composed of the normal FORTRAN workload of computers within five different enviroments. The first two sets of data were collected in Imperial College, from the normal workload of the same computer (an IBM 7094), but at different times. The other three sets of data (kindly supplied by ICL) are; from Oxford University; a mixture of jobs from CERN and the SRC Atlas Laboratories; and programs collected from various sources within ICL. Thus the data analysed can be divided and described as follows;

Data Set I - typical student jobs from Imperial College, 1969,

Data Set II - novice programmers at Imperial College, 1972,

Data Set III - typical student jobs from Oxford University,

Data Set IV - a mixture of research programs

Data Set V - commercial programs from ICL.

5.10 The results of analysis and appraisal of optimising techniques.

Within this section we describe and reference in APPENDIX VIII the full results of the analysis of the five,above described (5.01),data sets.The optimisation techniques are then evaluated with respect to this analysis.In sub-section 5.14 the practical utility of allowable error techniques is assessed using the same five data sets.

5.11 The analytical results and the pattern of FORTRAN usage.

We divide our presentation of the analytical results into three sub-sections.First we consider in detail the pattern of FORTRAN usage within one enviroment using Data Set I (5.111).

Second,we describe,in general terms,the results of the separate analysis of all five sets of data(5.01 above) and make general comparisons as to the nature of the usage pattern (5.112).

Third we make specific inter-enviromental comparisons at the level of particular subtree usage (5.113).

5.111 The detailed analysis of data within a single enviroment.

The experience tree illustrated in figure I,APPENDIX VIII, was used to analyse Data Set I.At first the experience tree was restructured (using the branch swopping mechanism,4.121, as all other references to 'restructuring' within sub-section 5.11) after every 100 statements processed,then after 600 statements had been processed the interval was extended to 300 statements until the total batch of 10,690 punched cards had

been processed.This was equivalent to 8,164 statements.

The degree of restructuring required to produce the optimum experience tree (in terms of minimising the average search length) is measured in terms of the percentage of nodes that are members of filial sets with more than one node,reordered.

The optimisation of the experience tree is measured as the saving in search length (4.01),exhibited by the actual language features matched,over a random distribution of features with the same experience tree structure.

The results of these measurements are tabulated in figure I and illustrated graphically in figure II.The graph shows how,after the initial drastic changes in the above described quantities (42% of nodes reordered:-46 to 72% change in search length saving) they settle to constant,low values after 4000 statements have been analysed.

The main feature that suggests that the pattern of FORTRAN usage within this enviroment will be particularly amenable to our optimisation techniques,is that not only does the degree of restructuring required for optimisation fall to a low value but also that the extra optimisation produced is very small.Thus after processing a few thousand statements the experience tree structure remains very close to the optimum during the processing of the remainder of the data set.

The other important feature illustrated in figure II is that the fairly optimum and stable structure exhibits a considerable saving in search length (nearly 60%).

The small amount of instability in the experience tree,as evidenced by the continual but small percentage of restructuring,is due to features such as the subtree containing all the alphabetic characters used in program variable names.This

Fig. I - detailed results from the analysis of Data Set I

Total number of cards processed	148	270	400	533	669	804	1220	1540	1928	2294	2674	3001	3396	3813	4196
Total number of statements processed	100	200	300	400	500	600	900	1094	1394	1694	2000	2300	2600	2900	3200
Number of statements with more than 199 chars.	4	4	4	4	4	4	4	8	10	10	13	13	16	18	18
Average statement length	17.96	19.25	19.91	20.05	20.96	21.60	22.15	22.23	22.60	22.85	22.02	21.38	21.65	22.03	21.82
Percentage of nodes reordered	41.7	10.33	6.27	7.38	3.32	1.85	10.0	3.7	14.0	14.0	10.5	10.9	3.7	4.5	7.5
Percentage/100 statement interval	41.7	10.33	6.27	7.38	3.32	1.85	3.33	1.91	4.67	4.67	3.43	3.63	1.23	1.50	2.50
Resulting change in oddity/item	1.76	.025	.012	.008	.0016	.0010	-	-	-	-	-	-	-	.0051	.0065
Resulting change in search length	118.38	0.83	0.27	0.27	0.07	0.16	0.00	0.00	0.1	0.9	-	-	-	-	-
Saving in search length in percent	72.36	74.18	74.53	73.89	73.85	73.48	72.6	72.5	68.0	63.9	61.7	59.8	58.7	57.6	57.3

Fig. I continued

4543	4867	5216	5579	5928	6287	6707	7320	7695	8032	8332	8791	9191	9848	10,191	10,498	10,690
3500	3800	4100	4400	4700	5000	5300	5600	5900	6200	6500	6800	7100	7400	7700	8000	8164
18	18	18	18	18	18	18	25	31	35	35	41	42	43	48	48	48
21.59	21.48	21.34	21.00	20.78	20.56	20.42	20.48	20.68	20.90	20.48	20.23	20.07	19.81	19.57	19.50	19.50
6.7	11.03	4.98	4.98	2.13	3.56	1.42	3.56	1.78	4.98	2.85	2.13	3.56	2.49	2.14	1.84	0.37
2.23	3.68	1.66	1.66	0.71	1.19	0.47	1.19	0.59	1.66	0.95	0.71	1.19	0.83	0.71	0.61	0.23
.0059	.0196	.0049	.0065	.0029	.0020	.0007	.0004	.0004	.0035	.0025	.0006	.0016	.0007	.0012	.0013	.00001
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
57.4	58.8	59.0	59.0	59.2	59.5	59.3	59.0	58.8	58.0	57.61	57.60	57.46	57.34	57.28	57.17	57.07

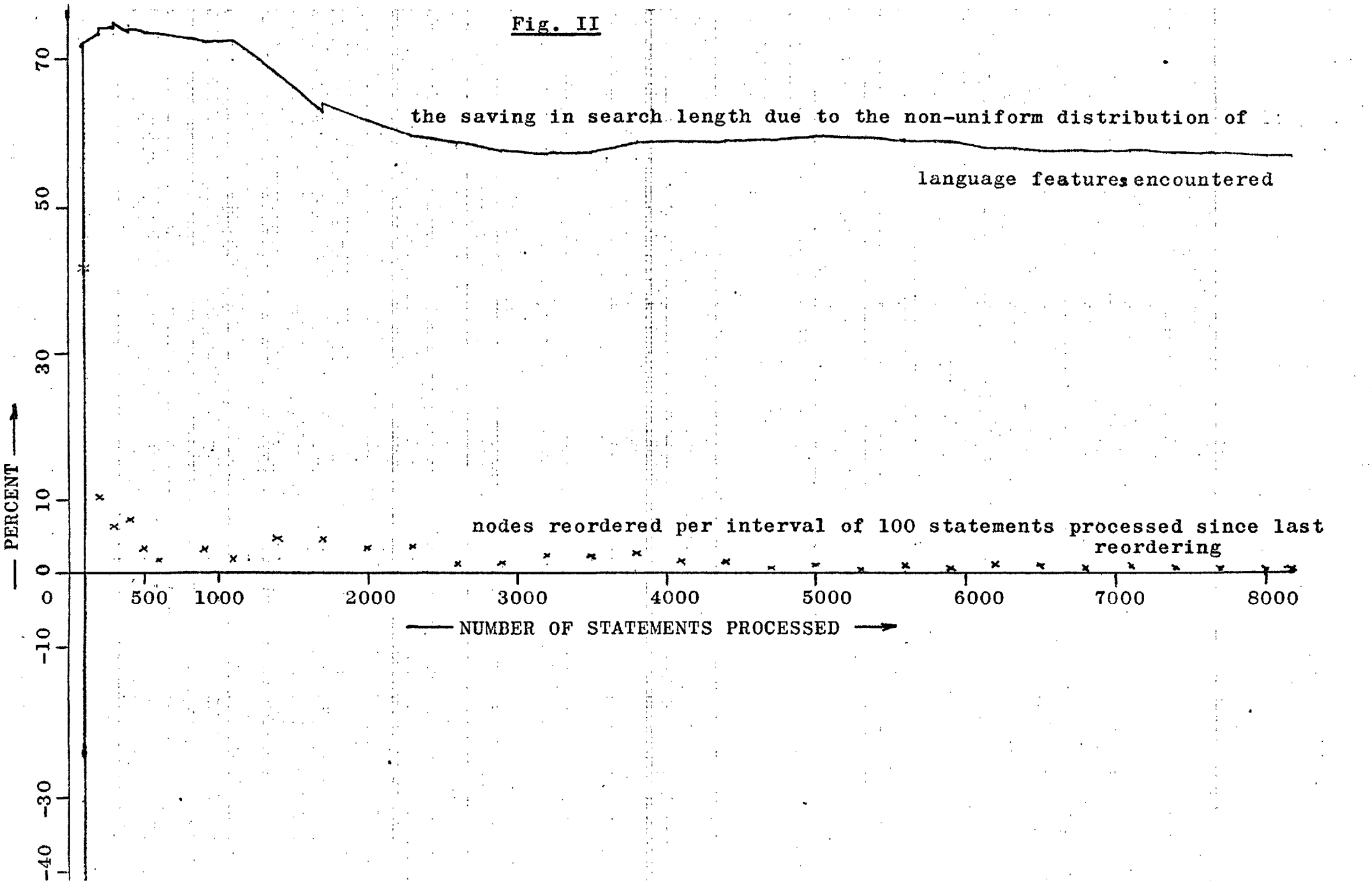


Fig. II
the saving in search length due to the non-uniform distribution of language features encountered
nodes reordered per interval of 100 statements processed since last reordering
PERCENT
70
50
30
10
0
-10
-30
-40
0
500
1000
2000
3000
4000
5000
6000
7000
8000
NUMBER OF STATEMENTS PROCESSED

subtree,as expected,constantly exhibits changes in the relative frequency with which all but the few highest precedence characters,are accessed.

The final,optimised experience tree resulting from the analysis of Data Set I is illustrated in figure II,APPENDIX VIII.

5.112 A general comparison of the analytical information obtained from five different enviroments.

Each of the remaining four Data Sets (Data Sets II,III,IV and V,5.01) was processed as follows.

The final experience tree from the above described analysis of Data Set I was first optimised using the branch swopping mechanism (4.121).Then the confidence levels were set throughout the experience tree (4.13),the frequency counts zeroed and slight alterations made to several items.The resulting experience tree is illustrated in figure III, APPENDIX VIII.

Each Data Set was then analysed separately with the above described experience tree and no restructuring.

Thus from the average saving in search length found for each of the Data Sets we have a general measure inter-enviromental stability.Then by optimising each of these final experience trees (using the branch swopping mechanism) we obtain a measure of how much each particular Data Set differs from Data Set I.

The results of each analysis are presented below, preceded by the comparable results obtained from the initial analysis of Data Set I (description and detailed results of this analysis are given earlier, 5.111 and figures I and II).

Data Set I 90 to 100 jobs

10,690 cards processed, containing 8,212 statements
of which 8,164 were successfully analysed,
(48 were longer than 199 characters)

3% of FORTRAN cards were continuation cards,
10% of statements were 'Comments',

the average statement length was 19.50 characters including blanks and
18.20 with the non-significant (non Hollerith) blanks removed.

48% of statements were assignments.
Total set exhibited a 57.1% saving in search length over a random distribution.

Then the following four analyses were carried out with Data Sets II, III, IV and V using an experience tree optimised with the above Data Set.

Data Set II 61 jobs

6326 cards processed, containing 3958 FORTRAN statements
of which 3730 were successfully analysed (46 longer than 199 chars.)
6% of FORTRAN cards were continuation cards,
23% of statements were 'Comments'
average statement length, 22.1 including blanks and 19.8 without

24% of statements were assignments.

Total set exhibited a 58.3% saving in search length over a random distribution,
30.3% of nodes were reordered,saving 0.224 oddity/item,
resulting optimised tree had 60.4% saving in search length.

Data Set III 19 jobs

5008 cards processed,containing 4410 FORTRAN statements
of which 425 7 were successfully analysed (87>199 chars.)
8% of FORTRAN cards were continuation cards,
4% of statements were 'Comments'
average statement length,19.0 including blanks and
18.2 with non-significant blanks removed.

51% of statements were assignments.

Total set exhibited a 50.0% saving in search length over a random distribution,
29.3% of nodes were reordered,saving 0.212 oddity/item,
resulting optimised tree had 52.4% saving in search length.

Data Set IV 19 jobs

3374 cards processed,containing 3058 FORTRAN statements,
of which 2952 were successfully analysed (38>199 chars.)
4% of FORTRAN cards were continuation cards,
6% of statements were 'Comments'.
average statement length,17.4 including blanks and
16.4 with non-significant blanks removed.

55% of statements were assignments.

Total set exhibited a 52.2% saving in search length over a random distribution,
21.6% of nodes were reordered,saving 0.179 oddity/item
resulting optimised tree had a 53.7% saving in search length.

Data Set V 12 jobs.

1128 cards processed,containing 1055 FORTRAN statements
of which 1118 were successfully analysed (12>199 chars.)
4.5% of FORTRAN cards were continuation cards,
2% of statements were 'Comments'
average statement length,15.4 including blanks and
14.9 with non-significant blanks removed.
49% of statements were assignments.

Total set exhibited a 49.1% saving in search length over a random distribution,
20.2% of nodes were reordered,saving 0.312 oddity/item,
resulting optimised tree had a 52.0% saving in search length.

One constant feature of program structure is that about 50% of statements are assignments (note: also in Data Set I,48%). The Data Set II does not exhibit this feature (only 24%),and differs from the others in several other respects,for instance, the incredibly high percentage of 'Comment' statements,23%! This division of program statements,half and half,between assignment and non-assignment statements was also found by KNUTH (1970).

The important point illustrated by the above results is that the experience tree optimised on Data Set I,still produced a search length saving of about 50% with each of the other

Data Sets, before any optimisation had been carried out. Thus the pattern of usage within different enviroments is sufficiently similar, with respect to our methods of analysis, for one experience tree structure to produce the same large saving in search length in different enviroments with no further restructuring.

When the final experience trees from each enviroment were optimised, a 20 to 30% reordering of nodes was required in all cases. This fairly large degree of restructuring produces only a small relative gain in search length saving (a few percent in each case). The reason for this is that a large portion of the nodes reordered are in subtrees in which the distribution of access frequencies approaches random and thus no two final examples of these subtrees are the same but the gain produced by optimisation is minimal. An example of such subtrees are the ones containing all alphabetic (subtree A, APP.VIII) and numeric (subtree N) characters which typically account for one third to a half of the nodes reordered in the complete experience tree.

The final and optimised experience tree for each Data Set is illustrated in APPENDIX VIII.

5.113 A comparison of particular aspects of inter-enviromental language usage.

We will divide the detailed discussion into two parts, first, non-assignment statements and second, assignment statements and arithmetic expressions. The full results from which we will select particular details are composed of the experienced trees illustrated in figures II, IV, V, VI and VII, in APPENDIX VIII.

The distribution of non-assignment statements is divided into 36 types and illustrated graphically in figures III, a, b, c, d and e, for the Data Sets I, II, III, IV and V, respectively.

Fig. III a - the frequency of occurrence of non-assignment statements in Data Set I

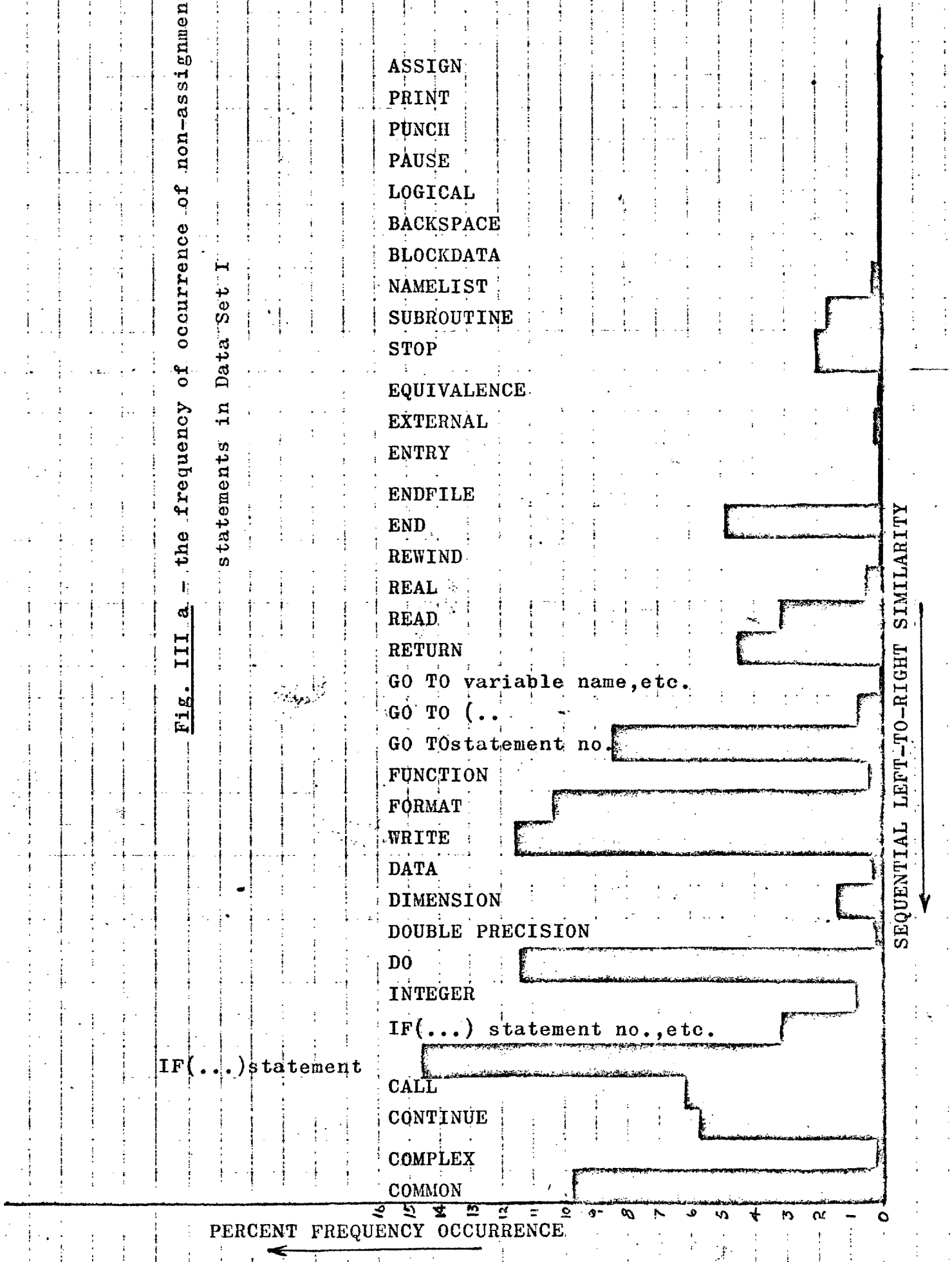


Fig. III b - the frequency of occurrence of non-assignment statements in Data Set II

Fig. III c = the frequency of occurrence of non-assignment statements in Data Set III

ASSIGN
PRINT
PUNCH
PAUSE
LOGICAL
BACKSPACE
BLOCKDATA
NAMELIST
SUBROUTINE
STOP
EQUIVALENCE
EXTERNAL
ENTRY
ENDFILE
END
REWIND
REAL
READ
RETURN
GO TO variable name,etc.
GO TO (..
GO TO statement no.
FUNCTION
FORMAT
WRITE
DATA
DIMENSION
DOUBLE PRECISION
DO
INTEGER
IF(...)statement no.,
IF(...)statement
CALL
CONTINUE
COMPLEX
COMMON

Fig. III d - the frequency of occurrence of non-assignment statements in Data Set IV

ASSIGN
PRINT
PUNCH
PAUSE
LOGICAL
BACKSPACE
BLOCKDATA
NAMELIST
SUBROUTINE
STOP
EQUIVALENCE
EXTERNAL
ENTRY
ENDFILE
END
REWIND
REAL
READ
RETURN
GO TO variable name, etc.
GO TO (..
GO TO statement no.
FUNCTION
FORMAT
WRITE
DATA
DIMENSION
DOUBLE PRECISION
DO
INTEGER
IF(..)statement no.,
IF(...) statement
CALL
CONTINUE
COMPLEX
COMMON

DISSIMILARITY

20 18 16 14 12 10 9 8 7 6 5 4 3 2 1 0

PERCENT FREQUENCY OCCURRENCE

Fig. III e - the frequency of occurrence of non-assignment statements in Data Set V

ASSIGN
PRINT
PUNCH
PAUSE
LOGICAL
BACKSPACE
BLOCKDATA
NAMELIST
SUBROUTINE
STOP
EQUIVALENCE
EXTERNAL
ENTRY
ENDFILE
END
REWIND
REAL
READ
RETURN
GO TO variable name,etc.
GO TO (..
GO TO statement no.
FUNCTION
FORMAT
WRITE
DATA
DIMENSION
DOUBLE PRECISION
DO
INTEGER
IF(...) statement no.,etc.
IF(...) statement no.
CALL
CONTINUE
COMPLEX
COMMON

24 22 20 18 16 14 12 10 9 8 7 6 5 4 3 2 1 0

PERCENT FREQUENCY OCCURRENCE

DISSIMILARITY

The statements are ordered in terms of their sequential, beginning-to-end similarity,as in the tree F of figure II, APPENDIX VIII.

The non-uniformity of distribution is very noticeable, and furthermore the higher frequency statements are fairly well spread out.This means that the analysed programs were composed of only a small number of quite different statements.

The individual statement frequencies generally follow those reported by KNUTH (1970).The noticeable exception is Data Set II,which has a very large percentage of 'CALL' statements and a very low percentage of 'DO' statements;and has many more statement types accessed at least once than any of the other Data Sets.

In all cases the 'IF' statement is the most frequently occurring (approx. 20%),but the constant total is composed of widely different proportions of the logical 'IF' statement (i.e. IF(...)statement) and the arithmetic 'IF' statement (i.e. IF(...)S1,S2,S3).Data Set I,heavily favours the former type,whilst Sets III,IV and V heavily favour the latter,and Set II exhibits approximately equal percentages.The almost exclusive usage of logical 'IF' statements found in Data Set I is probably a direct result of the elementary FORTRAN teaching at Imperial College which omits the arithmetic 'IF' statement.The non-conformity of Data Set II is discussed later (5.3),but we note that the programmers were indirectly taught by means of the above mentioned scheme.

To consider more detailed features of non-assignment statement usage;excluding Data Set II,60 to 70% of variable lists contain no more than three variables.

A result exactly the same as that found by Knuth is that 95% of 'DO' statements utilise the default option of one for

the index increment.

Of the variables that can be subscripted,an average of only 30% are:20% with one subscript and 10% with two.The occurrence of variables with more than two subscripts is negligible.An average of 95% of subscripts are simply a constant or a variable.

If we consider subtrees in which the individual items have a more uniform distribution,we can distinguish two types.The stable ones such as the subtrees,U (for variable names) and S (for statement numbers),and the unstable ones such as the subtrees,A (for alphabetic characters in variable names) and N (contains the ten digits that compose constants and variable names).

The subtree S was only restructured after the analysis of Data Set III,and that was to give priority to the relatively very high frequency of occurrence of two-digit statement numbers (56%).In all other cases there was no restructuring of the subtree S as the original tree was optimum and the two most frequently occurring statement numbers always consisted of two and three digits.

The subtree U was not restructured after any of the analyses following the optimisation with Data Set I.Variable names consisting of one and next,two characters were always the most frequent (except Data Set II in which the order was one and then five characters).

The subtree N exhibits an order of precedence for the usage of digits of,1,0,2 and then 3 (except Set II) while the order of the remainder is different in each case.

The most unstable subtree of all,is subtree A;the immediate symbol I has the highest precedence in all but Data Set III (in which it is A),usually followed by A,then N (but T is

second in Set II and third in IV).The order of the remaining large majority of this subtree is a highly variable quantity.

Of all possible arithmetic expressions,90% or more are simply a single operand,or two operands separated by an operator;which again appear to be very similar to Knuth's results.

One half of operands are simple variable names,whilst a quarter are constants and only 2 to 4% are bracketed expressions.The usage of operators is fairly uniform and changeable, excepting the exponentiation operator which occurs in only a few percent of instances.

5.12 Optimisation of the analysis process.

Within this sub-section we assess the practical utility of the two optimising techniques described earlier (1.12 and 4.12 for details).

5.121 The branch swopping mechanism.

The analysis of the five Data Sets(5.01) illustrates the amenability of the usage pattern to the branch swopping mechanism (4.121).The average saving in search length over a random distribution of language features is consistently 50% or more.

Furthermore,the patterns found in completely different enviroments are sufficiently similar for an optimisation within one enviroment to extend to other enviroments without much loss in processing efficiency.(5.112)

The non-uniform distribution that enables the branch swopping technique to effect a large saving in search length,also means that a rigid form of analysis could produce a considerable loss in search length in comparison with a random distribution of items accessed. For example,if we 'invert' (i.e. reverse the order of all nodes in each filial set) the optimised experience trees we find the saving in search length to average

-37%.Thus the saving in search length produced by the branch swopping mechanism averages 90% or more in comparison with the worst rigid analysis of the actual distributions of language feature usage encountered.

Another way to express this saving is in terms of the number of wrong nodes examined during the analysis of a statement. The saving over a 'worst' analysis is on average 3.8 wrong nodes per tree item matched,and knowing that the average statement involves matching nearly 24 tree items,the branch swopping mechanism saves the searching of over 90 nodes per statement analysed. The extra optimisation produced by continual restructuring is of the order of 0.1 nodes within one enviroment (see figure I),and only 4 nodes from one enviroment to another (see Data Set results in 5.112).

In terms of actual analysis times,which must include all the non-optimised system overheads (mentioned earlier 3.3), the saving *in time* due to the branch swopping mechanism is still very large.For example,in the analysis of 100 statements 'randomly' selected from Data Set II,we obtain a 30% saving in analysis time when using the experience tree optimised with Data Set I (fig.III,APP.VIII) as opposed to the 'worst' experience tree for Data Set II (i.e. 'inverted' fig.IV,APP.VIII).Whilst the extra optimisation dictated by the analysis of Data Set II,produced a negligible decrease in analysis time over that produced with the experience tree already optimised by Data Set I.

5.122 Use of the promotion mechanism to increase efficiency of analysis at the cost of increasing the size of the total tree hierarchy.

The optimisation of the analysis process by means of the promotion mechanism results from two effects.One,the reduct-

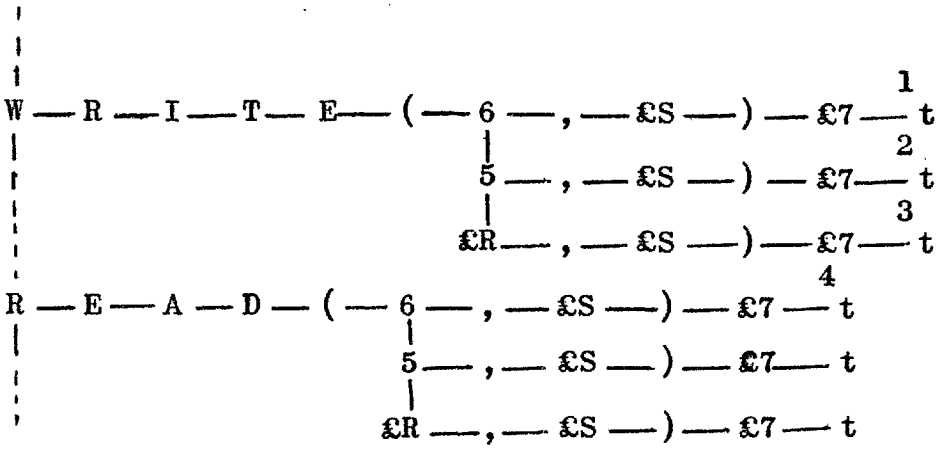
ion in the number of subtree references required in the analysis of some types of statement;two,reduction in the number of wrong nodes examined owing to the specificity of the "promotions' (i,e. the items promoted out of a subtree).

The most stringent criteria for 'promotion' used were that 25% or less of a subtree's items have been accessed on 90% or more of the references to the subtree.With these constraints the only items promoted were '5' and '6' (the standard input and output device numbers) from the subtree R.These two items were both automatically placed within the 'READ' and 'WRITE' statements in the main tree F,in figure II,APP.VIII.

When the same promotion criteria were applied to the experience tree resulting from the analysis of Data Set II (fig. IV,APP.VIII),the same two items were both promoted but to <u>five</u> places within the tree F.This was because there are three more references to subtree R (apart from the 'READ' and 'WRITE' statements) within tree F.Of these five references only two (the 'READ' and 'WRITE' statement ones) are utilised in the analysis of Data Set I,but all five are accessed at least once in the analysis of Data Set II.

The structure of the input and output statement items within tree F,after the promotion of items '5' and '6' is illustrated in figure IV.

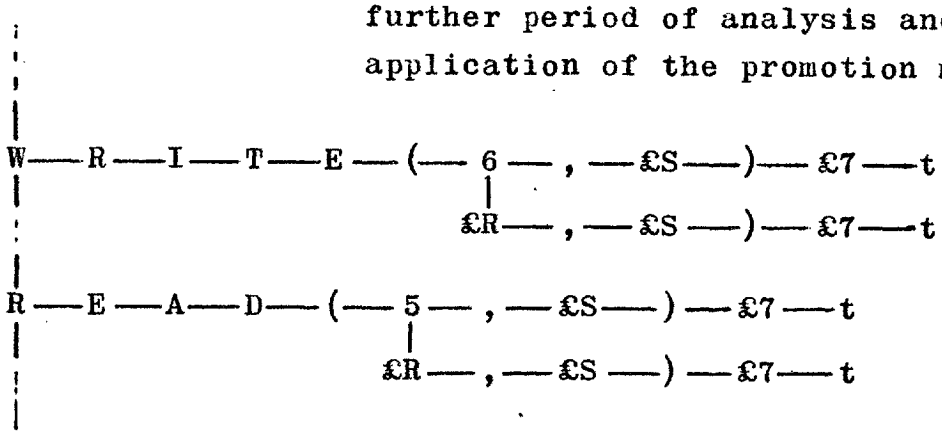
<u>Fig. IV</u> - the promotion of items '6' and '5' from subtree R to tree F.



The promotion to the two positions illustrated (fig,IV) is achieved at the cost of 50 nodes needed to construct the two promoted sub-structures (for the structure succeeding the promoted items is more complex than illustrated above,to accommodate the other possible versions of the input and output statements),or a 3% increase in the total experience tree size.With the above structure the gain in analysis time results simply from the avoidance of one subtree reference for 90% of input and output statements (the average statement analysis includes 23 to 25 subtree references).

If the process of analysis is continued the frequency counts within the terminal nodes 2 and 4 (fig.IV) will indicate that these two particular items,'WRITE(5,etc.' and 'READ(6,etc.', are not accessed.Then during a second application of the promotion mechanism these two items will be deleted because they are redundant (for both items,'5' and '6' are still maintained within the subtree R,see 4.122,which is referenced by an alternative node) and not used.The resulting structure is illustrated below (fig.V).

Fig. V - the promoted items of fig.IV after a further period of analysis and a second application of the promotion mechanism



Now the process of analysis is further optimised when matching input statements because the item 'READ(6,etc.' does not have

to be examined before the usual 'READ(5,etc.' type of input statement is matched.This gain would also be realised by the use of the branch swopping mechanism after some period of analysis following the first application of the promotion mechanism.But there is another important feature of the second application of the promotion mechanism which is that the total experience tree size will be reduced (because of the deleted items).In the above case the reduction amounts to 25 nodes.

Thus the overall increase in the size of the experience tree to automatically effect the specific optimisation of statements using the standard input and output device numbers (i.e. '5' and '6') is only 1.5%.In terms of the actual time saved,input and output statements are analysed 10 to 15% faster after the above promotions and these statements represent 12 to 15% of program statements.

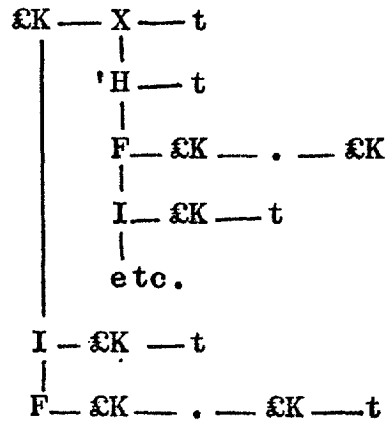
If the criteria for promotion are relaxed,so that promotion occurs if one third of a subtree's items have been accessed in at least 90% of references to the subtree.Then we find two separate promotions in the experience tree resulting from the analysis of Data Set II (fig.IV,APP.VIII).

First,the one described above,and second,promotion of the item '£P' (a reference to subtree P),from subtree M into seven places in subtree E.This promotion exploits the discovery that very few operands in arithmetic expressions have a prefix operator.A considerably reduced version of this promotion has been described and illustrated earlier (4.122),and it proves to be a very expensive process (for the size of subtree E is nearly doubled at every one of the seven insertions of '£P', by the size of the structure that has to succeed the inserted item).The complete promotion very nearly <u>doubles</u> the size of

the experience tree.The promotion saves on average a little more than one subtree reference per two statements analysed, and the saving in time per arithmetic statement is typically less than 10%.

When the promotion mechanism,with very similar criteria was applied to the experience tree resulting from the analysis of Data Set I (fig.II,APP.VIII),both of the above promotions were found and one extra one.This third promotion consisted of the highest precedence 'FORMAT' statement specifications from subtree W to two positions in subtree Q.The promoted items were;all the 'repeated' specifications and the two single specifications 'I' and 'F'.They are illustrated below in figure VI.

Fig. VI - the 'FORMAT' statement specifications promoted from subtree W to two positions in subtree Q of fig.II APP.VIII.



note:the node 'H contains the immediate symbol H,and the strategy code 6,APP.III

Unfortunately,

owing to the lack of sufficient storage space (see LAS,APP.I) as a result of the very large number of nodes required for the three promotions,the analyser was unable to function with the promoted experience tree.

Thus apart from the significant gain achieved by the auto-

matic exploitation of the standard input and output device numbers,the promotion mechanism has little to add to the efficiency displayed by the experience tree hierarchy,as initially organised.*Nevertheless* it does illuminate the places where a human operator might usefully consider more drastic reorganisation in the interests of processing efficiency.

5.13 An example of confidence levels within the experience tree.

Confidence levels as dictated by the frequency counts resulting from the analysis of Data Set I,were inserted in the experience tree,as illustrated in fig.III.APP.VIII.The actual values set within each node of every filial set with more than one member,are frequencies relative to 10,000.Thus a confidence level of 50% is illustrated as the value 5,000.

5.14 Non-exhaustive analysis and the effective reduction in size of the total tree hierarchy.

The introduction of allowable error as described earlier (4.14),is effected by setting a parameter within the general strategy which causes the matching process when utilising this particular general strategy to ignore alternative items within the experience tree which have confidence levels below a certain value;that is,the value of allowable error.Thus if certain items within the experience tree are ignored the optimisation of the analysis process is a result of the non-exhaustive analysis of the experience tree;the tree becomes in effect,more compact.

The practical utility of this procedure depends critically upon the distribution of language features encountered and decreases as this distribution approaches random.The most striking reduction in the effective size of the experience tree occurs when the minimum allowable error is introduced,

that is,less than 0.01% (neglecting alternative items with confidence levels of less than 1 in 10,000).In this event the experience trees illustrated in figs.II,IV,V,VI and VII of APPENDIX VIII are reduced by 25% (Data Set II) to 50% (Data Set V).The reduction in experience tree size continues with increasing allowable error to between 47 to 55%,when the allowable error is less than 1%.The most rapidly decreasing experience tree is that resulting from Data Set II,which is reduced by 25% with an allowable error of 0.01% and 50% when the allowable error reaches 1%.The reductions in experience tree size with allowable error increasing from zero to 1% are tabulated in figure VII.

Fig. VII - the reduction in effective size of the experience tree with increasing allowable error.

Data Set	I	II	III	IV	V
experience tree	fig.II	IV	V	VI	VII in APP.VIII
allowable error					
0.0	0.0%	0.0%	0.0%	0.0%	0.0%
0.01%	32.41	25.40	27.85	36.31	48.64
0.02%	32.89	25.40	27.85	36.31	48.64
0.03%	34.61	25.49	28.13	36.31	48.64
0.04%	35.00	29.54	28.69	36.31	48.64
0.05%	35.00	29.54	32.93	36.31	48.64
0.06%	37.97	29.82	32.93	36.41	48.64
0.07%	38.35	29.82	32.93	36.41	48.64
0.08%	38.35	32.93	32.93	38.00	48.64
0.09%	39.60	33.30	32.93	38.01	49.12
0.1%	39.60	34.00	32.93	38.19	49.12
0.2%	45.93	38.95	37.06	41.30	51.83
0.5%	50.05	45.44	40.92	44.87	52.96
1.0%	53.98	51.27	47.51	50.52	54.94

But,although the effective size of the experience tree, which represents the possible different interpretations of an incoming statement,is drastically reduced by the introduction of allowable error,we would not expect a similar reduction in analysis times.For our method of analysis is designed to examine the more likely alternatives first and thus on most occasions a match is found before the unlikely alternatives (which the allowable error technique eliminates) are examined anyway.

Thus after the analysis of 100 statements 'randomly' selected from Data Set II,the optimised experience tree (fig. IV,APP.VIII) produced a 1 to 2% gain in time when the allowable error was increased from zero to 0.01%.The individual savings in analysis time ranged from 11% for an 'INTEGER' statement composed of a list of five subscripted variables,to nothing for the keyword statements such as 'STOP' and 'CONTINUE'.

The most effective use of allowable error techniques occurs within the tolerant processing aspect of our system (5.20,below).

5.20 Improvement in the processing of incorrect statements.

Within this section we consider trivial and non-trivial errors and the effectiveness of the confidence jump mechanism (4.21) and,the strategy tree and force matching (4.22) which combine as the tolerant processing facet of our analyser.

We also examine the frequency of occurrence of syntactic errors in 'typical' FORTRAN programs.Further,by the introduction of allowable error,both processes are optimised in terms of faster processing and 'better' corrections.This process even extends to the 'correction' of syntactically

correct but wrong statements (that is,statements which are not 'meant' but just happen to be syntactically correct).

5.21 The exploitation of structural and statistical redundancy in the processing of trivial errors.

The analysis (and hence the definition) of 'trivial errors' which proceeds with no loss in efficiency in comparison to the analysis of correct statements is simply a result of the confidence jump mechanism.The confidence jump mechanism is initiated and controlled by the jump nodes which are automatically inserted in the experience tree (see 4.022).Thus the question of trivial errors hinges upon the structure and position of the jump nodes within the experience tree. The approximate matching produced by the confidence jump mechanism also increases the efficiency of analysis of correct statements.Typically the decrease in analysis time is only 1% which results from the insertion of 30 jump nodes in tree F which avoid matching 110 or 10% of the total experience tree.

Trivial errors processed when the jump nodes were inserted with no allowable error (i.e. exploits only structural redundancy),for example,statements with the misspelt keywords 'DIMENTION' and 'FORMIT'.If the jump nodes were inserted with an allowable error of less than 0.01% (items that have confidence levels of 1 in 10,000 are ignored) to exploit the statistical redundancy in the main tree F;then in the experience tree resulting from the analysis of Data Set I,one third of the statement types (that is,12) are effectively removed from the language,as a first approximation.As with the allowable error effects (see fig.VII,5.14 above) the saving 'tails off' rapidly with increasing allowable error.(see fig.VIII)

Fig. VIII - decrease in the number of valid statement types with increasing exploitation of statistical redundanc

degree of allowable error	0	0.01%	0.05%	0.1%	1.0%	5.0%
	tree F from Data Set I,fig.II,APP.VIII					
no. of statement types valid	36	24	23	21	15	10
no. of jump nodes inserted	32	17	16	16	13	8
	tree F from Data Set II,fig.IV,APP.VIII					
	36	31	27	25	16	11
	30	26	22	20	13	8

5.22 The processing of non-trivial errors.

Within this sub-section we present in detail the types of non-trivial error encountered during the analysis of the five Data Sets (5.01).We then introduce and describe 'non-storage processing' which was instigated as a result of the error statistics.Finally we assess the practical utility of the optimisation of our strategy tree (4.22) and the force matching process.

5.221 The nature and frequency of occurrence of non-trivial errors.

The frequency of occurrence of non-trivial errors was consistently of the order of 0.3%,that is,three incorrect statements in every 1000.This figure appears to be,surprisingly low until it is recast as,one incorrect statement per three jobs.If we couple this result with the fact that most computer

systems will reject any job containing an incorrect statement,then our original statistics amount to the result that one in three computer jobs are rejected.

The full details of the errors encountered whilst analysing Data Sets I to V,with individual error summaries are given in APPENDIX IX.An overall error summary is presented below (fig.IX),it is only intended as a general guide as it is often a debatable point as to whether a statement is incorrect or not,and what the correct form might be,without a detailed knowledge of the enviroments from whence the original data came.

Fig.IX - error summary for Data Sets I to V.

	missing	inserted	wrong
1 character	37	14	20
2 characters	0	0	1
3 characters	1	0	0

There were no examples of characters transposed and only three of the 'wrong character' errors were shift key mistakes.

Nearly all of the errors fall into one of the three simple classes which most error correcting systems are designed to correct,that is,one character wrong,inserted or omitted (see discussion of previous work in 2.2).Nevertheless the other constraint usually adopted,which is that there is only one error per statement was found to be questionable.Data Set I exhibited 1.7 errors per statement and Data Set III,1.23 errors.

Most of the characters missing were commas and a few right parentheses.The most error-prone statement is the 'FORMAT' statement.

5.222 The introduction of 'non-storage' processing.

Our analyser has no a priori limitations upon the types of error it can successfully process,and thus control must be able to return to any previous position in the analysis of a statement to be able to reprocess it with a different general strategy.This back-tracking ability necessitates the storage of all the initial conditions of all the previous features matched within a statement until the total statement has been successfully analysed.Naturally this is a time and space consuming process,and is termed 'storage' processing.

Now when the very small incidence of incorrect statements became apparent after the analysis of Data Set I,it was clear that in the majority of cases all the stored data was not used.So in order to speed the processing of the majority of statements (that is,statements that at most contain only trivial errors),'non-storage' processing was introduced for the analysis of Data Sets II to V.

In non-storage processing,the initial conditions for the matching of all the previous features of the statement currently being analysed are not stored and thus the analysis becomes much faster.When the occasional incorrect statement is encountered the complete analysis is restarted with full storage processing.

Thus the majority of the statements are analysed faster at the expense of the occasional more-than-trivially incorrect statement,which of course now takes longer to analyse.The average gain in analysis time using non-storage processing is by a factor of seven.

There are of course many other possible positions of compromise in this situation.For instance,the analysis need not be restarted unless the general strategy to be applied requires 'back-tracking' to a previous position in the current analysis.

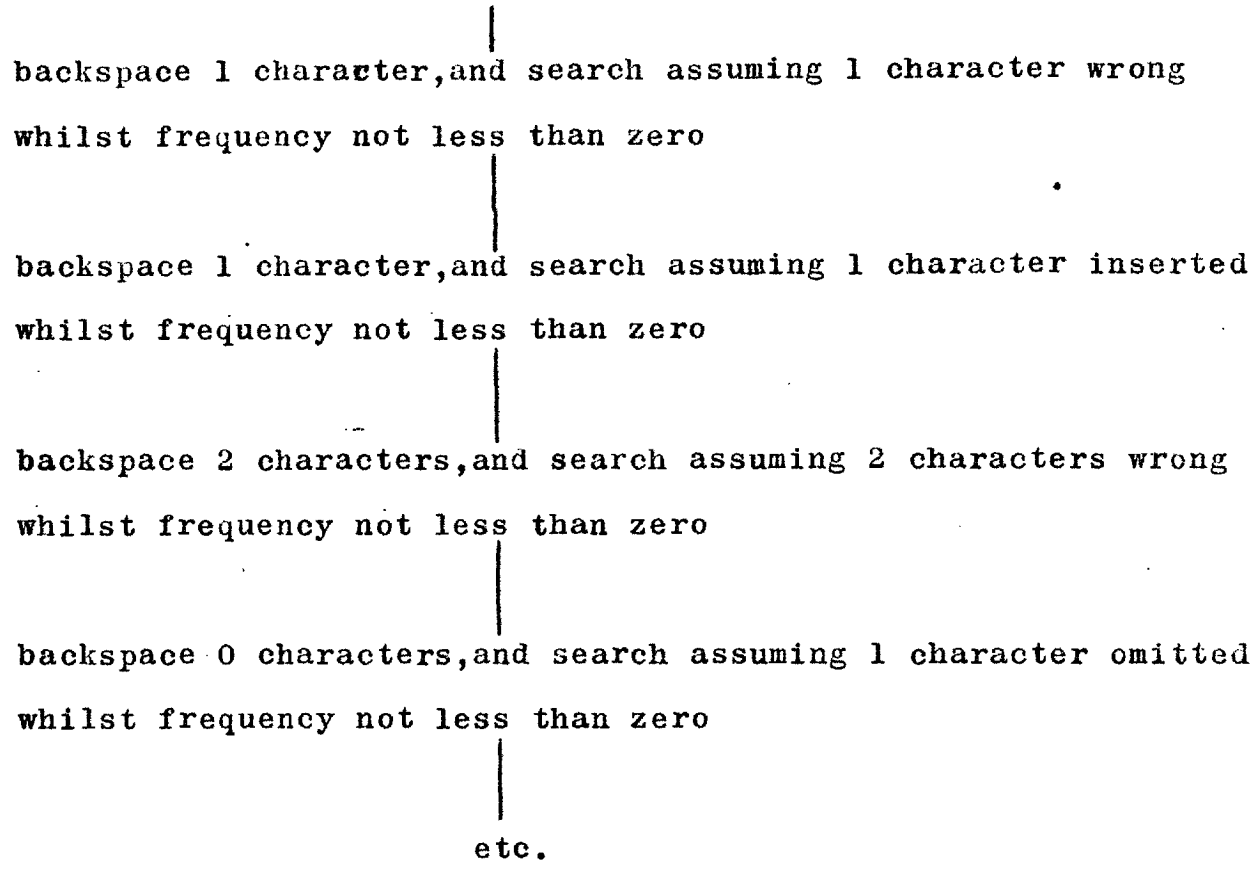
5.223 Restructuring the strategy tree and the utilisation of confidence levels in the restoration of non-trivial errors.

As described and exampled in detail earlier (4.22) statements containing non-trivial errors are analysed by utilising the general strategies,in order of precedence from the strategy tree,until a match is obtained;or a complete failure with the statement.

The 'random' strategy tree,illustrated below (fig.X),was utilised in the analysis of Data Set I and although 53% of the incorrect statements were restored,none of the restorations were 'correct'.All but one of the restorations were performed by the second general strategy (that is the first 'correcting' strategy,for the first strategy will match only correct statements and was given the highest precedence whilst the remainder were ordered 'randomly').This is because the errors occurred within subtrees which presented several alternatives which were equally likely (purely upon the basis of confidence levels).For instance,when a comma is omitted from between two variables the confidence levels within the subtree U, are not sufficiently dissimilar to dictate which of the possible restorations is 'correct'.Thus in each case the second general strategy is able to effect a syntactically correct structure.

Fig.X - the 'random' strategy tree.

backspace 0 characters,and search assuming 0 characters omitted whilst frequency not less than 0



Note: the first strategy is equivalent to 'obey the strategy codes' as described earlier (4.22),and the optimisation that can be produced by neglecting the low likelihood experience tree items is not being used,all the strategies search down to items with 'frequency' zero (that is, relative frequency or confidence level).

Thus the errors detected and restored 'incorrectly' by the above strategy tree during the analysis of Data Set I,were analysed by hand and the strategy tree was restructured to place the appropriate strategies in order of precedence. Furthermore,a simple linear function of the error frequencies (which are given in fig,Ib,APP.IX) were added to each of the strategies to direct the force matching to the more likely experience tree items.The restructured strategy tree (fig.XI, below) was utilised in the analysis of Data Sets II to V.The results (figs.II to V,APP.IX) illustrate a significant

proportion of 'correct' modifications,and strategies as far down as the tenth performed modifications.

A more direct comparison of the two strategy trees was made with the incorrect statements below (fig.XII),all of which were found within the analysed Data Sets.

Fig.XI - the strategy tree of fig.X restructured as a result of analysis of Data Set I.

backspace 0 characters,and search assuming 0 characters omitted
whilst frequency not less than zero

|

backspace 0 characters,and search assuming 1 character wrong
whilst frequency not less than 100

|

backspace 0 characters,and search assuming 1 character inserted
whilst frequency not less than 125

|

backspace 1 character,and search assuming 1 character omitted
whilst frequency not less than 143

|

backspace 0 characters,and search assuming 1 character omitted
whilst frequency not less than 166

|

backspace 2 characters,and search assuming 1 character omitted
whilst frequency not less than 200

|

etc.

Note: no allowable error introduced because the first strategy searches all of the experience tree;the full strategy tree is illustrated in figure VI,APPENDIX IX.

Fig.XII - the restoration ability of the two (figs. X and XI) strategy trees

incorrect statement	'random' strategy tree	optimised strategy tree
1) IFINISH=0	failed	IFINIS=0
2) FORMAT(1X,FK.2)	failed	FORMAT(1X,F1.2)
3) READ(5,21)	failed	failed
4) EDASH=1.*COS(M)*E)	failed	failed
5) WRITE(6,20	failed	WRITE(6,20)
6) FORMAT(4H NAME 12X/X)	failed	FORMAT(4H NAM 12X/1X
7) FORMAT(1X,F10.0/)	failed	FORMAT(1X,F10.1/)
8) FORMIT(X 22A6)	FORMAT(1X,2A6)	same
9) M=IFINISH/2	M=IFINI+H/2	M=IFINIS/2
10) DO 99 I=,M	failed	failed
11) K(J+1-2)=K(J)	failed	K(J+1,2)=K(J)
12) READ(5,24)KPLUS WAYB	..(..)KPLUSW,AYB	..(..)KPLUSW,YB
13) FORMAT(I3,X,F9.5)	failed	FORMAT(I3,/,F9.5)
14) WRITE(6,21),K(1)	failed	WRITE(6,21)IK(1)
15) WRITE(60,25)(J,K(J),J=1,M)	failed	WRITE(6,25)(J,K(J),J= etc.
16) K(1)=(KPLUS+3) 14	K(1)=(KPLUS+3)/4	same
17) 96(M .LT. 2)KPLUS=2	IF(M.LT.2)KPLUS=2	same
18) FORMAT(16H-FILE LENGTH I2/(5X 19A1))	failed	..../(5X,9A1))
19) GO TO(11,12) M	GOTO(11,12),M	same

The restructured strategy tree is clearly the'better' of the two,as there are thirteen failures with the 'random' tree and only three with the restructured tree.A specimen listing of the analyser's output which includes all of the above examples and the initial and final experience and strategy

tree structures,is in figure VI,APPENDIX IX.

Of the thirteen modifications produced by the optimised strategy tree,seven can be considered as the 'best' or 'correct' recoveries (i.e. numbers 1,5,6,9,15,17 and 19 in fig.XII);a further three would not effect the essential 'correctness' of the results from the program which contained them (i.e. 7, 13 and 18);finally only two of all the modifications would lead to compilation errors and thus prevent execution (i.e. 2 and 11).

When allowable error is introduced by causing the first strategy to only search items that have a certain level of confidence,no increase in the speed of processing the above statements is found until the degree of allowable error reaches 1%.Then the gain in analysis time over the same system functioning with no allowable error is about 5%.The modifications are different because the very unlikely alternatives are effectively deleted.For instance,with the restructured strategy tree and 1% allowable error the modification to the incorrect statement number 11,in fig.XII becomes 'K(J+192)=K(J)' which is a 'better' recovery.

Another feature of the use of confidence levels is that we can detect errors in the form of syntactically correct statements that were not 'meant'.These statements become apparent to the analyser when the analysis is directed by an optimised experience tree,for they match statistically redundant items within the tree.Seven such statements were detected within Data Set I.

For example,the result of a shift key mistake gave the syntactically correct error 'WRITE(0,2001)L3,N3',which was detected as soon as the minimum allowable error (0.01%) was introduced into the first strategy.Although it would be perhaps unwise to alter such statements they could be flagged as suspect.

5.30 Summary of the results.

The actual distribution of language features encountered is far from uniform over the range of possibilities,and thus the branch swopping mechanism (5.121) produces a significant reduction in the average search length during the analysis of program statements.Typically the saving is 50 to 60% of the average search length for a random distribution of language features.

Another important feature is that this non-uniform distribution remains stable not only within a particular enviroment (5.111),but also from one enviroment to another (5.112).Thus once the analysis has been optimised,by the action of the branch swopping mechanism upon the experience tree,it will remain close to the optimum without continual restructuring.

The promotion mechanism (5.122) produces a significant saving in analysis time for the 'standard' input and output statements at the cost of a 1.5% increase in the size of the basic language description (the experience tree).But the initial experience tree was so close to the optimum in terms of tree and subtree organisation,that this mechanism is too crude to automatically effect more subtle optimisations.

The distribution of language features appears to be amenable exploitation of allowable error techniques (5.14) but the potential savings are largely undermined by the effectiveness of the branch swopping mechanism.Nevertheless,the fact that the size of the basic language description can be reduced by a third at the cost of introducing,and thus if necessary having to recover,approximately 0.01% of errors,illustrates the potential of allowable error techniques in this area.

The analyser is able to 'recover' most incorrect statements encountered and the use of confidence levels and the strategy tree certainly improves the quality of these 'recoveries'.

There is 10% of redundant immediate symbols within the main tree F,this redundancy is exploited by the confidence jump mechanism to produce an efficient analysis of statements which are only trivially incorrect.The extent to which errors can be regarded as trivial is increased by the use of allowable error in the siting of the jump nodes,to exploit statistical redundancy (5.21).

The Data Set II is singled out by its numerous points of divergence from the general usage pattern closely adhered to by the other four Data Sets.The programs that compose Data Set II were all written by schoolboys and are mainly concerned with character manipulation and are generally non-numerical, as evidenced by the very low proportion of assignment statements (24%,5.112). The individual program structures seem to have been influenced more by a desire to investigate the full range of the FORTRAN lamguage than a need to solve specific problems.

6.0 Future development of the system.

Within this chapter we describe some possible avenues of extension of our analyser.

In section 6.1 we discuss possible improvements to the restructuring mechanisms.In 6.2 we consider ways of improving the processing of imprecise statements.

We describe an extension of the experience tree structure up to the level of program logic in order to improve the quality of error recovery (6.22).A similar extension of the experience tree hierarchical structure down to the level of character recognition is described in 6.13.

Finally (in 6.3),we consider the incorporation of program context to extend the power of the statement translation feature using the pattern technique (described in 4.31).

6.1 The development of the analysis and restructuring techniques

First we consider the use of delimiter checking to supplement the usual 'simple precedence' analysis in areas of the experience tree in which alternative tree items are approximately equally likely to be correct (6.11).

Second,a subroutine discovery procedure is described and a method of incorporating 'similar' rather than exact subtrees within our tree hierarchy (6.12).We suggest attaching argument lists to nodes which reference subtrees (see figure II),as a means of implementing both of the features described.

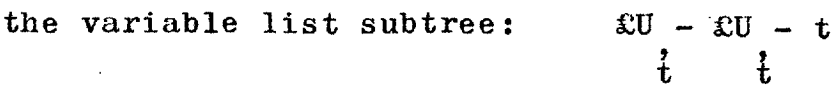
In section 6.13 we give an example of the extension of the basic tree hierarchy in which the analysis of a statement is begun with parts of characters that are examined to produce 'character' decisions which are then processed further to produce statement decisions.

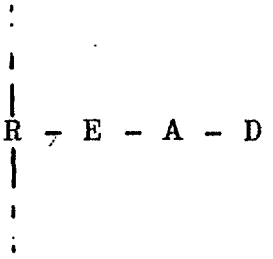
6.11 Delimiter checking in conjunction with the 'simple precedence' analysis

The simple precedence structure of the experience tree which directs the analysis of incoming statements is not particularly efficient with subtrees in which the items are approximately equally likely.For example,in the analysis of a variable list,when a variable name consisting of three characters is only slightly less likely than a four character name.After matching three characters of a variable name it would,perhaps be more efficient to determine if the next character is a delimiter say a comma,rather than to test for all 36 possible variable name characters because,as a group, they are more likely to occur than the comma.The important point is that the difference between the confidence level values could be used as the criterion of whether the analysis should be 'simple precedence' (as we have used and described earlier,4.11) or not.

The delimiter checking feature might be implemented by attaching an argument list to subtree reference nodes.For example,a reference to the subtree of valid variable names from say,the variable list subtree,would carry the appropriate delimiters with it.In figure I these delimiters would be a ',' and the terminal symbol 't'.Similarly the reference from the input statement illustrated (again in fig. I) would carry the character ')' as the appropriate delimiter value.

Fig. I - the argument list delimiters attached to subroutine reference nodes





where the node illustrated as '£U' is a reference to the subtree which can match a valid variable name,subtree U.

6.12 Automatic subroutine discovery

The promotion mechanism which we have introduced and described (4.122) as an extension of the branch swopping mechanism,can also be viewed as the first step towards an automatic subroutine discovery mechanism.

This subroutine feature would minimise the storage space occupied by the tree structures which direct the analysis process.We utilise this feature when initially setting up the experience tree (see 4.03),but now we are considering a procedure,which as the trees develop through the 'learning' process,is able to discover common substructures for itself and to restructure the trees automatically in terms of the common substructures.Such a procedure would be an automatic subroutine discovery mechanism.The crux of this problem is, 'what is a useful subroutine?';the criteria which begin to answer this question and thus form the basis for the automatic subroutine discovery mechanism are described below.

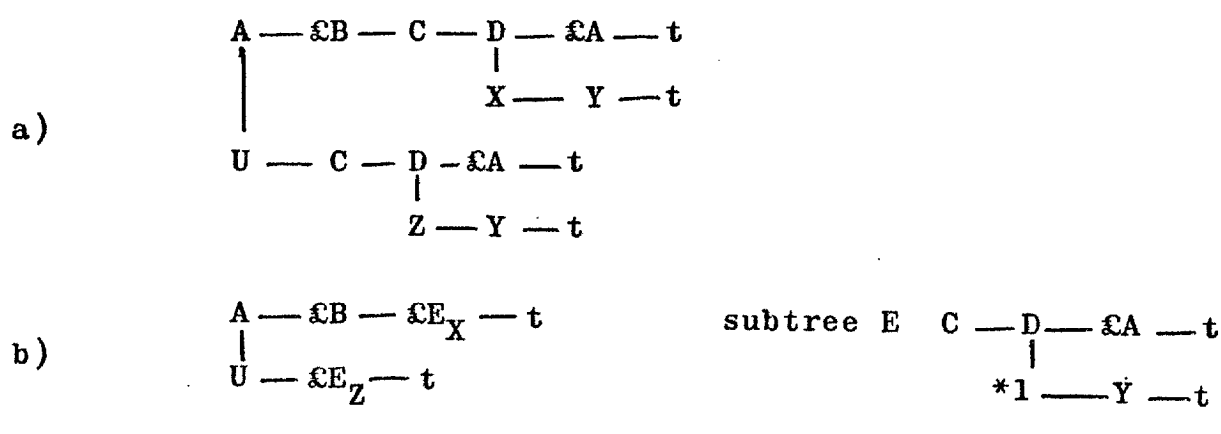
The development of such a mechanism in a general form represents a formidable problem.As with the promotion mechanism the task would be to maintain a balance between minimising the average analysis time by moving common particular items to higher levels in the total hierarchy; and minimising the total hierarchy size by removing infrequently used and similar substructures to lower levels where

the substructure is specified only once and referenced from the appropriate positions - that is,subroutines are created. Clearly the promotion mechanism only deals with a small part of this problem,that is,the promotion of frequently accessed items to higher levels within the hierarchy.The mechanism deals with the particularisation aspect but not the generalisation of the total subroutine discovery procedure.

The difficulty of implementing such a mechanism is that of determining an optimum position of compromise between the complex interaction of opposing features.Such features as, the total hierarchy size,average analysis time and the degree of 'similarity' between substructures.

The question of the similarity of substructures is important, for it is unlikely that many identical substructures will be found;but mainly substructures which are similar in structure and node contents.The experience tree used in the analysis described earlier contained only identical substructures as subroutines.To incorporate similar substructures into the experience tree hierarchy,we again suggest the idea of subroutine arguments.As an example,consider the tree illustrated in figure IIa and the subsequent removal of two similar substructures as the subtree E,figure IIb.

Fig.II - an example of the removal of similar subroutines



The subscripts X and Z associated with the subroutine reference nodes '£E' are in each case,the first value in the argument list and will be inserted in the node '*1' within the subtree E,as the subtree is referenced.

The work of RIJSBERGEN (1971) described earlier (2.01), most closely resembles this aspect of our analyser.He uses a "dissimilarity coefficient" to cluster items and form useful subroutines.

6.13 Extension of the tree hierarchy down to the level of character recognition.

The analyser described in the earlier chapters uses complete characters as the basic symbols recognised.We are now considering an extension in which characters are recognised as a result of a series of lower level decisions.The basic features recognised will then be parts of characters.

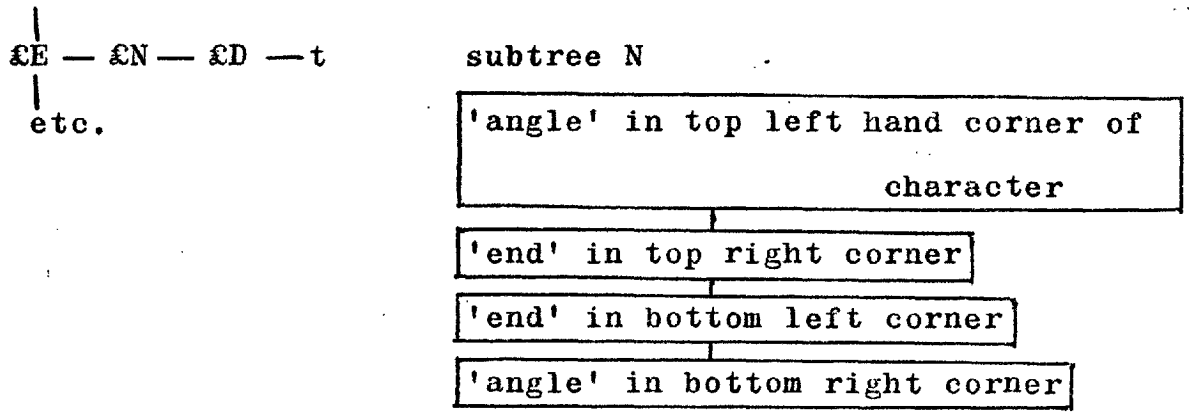
The character recognition involved will be completely integrated with the analysis of the program as a whole.The lowest level subtrees would specify parts of a character within each node and a sequence of nodes,a total tree item, might specify a complete character.

A fundamental ability of our analyser is to be able to process incomplete and poor quality input.Thus this extension might be particularly applicable to the recognition of 'real' characters,where the poor quality and variability of characters,even machine printed ones,has defeated all general attempts to produce an automatic recogniser.Techniques such as the confidence jump mechanism (4.21),which exploits redundancy within the input data in order to efficently process poor quality information,might prove to be particularly useful.

Consider,for instance,that the handwritten statement,'END' is being processed.A possible tree hierarchy for the recog-

nition of this statement is illustrated in figure III.

Fig. III - part of the tree hierarchy for the recognition of statement 'END'



note: the successive nodes of subtree N are illustrated vertically contrary to our normal convention, and subtrees E and D are not illustrated.

Given that the letter E has been recognised and matched at the 'character' level on the main tree in figure III, then the node '£N' represents an entry to the subroutine for recognising N. Assume that N is the only character normally encountered after E as illustrated in the main tree above (fig. III). Then the confidence jump mechanism may ensure that only the feature described by the first node in the subroutine, 'angle in top left corner', is actually checked for before the return to the character level (i.e. node '£N') reporting a successful match of character N.

This leads to the prospect of recognising statements from the approximate matching of only one or a few salient character features, thus avoiding a time consuming detailed analysis of the redundant features. It is appreciated that these suggestions completely gloss over well known difficulties in conventional systems, but we suggest that the use of context

at several levels simultaneously may make it not even necessary to solve problems which are currently absorbing a considerable effort.

The organisation of this recognition system is similar to that described by SELFRIDGE (1959,1970) in which a hierarchy of "demons" are organised to translate Morse into letters, then words,etc. in successively higher levels.The extra feature in our analyser is that it will 'force' a match with poor quality input data and thus obtain analytical information that will be used to optimise future processing.

6.20 The further optimisation and improvement of tolerant processing.

We first describe a method of speeding the processing of common non-trivial errors (6.21),and then a possible extension of the tree hierarchy up to the level of program logic in order to improve the quality of error restoration when a consideration of only previous frequencies of occurrence is not sufficient (6.22).

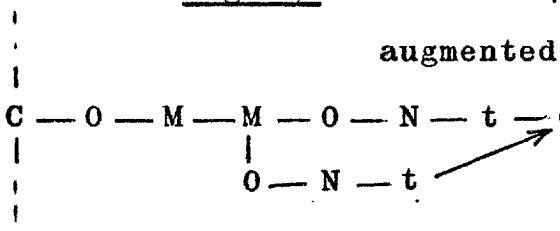
6.21 Optimisation of the 'force fit' error recovery process.

The initial structure of the tree hierarchy provided to the analysing algorithm expresses the structure of all 'expected' language statements.The operation of the 'force fit' (4.22) will be required for all 'unexpected' statements (except those containing only trivial errors) including those which involve common mistakes.Since it is inefficient to continually invoke a lengthy fitting process for a succession of incorrect statements with identical structures,the structure of the experience tree can be augmented to incorporate structures which correspond to common mistakes and a successful match of these structures can lead to the same

outcome (i.e. the same pattern) as the corresponding correct structures.The successful match against an 'incorrect' structure will always be faster than the corresponding force fit to the correct structure.

For instance,if the FORTRAN statement 'COMMON' is often misspelt as 'COMON' then the partial syntax tree illustrated earlier (4.04,fig.IV) would develop the following structure.

Fig.IV - the correct tree item 'COMMON' augmented with an incorrect structure



It is possible to add to the experience tree every incorrect structure which can be matched,but since the universe of possible mistakes is far larger than that of correct structures it is not practical to retain details details of infrequent mistakes.Thus there needs to be at intervals a weeding out of the infrequent mistakes or more correctly,a pruning of the experience tree;this pruning would be controlled by the frequency counts.It is also necessary to take precautions not to prune away statement structures which are correct but seldom used.

6.22 Improving the quality of corrections.

There are important areas of programming error which our system,which is based upon the frequency of previous occurrence is not sufficiently sensitive to recover 'correctly';except for the occasional fluke.For example,many programming errors stem from the usage of particular program variables.If a variable name has a syntax error,say 'AB.D' (illegal period),then our analyser would detect this error and modify the variable

dependent upon the most common general strategy (i.e. the highest precedence strategy from the strategy tree,4.22) that produces a syntactically correct variable name,say the strategy 'one character wrong'.Assuming 'A' is the most commonly used letter,then the modified variable name would be 'ABAD',which although it is syntactically correct,it is unlikely to be the name intended by the programmer.

To produce more likely corrections,in these circumstances, we must extend our hierarchy up to the level of program logic. We could,for example,have a subtree on this level which dictates the different,valid sequential uses of a variable name,'the variable usage' subtree.Each of the program variable names,processed so far,are associated with a node within this tree.Then the illogical use of a particular name will become apparent (in exactly the same way is syntactic errors) by the lack of an alternative within the tree at the current position. We can then examine this variable as a misspelling of another which is valid at the current position within the 'variable usage ' tree,using the usual force fit technique.

6.30 An extension of the pattern technique for statement translation.

The translation ability of our analyser,via the pattern technique (4.31),is currently limited to the level of statement context;this is because the list of parameter values is recreated as each statement is analysed.and discarded before the next statement is input.An extension of the pattern technique to embrace program context can be implemented by maintaining the same parameter list throughout the processing of a complete program,with perhaps sublists for each subroutine.

The patterning of the experience tree would have to be

extended to allocate parameter numbers with regard to the total experience tree structure and not just,as at present, with regard only to the total statement structure.

7.0 Summary and conclusions.

A generalised system for language analysis and translation has been described,and we have demonstrated in detail one particular application,in which statements in the FORTRAN language are checked for syntactic correctness and corrected if necessary.

The FORTRAN analyser was used to process 'normal' FORTRAN programs in five batches each of which originated within a different enviroment.

A similar pattern of usage of FORTRAN was found within the five enviroments.The most interesting point being not, the differences say,between commercial and academic enviroments, but the close similarity between all (except possibly Data Set II,5.30) enviroments with respect to out method of analysis (5.112)

The practical utility of the two heuristic techniques (4.121, 4.122) was assessed in terms of the resultant improvement in processing ability.The branch swopping mechanism was found to be particularly good for optimising the analysis process.

We have also demonstrated the large potential optimisation of the analysis process that can be gained at the cost of coping with a very small proportion of incorrect results (5.14). As with the promotion mechanism (4.122),the final position adopted must represent a compromise between the optimisation gained and the amount of adverse effect (such as,increase in size of data structure or proportion of resulting errors) that can be tolerated in a particular application.

10% of immediate symbols within the language description are structurally redundant (5.21),but this redundancy is virtually all concentrated within the statement keywords.Thus error 'correction' as opposed to the recovery of a syntactically correct structure,is only feasible for errors in keywords

without some extra information.This extra information may be gained from 'past experience' (5.223) and/or program context (6.22).The error recovery ability of our analyser was clearly demonstrated and further improvement in recovery to what we might term 'correction' was shown to result from the use of past experience.

We can view this extra information as providing extra redundancy,for as RANDELL (1971,1972) has repeatedly said, "All error detection is based on the provision of redundant information,whose consistency can be checked".He suggests that complex computer systems might be provided with redundancy,spread throughout in the interests of reliability. Similarly in conjunction with the work to develop less error-prone languages,we should also consider adding some redundancy throughout the language to aid error correction (a conclusion also reached by BACKHOUSE 1972).A confidence jump type mechanism (4.21) would largely eliminate the opposition between less error-prone languages and more redundancy.

The large quantities of analytical information available as a by-product of our analyser,point to immediate savings that can be made in more conventional systems by analysing very common statements as particular instances before invoking the generalised mechanism.The main conclusion to be drawn from this information is,as KNUTH (1970) found,that compilers spend most of their time doing surprisingly simple things.

REFERENCES

ALBERGA, C. N. (1967). String similarity and misspellings, Comm. ACM 10 5 (May 1967)

ALPIAR, R. (1971). Double syntax oriented processing, Comp. J. 14 1 (1971), 25-37

ANDERSON, R. H. (1971). An introduction to linguistic pattern recognition, Report P-4669 RAND Corp. Santa Monica, Calif. (July 1971), 38pp.

ARORA, S. R. & DENT, W. T. (1969). Randomized binary search technique, Comm. ACM 12 2 (February 1969) 77-80

BACKHOUSE, R. C. (1972). An investigation into a finite-state tranducer for pre-parsing and spelling correction, DIC Dissertation, Department of Computing and Control, Imperial College, London University (May 1972)

BACKUS, J. W. (1959). The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conf., Conf. Proc. Internat. Conf. Inform. Processing UNESCO (June 1959) 125-132

BARRON, D. W. (1971). Programming in wonderland, Comp. Bull. (April 1971), p.153

BLAIR, C. R. (1960). A program for correcting spelling errors, Information and Control 3, (1960), 60-67

BLOOM, B. H. (1970). Space/time trade-offs in hash coding with allowable errors, Comm. ACM 13 7 (July 1970)

BROOKER, R. A. et al (1963). The compiler compiler, Ann. Rev. Automatic Prog. 3, 229-271

CARBONELL, J. R. (1970).AI in CAI:an artificial intelligence approach to computer assisted instruction, IEEE Trans. on Man-Machine Systems MMS-11 4 (December 1970)

CHOMSKY, N. (1964). A transformational approach to syntax,in The Structure of Language,Fodor & Katz (eds Prentice Hall,Englewood Cliffs,New Jersey.

CHOMSKY, N. (1965). Aspects of the Theory of Syntax,MIT Press, Cam. Mass.

CLAPPER, G. L. (1971).Automatic word redognition,IEEE Spectrum (August 1971) 57-69

COFFMAN, E. G. & BRUNO, J. (1970).On file structuring for non-uniform access frequencies,University of Newcastle upon Tyne,Computing Laboratory, Tech. Rep. Series no.6 (January 1970)

COHEN, J & CHRISTENSEN, I. (1970).Information and Choice,Oliver & Boyd,Edinburgh,1970

CONWAY, R. W. & MAXWELL, W. L. (1963).CORC - The Cornell computer language,Comm ACM 6 6 (June 1963) 317-321

CONWAY, R. W. et al (1971).User's guide to PL/C - the Cornell compiler for PL/1,Department of Computer Science,Cornell University,Ithaca,New York (August 1971)

CONWAY, R. W. & WILCOX, T. R. (1971a).Design and implementation of a diagnostic compiler for PL/1,Research Report 71-107,Dept. Computer Science, Cornell University (September 1971)

CORNEW, R. W. (1968). A statistical method of spelling correction, Information and Control 12 (1968) 79-93

DAMARAU, F. J. (1964). A technique for computer detection and correction of spelling errors, Comm ACM 7 3 (March 1964)

DARDEN, S. C. (1969). A contextual recognition system for formal languages, Proc. IJCAI Walker & Norton (eds.) (May 1969) 597-607

DAVIDSON, L. (1962). Retrieval of misspelled names in an airline passenger record system, Comm ACM 5 3 (March 1962)

DAY, A. C. (1970). The use of symbol-state tables, Comp. J. 13 4 (November 1970), 332-339

EVERSHED, D.G. & RIPPON, G. E. (1971). High level languages for low level users, Comp. J. 14 1, 89-90

FORSTER, D. F. (1970). Man-machine interaction for the discovery of high-level patterns, Proc. AFIPS 1970 SJCC Vol. 36, AFIPS Press, Montvale, 649-651

FU, K. S. (1968). Sequential methods in pattern recognition and machine learning, Academic Press, New York, 1968

GALLI, E. J. & YAMANDA, H. (1967). An automatic dictionary and the verification of machine readable text, IBM J. Res. & Dev. 6 3 (1967)

GLANTZ, H. T. (1957). On the recognition of information with a digital computer, J. ACM 4 (1957), 178-188

GOOD, I. J. (1965). The first ultraintelligent machine, Advances in Computers 6, Academic Press, New York, 1965, 31-88

GOOD, I. J. (1969). Some future social repercussions of computers, Proc. Symposium Cybernetics in the Seventies and Conflict Resolution, Spartan Press, 1970

GOOD, I. J. (1969a). Statistics of language:introduction,in Ency. of Linguistics,Information and Control,Pergamon Press,Oxford,567-581

GUZMAN, A. (1968). Decomposition of a visual scene into three-dimensional bodies,Proc. AFIPS 1968 FJCC Vol.33,Thomson Book Co., Washington,1968, 291-304

*see Addendum p.168

HOCKETT, C. F. (1968).The State of the Art,Mouton,The Hague, 1968

HUESMAN, L. R. (1970).A study of heuristic learning methods for optimisation tasks requiring a sequence of decisions, Proc. AFIPS 1970 SJCC Vol.36,AFIPS Press, Montvale,1970, 629-647

IRONS, E. T. (1963). An error-correcting parse algorithm Comm. ACM 6 11 (November 1963)

IVAKHNENKO, A. G. (1970).Heuristic self-organization in problems of engineering cybernetics,Automatica 6 207-219

JACKSON, M. (1967). Mnemonics,Datamation 13 (April 1967)

JAMES, E. B. & PARTRIDGE, D. (1972).Machine intelligence:the best of both worlds?,Int. J. Man-Machine Studies 4 (1972), 23-31

KISS, G. R. (1969). A computer model for certain classes of verbal behaviour,Proc. Internat. Joint Conf. on Artificial Intelligence Washington D.C. (May 1969)

KISS, G. R. (1969a). Steps towards a model of word selection, Machine Intelligence 4 Michie D.(ed.), University Press,Edinburgh,1969

KNUTH, D. E. (1970). An empirical study of FORTRAN programs Stanford A.I. Project Memo AIM-137, Com. Sci. Dept. CS-186

KUNO, S. (1965). The predictive analyser and a path elimination technique, Comm. ACM 8 7 (July 1965), 453-462

KUNO, S. (1966). The augmented predictive analyser for context-free languages - its relative efficiency, Comm. ACM 9 11 (November 1966), 810-823

LAFRANCE, J. (1970). Optimisation of error recovery in syntax directed parsing algorithms, SIGPLAN Notices 5 12 (December 1970), 2-17

LAFRANCE, J. (1971). Syntax-directed error recovery for compilers, Ph.D. Thesis, University of Illinois at Urbana-Champaign, 1971

LEVY, J. (1971). Automatic correction of syntax errors in programming languages, Ph.D. thesis, Cornell University, Ithaca, New York, (December 1971)

LONGUET-HIGGINS, H. C. & ORTONY, A. (1968). The adaptive memorisation of sequences, Machine Intelligence 3, Michie D. (ed.) University Press, Edinburgh, 1968

MARTIN, W. A. & NESS, D. N. (1972). Optimising binary trees grown with a sorting algorithm, Comm. ACM 15 2 (February 1972), 88-93

MICHIE, D. (1967). 'Memo' functions: a language feature with rote learning properties, Res. Memo. MIP-R-29, Dept. Machine Intelligence and Perception, University of Edinburgh

MICHIE, D. (1968). 'Memo' functions and machine learning, Nature 218 (1968), 19-22

MICHIE, D. & ROSS, R. (1970). Experiments with the adaptive graph traverser, Machine Intelligence 5 Meltzer Michie (eds.), University Press, Edinburgh, 1970, 301-318

MICHIE, D. et al (1970a). Experimental programming 1970, Exptl. Programming Unit, Dept. MIP, University of Edinburgh (July 1970)

MILLER, G. A. (1962). Decision units in the perception of speech, IRE Trans. on Information Theory (1962) 81-83

MILLER, W. F. & SHAW, A. C. (1968). Linguistic methods in picture processing - a survey, Proc. AFIPS FJCC 1968 Vol.33, Thomson Book Co., Washington, 1968 279-290

MINSKY, M. (1970). Form and content, J. ACM 17 2 (April 1970) 197-215

MORGAN, H. L. (1970). Spelling correction in systems programs, Comm. ACM 13 2 (February 1970), 90-94

MOYNE, J. A. (1969). Information retrieval and natural language, Proc. Amer. Soc. for Information Science 6 259-263

MUNSON, J. H. (1968). Experiments in the recognition of hand-printed text: Part I - character recognition, Proc. AFIPS 1968 FJCC Vol.33, Thomson Book Co., Washington, 1968, 1125-1138

McELWAIN, C. K. & EVENS, M. B. (1962). The degarbler - a program for correcting machine-read Morse Code, Information and Control 5 (1962), 368-384

NORMAN, D. A. (1969). Memory and Attention, an introduction to human information processing, Wiley, New York, 1969

NOTLEY, M. G. (1970). The cumulative recurrence library, Comp. J. 13 1, (February 1970), 14-19

NOTON, D. (1970). A theory of visual pattern perception, IEEE Trans. on Systems Science & Cybernetics Vol.SCC-6, No.4 (October 1970), 349-357

PALME, J. (1972). Developments in using natural language, Computer Weekly, April 27th, 1972

PATT, Y. N. (1969). Variable length tree structures having minimum average search time, Comm. ACM 12 2, (February 1969), 72-76

POPPER, K. R. (1968). The logic of scientific discovery, Hutchinson, London, 1959 (2nd. edition 1968)

RANDELL, B. (1971). Highly reliable computing systems, University of Newcastle upon Tyne, Computing Laboratory, Tech. Res. Rep. Series, no.20 (1971)

RANDELL, B. & HORNING, J. J. (1972). Process structuring, University of Newcastle upon Tyne, Computing Laboratory, Tech. Res. Rep. Series, no.31 (March 1972)

RIJSBERGEN, C. J. van (1971). An algorithm for information structuring and retrieval, Comp. J. 14 4 (November 1971), 407-412

*see addendum p.168

SELFRIDGE, O. G. (1959, 1970). Pandemonium: a paradigm for learning, in Mechanisation of thought processes, HMSO (1959), 511-527
reprinted, in Perceptual learning and adaptation, Dodwell (ed.), Penguin Books, Harmondsworth, 1970, 465-478

SELFRIDGE, O. G. & NEISSER, U. (1960).Pattern recognition by machine,Scientific American 203 2 (August 1960)

SHAW, A.C. (1970). Parsing of graph-representable pictures, J. ACM 17 3 (July 1970),453-481

SLAGLE, J. R. (1963). A heuristic program that solves symbolic integration problems in Freshman calculus, in Computers and Thought,Feigenbaum and Feldman (eds.),McGraw-Hill,New York,1963

SLAGLE, J. R. & LEE, R.C.T. (1971).Application of game tree searching techniques to sequential pattern recognition,Comm. ACM 14 2 (February 1971),103-110

SLAGLE, J. R. & FARRELL, C. D. (1971a).Experiments in automatic learning for a multipurpose heuristic program,Comm. ACM 14 2 (February 1971),91-99

SPARKES, J. J. (1969).Pattern recognition and a model of the brain,Internat. J. Man-Machine Studies 1 (1969), 263-278

STANFEL, L. E. (1970).Tree structures for optimal searching, J. ACM 17 3 (July 1970), 508-517

SUSSENGUTH, E. H. (1963).Use of tree structures for processing files,Comm. ACM 6 5 (May 1963),272-279

SZANSER, A.J.M. (1968).Error-correcting methods in natural language processing,in Information processing 68,Morrell (ed.),North Holland Publ. Co.,Amsterdam,1968,vol.II,1412-1416

SZANSER, A.J.M. (1971).Automatic error correction in natural texts - part I,National Physical Laboratory Com. Sci. 46, (May 1971), 38pp

TEITELMAN, W. (1972). "DO WHAT I MEAN":The programmer's assistant,Computers and Automation (April 1972), 8-11

THORELLI, L. E. (1962).Automatic correction of errors in text, BIT 2 (1962), 45-62

UTTLEY, A. M. (1970). The Informon:a network for adaptive pattern recognition,J. theor. Biol. 27 31-67

VIGOR, D.B.,URQUHART,D. & WILKINSON,A. (1969) PROSE - parsing recogniser outputting sentences in English, Machine Intelligence 4 ,Meltzer & Michie (eds.),University Press,Edinburgh,1969

WALWYN, P.R. (1971). Correspondence in Comp. J. 14 4,p.425

WATERMAN, D. A. (1970).Generalization of learning techniques for automating the learning of heuristics, Artificial Intelligence 1 (1970),121-170

WEIZENBAUM, J. (1963).Symmetric list processing - SLIP, Comm. ACM 6 9 (September 1963),524-544

WOODS, W. A. (1970). Transition network grammars for natural language analysis,Comm ACM 13 10 (October 1970),591-606

Addendum:

HEINDEL, L. E. (1972). private communication, 15 May 1972

HEINDEL, L. E. & ROBERTO, J. T. (1972a).LANG-PAK - an interactive incremental compiler-compiler,Bell Telephone Laboratories,Holmdel,New Jersey, 12 pps.

SAMUEL, A.L. (1959). IBM. J. Res. Dev. 3, 210

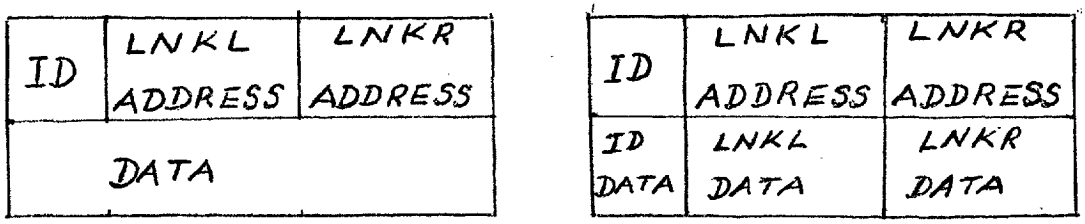
APPENDIX I - THe MINISLIP routines.

The purpose of these routines is to enable the construction of lists and binary trees plus a limited amount of list and tree processing.The Symmetric List Processing (SLIP) originates with WEIZENBAUM(1963) but have been altered in many ways by various programmers since.We use a very limited version of the original system and hence the name MINISLIP.

Most of the routines are written in FORTRAN IV;the few machine dependent routines are available in CDC Assembler and MAP (IBM 7090/7094) at present.These routines are collected in a single deck called MCOPS.Conversion to other machines and/or languages is trivial.

The basic item of information is a cell,which occupies one or more (usually two) machine words.A cell contains four fields,one of which can further subdivided into three thus giving a maximum of six fields per cell.

Fig. I - diagrammatic representation of the two subdivisions of a cell.



MCOPS

The following ten functions and subroutines, written as one machine dependent routine with ten entry points called MCOPS perform basic operations on the components of a cell.

The following examples will be used to illustrate the functions of the routines described.

Assume this cell and the Fortran variable, NAME, are

17	2666	3434
A B C D E F		

stored at machine location 2455.

Assume that this cell and the Fortran variable, ICELL,

1	3409	2666
A	99	700

are stored at machine location 2457.

1. Function ID(NAME) obtains the values of the ID field.
2. " LNKL(NAME) " " " " " LNKL ADDRESS.
3. " LNKR(NAME) " " " " " LNKR ADDRESS.

Examples: ID(NAME)=17 LNKL(ICELL)=3409 LNKR(NAME)=3434

4. INHALT (N) obtains the contents of the machine location whose address is the value of the integer variable N.
5. CONT (N) is exactly the same as INHALT (N) but is a real function, both are required to overcome the difficulty of unrequired type changes in assignment statements.

Examples: Let the location that is assigned to N have the value (or contain the integer) 2455. Then the value of INHALT (N) is 17 2666 3434 , and LNKR(INHALT(N))=3434 ID(INHALT(N))=17 LNKL(INHALT(N))=2666.

INHALT(N+1) is the contents of the data field which is, 'ABCDEF'.

Let the location that is assigned to MCELL contain the integer 2457. Then ID(INHALT(MCELL+1))=A (the specific character 'A', right adjusted with preceeding zeros).

Similarly LNKR(INHALT(MCELL+1))=700 LNKL(INHALT(MCELL+1))=99

6. MADOV (NAME) obtains the machine address of the Fortran variable NAME.
   Example: MADOV (ICELL) = 2457

The previous six functions have obtained values connected with a cell, the following four subroutines store values in the various fields of a cell.

7. STRDIR (I,N) stores the value I in the machine location occupied by variable N.
   Example: CALL STRDIR(N,NAME+1) if N contains the characters 'ABCDEF', then the cell stored at the same location as variable, NAME, will contain 'ABCDEF' as the data item, i.e. the cell will be as illustrated.

8. STRIND (I,N) stores the value I in the machine location addressed by the contents of N.
   Example: CALL STRIND(NDATA,N+1) if variable NDATA contains the character string 'ABCDEF' and variable N contains the integer 2455, then the characters 'ABCDEF' will be stored in the data field of the cell as illustrated.

9. SETDIR (I,J,K,L) puts the values of the Fortran integer variables I,J and K into the ID,LNKL and LNKR fields of the machine location occupied by the variable L. If any of I,J or K have a negative value, then the corresponding field is unaltered in L.
   Example: CALL SETDIR(I,J,K,NAME) if I=17, J=2666 and K=3434 then the ID, LNKL ADDRESS and LNKR ADDRESS fields will be as illustrated for cell at 2455.

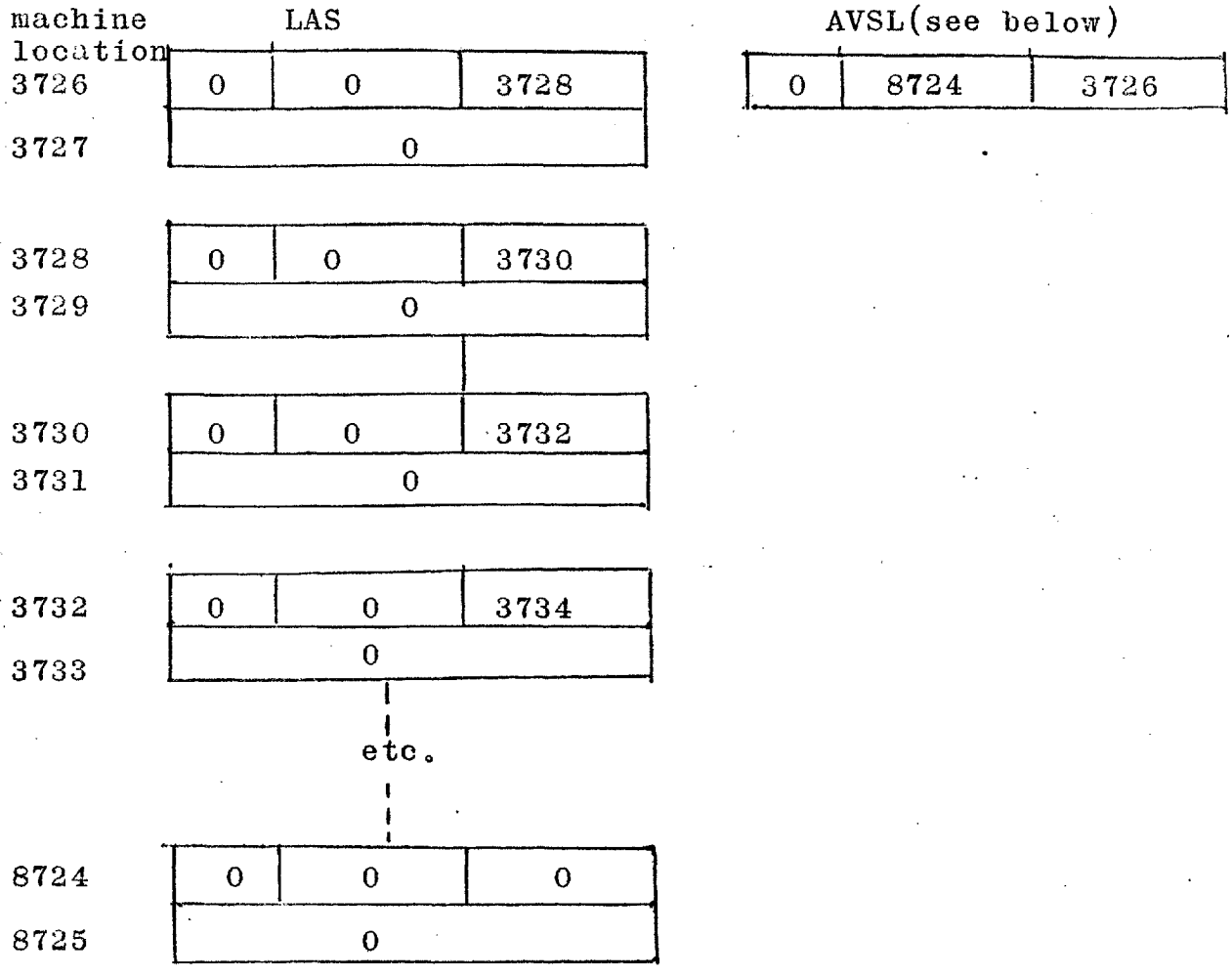
CALL SETDIR(-1,-1,-1,NAME) will leave the cell unaltered.

10. SETIND (I,J,K,M) has the same effect as SETDIR except that the values are set into the fields of the machine location whose address is the value of variable M.
Example: CALL SETIND(I,J,K,M) followed by CALL SETIND(I1, J1,K1,M+1),if I=1,J=3409,K=2666,M=2457,I1=99,K1=700 and I1 contains the right adjusted character 'A',then the second cell as illustrated would be set.

All the variables in the previous routines can of course be replaced with arithmetic expressions where appropriate.
Example: CALL SETIND(ID(INHALT(NAME)),LNKL(INHALT(NAME)), LNKR(INHALT(NAME)),NAME) does nothing at all.

The following routines are written in Fortran IV and reference MCOPS:-

1. ERROR (J) writes out error messages.Used only internally to the other routines,but we have adapted to set a flag when an error occurs which can be examined within the main system.
2. INITAS (A,N) this subroutine is used at the start of the program to create a collection of cells,called the List of Available Space (LAS).The Fortran array A is declared prior to the call to INITAS,and must be of sufficient size to provide space for the number of cells required,N.
Example: INTEGER A(5000) followed by CALL INITAS(A,5000) assuming no more than two machine words are used for each cell,will create a LAS with 2500 cells (N/2 or (N-1)/2 if N is odd).If A(1) is stored at location 3726 and there are two machine words per cell,then the structure will be as follows:-



AVSL is a special location initialised and stored within MCOPS, it carries in its LNKR field the address of the first cell and in its LNKL field the address of the last cell in the list. If LNKR(AVSL) becomes zero, then there are no cells remaining in the list and an error message to this effect is output via a call to routine ERROR.

2. NUCELL (L) detaches a cell from the LAS, the value of L is set to its address.

   Example: CALL NUCELL(ICELL) will detach the first cell by setting LNKR(AVSL)=3728 and L=3726

3. RCELL (L) attaches the cell whose address is in L to the LAS.

   Example: CALL RCELL(ICELL) if considered as succeeding the

previous example and thus variable ICELL=3726,then the cell previously removed will be returned to LAS,but at the other end.The LNKR ADDRESS field of the last cell (the cell whose address is 8724) is set to 3726 and LNKL(AVSL) to 3726.

The following routines are used to set up and manipulate symmetric lists,which enable the structure to be traced in either direction.The LNKR ADDRESS field in each cell carries the address of the succeeding cell in that list,the LNKL ADDRESS field carries that of the preceeding cell.

4. LIST (M) creates an 'empty' list ready for later use by detaching one cell from the LAS and setting it up as a list 'header'.The LNKL ADDRESS and the LNKR ADDRESS fields of the cell are both set to its own address.The contents of the variable M are also divided into an ID,LNKL and LNKR fields,the values set are 0,the header cell address,and the header cell address ,respectively.

Example: CALL LIST(NLIST) if the LAS is as illustrated previously,the first cell is detached (cell address 3726) and LNKR(AVSL) is set to 3728,the header cell and the variable NLIST are set as follows:-

the list 'header' cell

3726	0	3726	3726
3727	0		

the variable NLIST

0	3726	3726

Later when cells are added to and subsequently removed from the list,the LNKL and LNKR fields of the header contain the address of the bottom cell and top cell of the list.The list is referred to by the Fortran name NLIST, which now the name of a list or a valid list name.

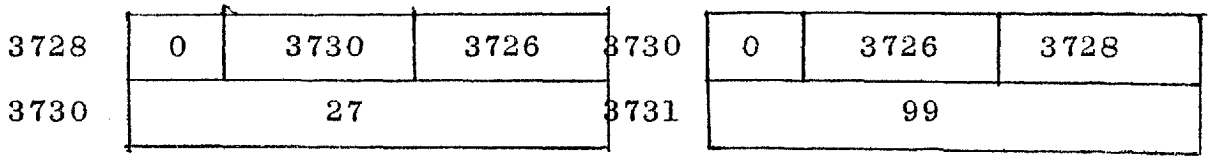
5. LOCT (M) obtains the address of the header cell of list M.
   Example: succeeding the previous call to LIST,LOCT(NLIST) =3726

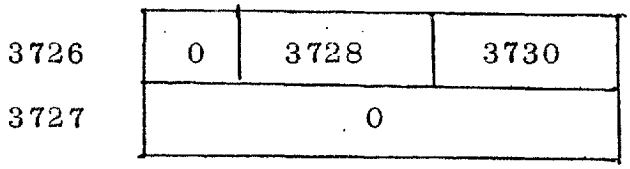
6. NAMST (M) is a logical function which has the value TRUE if M is a valid list name,that is if the LNKR ADDRESS field and the LNKL ADDRESS field of the variable are equal.

7. LISTMT (M) is a logical function which has the value TRUE if list M is empty,that is if the contents of the first word in the list header cell are equal to the contents of the list name variable.(see example after routine 4. LIST)

8. NEWTOP (I,M) inserts the value of I at the top of list M.
   Example: CALL NEWTOP(J,NLIST) followed by CALL NEWTOP(99, NLIST),consider that we have the list NLIST as illustrated previously,then the list structure will become:-

   assume J=27





9. NOKTOP (M) removes the top cell from list M,and returns it to the LAS,the value of the function is the value of the data field of the cell removed.
   Example: following the above example,I=NOKTOP(NLIST) will return cell 3730 to the LAS and set variable I as 99.

10. NOKOFF (L) removes the cell whose address is the value of L from its surrounding list structure and returns it to the LAS.The value of the function is the data item of the cell deleted.This routine is used by routines NOKTOP and NOKBOT.

11. NOKBOT (M) as for routine NOKTOP but removes the bottom cell of the list.

12. NXTRGT (A,L) inserts a cell immediately 'above' the cell whose address is the value of L.The routine is called 'NeXTRiGhT' because the address of the cell added is set in the LNKR ADDRESS field of the list header cell. The value of the function returned is the address of the cell inserted.This routine is used by subroutine NEWTOP.

APPENDIX II

Description of the routines of the main system.

The routines that comprise the main system are divided into five sections;General Purpose (GP),Input and Output (I/O), Tree Building (TB),Matching Process (MP) and Restructuring Process (RP).Each section is preceded by an index of the routines described within it.Each routine is allocated a code name which consists of its section name abbreviation followed by an integer which specifies its position within that section.

Reference to routines within the same section are given simply as the routine name.References to routines in other sections are given as the routine name preceded by the section code or followed by the code name.

Two flowcharts of the analysis process are given immediately following the index in section MP.The first flowchart gives the logical structure of the analysis process and the second gives the structure in terms of the routine names.

APPENDIX II - Description of the routines

The General Purpose Routines:section GP

Index

1. Identification GP1 BLOCKDATA

Description This routine initialises the Block Common variables by the use of a Data Statement.

Entry Conditions None.

Exit Conditions The Block Common variables,PROCED,SUCCES, FAILUR,ERROR,ENDED and YES,are initialised with various character strings.

Subroutines Used None.

Calling Routines None.

2. Identification GP2 KERROR(K,STATUS,IARG,NROUTN)

Description This routine outputs system error messages, which particular message that is output is determined by the value of the first argument,K.The name of the routine within which the error was detected is passed to KERROR as a character string in the last argument,NROUTN.All messages output by this routine (see APPENDIX VI for details) are preceeded by:- "******* SYSTEM ERROR ******* SIGNALED FROM 'routine name' ". The characters "ERROR" are assigned to the second argument STATUS,before exit.

Entry Conditions There are three integer arguments and one real,STATUS.The only argument not described above can be used to pass an integer constant or character string for output within the subsequent error message.

Exit Conditions The values of the three integer arguments remain unchanged while the second argument,STATUS,is assigned the character string "ERROR".

Subroutines Used None.

Calling Routines MACHIN,NBOTOP,SPACE,SWPSTK,UNSTAK

Section MP:BRANCH,CHECKS,FAIL02,FAIL07,ICOUNT,IMATCH,PROC01, SAMCEL,SETCEL,SETKSG,SETEST,STOSET,STRAT,SUCC01,SUCC07,SUCC10, TRUNID,TRUN01,TRUN02,TRUN07,TRUN08,TRUN03,TRUN04,PROC04,INSERT RP:PATERN,PATOUT,JUMPIN,JMPOUT,KNODES,SETJNC,RESTRC,ASORT,COPY,

3. Identification GP3 EMPTY(NSTACK)

Description This routine empties the list, the address of whose name is passed as the value of the argument, NSTACK.

Entry Conditions The single argument, NSTACK, must contain a valid list address (see APPENDIX I).

Exit Conditions None.

Subroutines Called MINISLIP: LISTMT, NOKTOP

Calling Routines STOINV

Section MP: BRANCH, CHECKS, ICOUNT, IMATCH
RP: all except PROMOT, TOTITM, INTREE
I/O: NUMODD

4. Identification GP4 MACHIN(SWIT, IARG1, STATUS)

Description The function of this routine is to focus the accessing of all machine dependent routines (except MCOPS APPENDIX I). The value of the first argument, ISWIT, determines which routine is called.

Entry Conditions The first argument must have an integer value on entry.

Exit Conditions The value of the second argument is overwritten to carry a return value.

Subroutines Used KERROR, LFTORT, RGTOLF

Calling Routines Section I/O: SETUP, TPRINT
Section TB: NTGROW, SAMCEL, SETCEL
Section MP: SETEST

5. Idenfication GP5 LFTORT(LOCATN)

Description Routine moves a single character from the left end of the locatioh whose address is passed as the value of the argument, LOCATN, to the right end and fills with preceeding zeros (to maintain compatability with a single character removed from cell by function ID, see APPENDIX I).

Entry Conditions The argument should contain the address of an integer variable.

Exit Conditions None.

Subroutines Used None.

Calling Routines MACHIN

6. Identification GP6 NBOTOP(STATUS,ISTACK,INUM,NSTACK, NADDED)

Description This routine transfers INUM items,from the bottom of the list whose address is in ISTACK to the top of the list whose address is in NSTACK.

If INUM=-1 then all the items,if any,in list ISTACK are transferred,in the above described manner,to list NSTACK.

If the value of INUM is less than -1 on entry or the list ISTACK is emptied before INUM items have been removed,then an error condition is activated which causes an appropriate message to be output (via a call to KERROR) and control returned with the argument STATUS assigned the character string,"ERROR".

The number of items transferred is returned in the argument NADDED.

Entry Conditions The second and fourth arguments must contain valid list *name* addresses (see APPENDIX I) and INUM must contain an integer not less than -1.

Exit Conditions If the value of INUM is greater than -1 on entry,then it is returned with the value zero,else it remains unchanged.The fifth argument,NADDED,is returned containing the integer number of items transferred.

If an error condition has been activated the variable STATUS is assigned and hence returns the character string "ERROR".

Subroutines Used KERROR and MINSLIP:LISTMT,NEWTOP,NOKBOT

Calling Routines Section MP: CHECKS

7. Identification GP7 RGTOLF(LOCATN)

Description Routine moves a single character from the right hand end of the location whose address is passed as the value of the argument,LOACTN,to the left hand end usually just prior to output under Fortran A1 format.

Entry Conditions The argument should contain an integer variable address on entry.

Exit Conditions None.

Subroutines Used None.

Calling Routines MACHIN

8. Identification GP8 SPACE(I,N)

Description This routine monitors the number of cells currently in the List of Available Space (LAS,see APPENDIX I), and is used to return;the current number of available cells, if I=4;the minimum number that has been available,if I=5;and the total number of cells initialised by the last call to INITAS (APPENDIX I),if I=6.The values 1 and 2 for I are used by routines NUCELL and RCELL (both APPENDIX I),respectively to alter the count of the current number of cells.The value of 3 is used for I by routine INITAS to set the total number of cells initialised.

Entry Conditions The argument I must contain an integer value of between 1 and 6,inclusive,else and error condition is activated and a message output.

Exit Conditions The value of the second argument,N, is overwritten on exit if I=4,5 or 6,or is less than 1 or gr ater than 6.

Subroutines Used KERROR

Calling Routines MINISLIP: INITAS,RCELL,NUCELL

9. Identification GP9 STOREN(STATUS,ISTACK,JSTACK,NADDED)

Description Routine copies the list whose name address is in the second argument,ISTACK,into the list,the address of whose name is in the third argument,JSTACK.The number of items copied across is set into the fourth argument,NADDED.

Entry Conditions The second and third arguments must contain valid list addresses (APPENDIX I).

Exit Conditions An integer value is assigned to the fourth argument prior to exit.

Subroutines Used MINISLIP: NOKBOT,NEWTOP

Calling Routines MP:STOSET

10. Idehtification GP10 STOINV(STATUS,ISTACK,JSTACK,IADD)

Description This routine copies the contents of the list whose name address is in the second argument,ISTACK,into the list whose name address is in JSTACK,but in the inverse order.The number of items copied across is assigned to the variable IADD.

Entry Conditions The second and third arguments must contain valid list addresses (APPENDIX I).

Exit Conditions The argument IADD is assigned the number of items copied across between the lists.

Subroutines Used EMPTY

MINISLIP: LISTMT,NEWTOP,NOKBOT,NOKTOP

Calling Routines

MP: STOSET,PROC01,PROC07

11. Identification GP11 SWPSTK(STATUS,ISTACK,INUM,NSTACK, NADDED)

Description If the value of the third argument,INUM,is -1 then the list whose name address is in ISTACK is emptied into the list whose name address is in NSTACK;on exit list ISTACK is empty and list NSTADK contains the previous contents of ISTACK in inverse order on top of the items it contained,if any,on entry.

If the value of INUM is zero or positive then INUM items are removed from list ISTACK and placed in list NSTACK,in inverse order,If the value of INUM is less than -1 an error condition is activated and an error mesage output(via KERROR).

Entry Conditions The arguments ISTACK and NSTACK must contain valid list addresses (APPENDIX I).The third argument INUM must contain an integer not less than -1.

Exit Conditions The third argument INUM is returned as zero,if it was zero or positive on entry;otherwise its value remains unaltered.The fifth argument NADDED is used to return the number of items transferred from ISTACK to NSTACK.

Subroutines Used KERROR

MINISLIP: LISTMT,NEWTOP,NOKTOP

Calling Routines

MP: BRANCH,CHECKS,ICOUNT,IMATCH,STOSET,STRAT
RP: NODECT

12. Identification GP12 UNSTAK(ISTACK,NUM,STATUS)

Description Routine removes and discards the number of items specified by the second argument NUM,from the list whose name address is contained in the first argument,ISTACK.If the list is found to be empty before NUM item have been removed an error is signaled (via KERROR).

Entry Conditions There are three arguments, the first ISTACK must contain a valid list address (APPENDIX I), the second, NUM, must contain an integer which is less than or equal to the number of items in the list ISTACK and the third may be used for a return value (see below).

Exit Conditions The arguments are the same as on entry, unless an error condition has been activated then STATUS will have been reset (in KERROR) to contain the character string "ERROR".

Subroutines Used KERROR

MINISLIP: LISTMT, NOKTOP

Calling Routines

MP: ICOUNT

APPENDIX II - Description of the routines

The routines mainly concerned with input and output:

section I/O

Index

1. I/01 SETUP
2. I/02 TPRINT
3. I/03 PRID05
4. I/04 TAPOUT
5. I/05 TAPEIN
6. I/06 STATS
7. I/07 ADD
8. I/08 DELETE
9. I/09 CHANGE
10. I/010 READ
11. I/011 NUMODD

Note: the routine which outputs system error messages, KERROR is contained within section GP,

all messages that the system can output are listed alphabetically in APPENDIX VI.

1. Identification I/O1 SETUP

Description This routine is used to contruct from a deck of cards,in the current implementation,a hierarchical structure of trees and subtrees to be used by the subsequent Matching Process as a description of the language to be processed and also the details of the processing.

First a list of valid tree names is contructed within the Fortran list NTREES with the corresponding root addresses in the Fortran list ITREES.The tree names are input as a single character in the first column of successive data cards and stored right-adjusted (by means of a call to MACHIN,section GP) so as to be compatible with tree names in the ID data field (APPENDIX III) of a tree node and extracted by means of the MINISLIP function ID.As each tree name is input and stored so a new cell is acquired from the List of Available Space (LAS,See APPENDIX I) and its address is stored within the corresponding location of the Fortran list ITREES and will be the tree root address.Thus the i^{th} location in array ITREES contains the address of the root of the tree whose name is stored within the i^{th} location of array NTREES.The last tree name to be input and stored in the above manner must be "E", then the variable KTREES is set to the total number of tree names input and control is switched to reading complete tree items of each data card.

The data for each tree item is input in one of two ways; expanded,one tree node per card or,condensed,up to 22 nodes per card (coding details in APPENDIX III).After storing the tree names and root addresses control is switched to expect data in the condensed format,but the occurrence of a blank card will cause control to switch to expect input in the condensed format,another blank card will switch control back,and so on.

This process of inputting the data for tree items, which is passed on to routine NTGROW (section TB) for allocation of the item to the tree specified, is terminated by the occurrence of the characters "00000E" in the first six columns of a data card. The tree structures are then output by means of a call to routine TPRINT and control is returned.

There are two error traps which, when activated, cause a message to be printed and the program terminated. The first trap is activated if more than 64 tree names are input (for this is the maximum allowed under the present implementation), the message output is " TOO MANY TREES SPECIFIED, STOPPING ". The second trap is activated when inputting tree item data in condensed format and the item is expected to continue on the next data card, but the next data card does not have the characters "KONTIN" in the first six columns; the message is " ERROR IN DATA SETUP CONTINUATION CARD EXPECTED, NOT FOUND...CARD IS 'card image output on next line' ".

Entry Conditions The variables ITREES and NTREES must be Fortran lists of dimension at least 64. The deck of data cards must have the correct structure (APPENDIX III).

Exit Conditions The Fortran list NTREES will contain all the valid tree names and the Fortran list ITREES all the corresponding tree root addresses.

Subroutines Referenced TPRINT

MINISLIP: NUCELL, STRIND

GP: MACHIN, SPACE

TB: NTGROW

Referencing Routines

MP: the main routine MP1

2. Identification I/02 TPRINT(LHEAD,LDEPTH)

Description This routine outputs the tree structure governed by the node whose address is in the argument LHEAD. The value of the second argument LDEPTH determines if the frequency count associated with each item in each tree is just output as stored or as a percentage of the total frequency counts within the tree structure.

Entry Conditions The argument LHEAD must contain the address of a tree node.

Exit Conditions The values of both arguments remain unchanged.

Subroutines Referenced PRID05

MINISLIP: LIST,ID,INHALT,NEWTOP,LNKR,LNKL,NOKTOP

GP: MACHIN RP: NUMTRE

Referencing Routines SETUP,STATS

RP: PROMOT ,INSERT,INSTRC,NODOUT

3. Identification I/03 PRID05(MROOT)

Description This routine is used to output the strategy tree (see section 2.6,and APPENDIX III for details of structure) in an explicit form,as opposed to the coded form if routine TPRINT is used.

Entry Conditions The argument MROOT must contain the root address of the strategy tree or the address of the first node of any strategy within the tree.

Exit Conditions None.

Subroutines Referenced

MINISLIP: ID,INHALT,LNKL,LNKR

Referencing Routines TPRINT

4. Identification I/04 TAPOUT(ICELLS,LINE,LINEP,SW1,SW2, MARK,IMARK,XS200,ISTRIN,NSTRIN,DATAR,NSIZE)

Description This routine is used to output the tree structures and all the associated variables needed to specify the process of analysis completely,thus the process can be restarted from the information output.The List of Available Space and hence the tree structures (APPENDIX I) are expected to be within the Fortran list ICELLS whose dimension is NSIZE. The MINISLIP routines (APPENDIX I) use absolute addresses to link cells and tree nodes together,thus before the trees, List of Available Space and Fortran list,ITREES,of the root addresses are stored,the absolute address of the first location of the Fortran list ICELLS is assigned to the variable MARK tobe output,then the relative addresses of various other crucial locations are assigned to overwrite the absolute addresses prior to output.

Then the relevant Fortran lists and variables are output and control is returned.

Entry Conditions The Fortran lists ICELLS,ITREES and NTREES should contain the List of Available Space,the valid tree root addresses (see SETUP) and the valid tree names (again see SETUP),respectively.The other variables contain data pertinent to the matching process such that it can be continued from the position in which it was halted.

Exit Conditions Locations essential for the working of the matching process or any tree manipulation,such locations are AVSL (see APPENDIX I) and the Fortran list of tree root addresses,have been overwritten with relative addresses and thus routine TAPEIN must be used to replace the absolute addresses prior to any subsequent usage.

Subroutines Referenced

MINISLIP: MADOV,LNKL,LNKR

Referencing Routines

MP: the main routine MP1

RP: the main routine RP1

5. Identification I/05 TAPEIN(ICELLS,LINE,LINEP,SW1,SW2, MARK,IMARK,XS200,ISTRIN,NSTRIN,DATAR,NSIZE)

Description This routine inputs and modifies,if necessary, all the variables output by routine TAPOUT,such that the system is in exactly the same position as it was before routine TAPOUT was used;thus the trees can be manipulated as in the Restructuring Process or analysis can be continued as in the Matching Process.

The variable values are input and from the new absolute address of the first location in the Fortran list ICELLS, the relative addresses in some variables (assigned by routine TAPOUT) are replaced by new absolute addresses.Certain of the input values are immediately output,as a check.Finally control is returned.

Entry Conditions The input source,currently magnetic tape,must have the correct structure,as constructed by routine TAPOUT.

Exit Conditions All variables and Fortran lists are initialised for restructuring the trees or continuing analysing input statements from the position at which the analysis was previously terminated.

Subroutines Referenced

MINISLIP: MADOV,SETDIR,LNKL,LNKR

Referencing Routines

MP: the main routine MP1

RP: the main routine RP1

6. Identification I/06 STATS

Description This routine is used to output statistics of the current status of the Matching Process.The statistics include outputting all of the tree structures involved.

Entry Conditions Various Common Block variables are assigned values in the main routine,MP1 and routine PROCES, MP2.

Exit Conditions None.

Subroutines Referenced TPRINT

GP: SPACE,INTIST

Referencing Routines

MP: the main routine MP1

7. Identification I/07 ADD

Description This routine is used to add items to the trees.The input format is exactly the same as for routine SETUP (I/01),and is described in detail in APPENDIX III,ii). Control is returned when the characters "OOOOOE" are encountered in the first six columns of a data card.

Entry Conditions The variables ITREES and NTREES must be Fortran lists and contain the tree root addresses and names,respectively - as set in routine SETUP (I/01).

Exit Conditions None.

Subroutines Referenced

MINISLIP: NUCELL

Referencing Routines

RP: the main routine RP1

8. Identification I/08 DELETE (LNKLST)

Description This routine is used to delete items from trees.The item and the tree within which it is to be found are input in exactly the same format as that used by routine SETUP (I/01,and described in detail in APPENDIX III,ii).

If the item specified cannot be found upon the tree specified,then the following message is output:-

" NODE WITH ID='ID value of node' AND DATA CHARACTERS 'value of IDDATA FIELD' NOT ON TREE 'tree name' ",and the next data item is sought.

Control is returned by the occurrence of the characters "KENDEL" in the first six columns of a data card.

Entry Conditions The single argument,LNKLST,must contain a valid list address (APPENDIX I,4.).Variables NTREES and ITREES must be Fortran lists which contain the current tree names and root addresses,respectively.

Exit Conditions None.

Subroutines Referenced

MINISLIP: ID,INHALT,NEWTOP,LNKR,LNKL,SETIND,RCELL,NUCELL,NOKTOP,

GP: EMPTY,MACHIN LISTMT

Referencing Routine

RP: the main routine RP1

9. Identification I/09 CHANGE

Description This routine is used to alter the frequency count (see 2.7) associated with a specified item within the tree specified.The new value of the frequency count and the particular item within which it is to be inserted are specified by the input data (detailed format,see APPENDIX III,iii). The first data card must contain the number of cards to follow (in the first three columns),control is returned after this number of cards have been dealt with.

Entry Conditions There must be at least one data item present for this routine to input and this first item and any following ones must contain data in the correct format (details APPENDIX III,iii).

Exit Conditions None.

Subroutines Referenced

MINISLIP: ID,INHALT,LNKL,LNKR,SETIND

GP: MACHIN

MP: TRNAME

Referencing Routine

RP: the main routine RP1

10. Identification I/010 READ (LINE)

Description This routine is used to input one card image from the data stream of statements being processed. The argument,LINE is a Fortran list into which a character from each successive column of the card is read into successive locations,left adjusted.

Entry Conditions The argument LINE must be a Fortran list,dimensioned at least eighty.

Exit Conditions The Fortran list LINE contains the next card image as described above.

Subroutines Referenced None.

Referencing Routine

MP: the main routine MP1

11. Identification I/011 NUMODD (NROOT,RODITY,RITEM, AVLNGT,ALNMAT,LNKLST)

Description This routines examines the tree whose root address is passed in argument NROOT.It calculates and returns;in RODITY,the average oddity (see "Terminology" chapter 2) per item matched onto the tree,in RITEM,the

oddity per tree item as defined,in AVLNGT,the average length of a tree item as defined,and in ALNMAT,the average length of the items matched.

These values are output within an explanatory message by the routine,and further the average oddities per character, both matched and as defined are output.Finally the percentage saving in oddity per tree symbol is output and control is returned.

Entry Conditions The argument LNKLST must contain a valid list address (APPENDIX I),and NROOT must contain a tree node address.

Exit Conditions The four arguments RODITY,RITEM,AVLNGT and ALNMAT are overwritten as described above.

Subroutines Referenced

MINISLIP:ID,INHALT,LNKL,LNKR,NEWTOP,NOKTOP,LISTMT

GP: EMPTY

RP: NUMTRE

Referencing Routines

RP: the main routine RP1

APPENDIX II - Description of the routines

The Tree Building Routines:section TB

Index

1. Identification TBl NTGROW(NTNAME,NUMSEQ,ITYPE,IDDATA, LLDATA,LRDATA)

Description When data for the contruction of the trees is input in expanded format (one node per card,see APPENDIX III) the cards are sequenced to ensure that no cards are misplaced.This routine checks the sequence numbers which are received in the argument NUMSEQ.

If the sequencing is correct then the new node data which is received within the last four arguments,ITYPE,IDDATA,LLDATA, and LRDATA,is passed on to routine NTPUT;or if the value of NUMSEQ indicates that the current node data is the initial node of a new tree item then the tree root address is retrieved by the use of argument NTNAME and a call to routine TRNAME,and passed with the next node data to routine NTPUT. On return from routine NTPUT control is returned.

If the sequencing is found to be incorrect then an error message is output and processing is terminated.The error message is "INCORRECT DATA SETUP,CARD READ IS *** 'card image' AND PREV. SEQ, NO. 'previous sequence number' ".

Entry Conditions This routine is passed values within the six arguments which specify the next tree node that is possibly to be contructed.The value of the variable NUMSEQ is critical and must be less than the previous value,stored within variable NUMPRE,unless the previous value is "99" in which case the current value (in NUMSEQ) must be "1" and we are dealing with the first node of a new item.In the event of the value of NUMSEQ being "1" the value of the First argument, NTNAME,must be a valid tree name (see SETUP,I/O section).

Exit Conditions If there is within the specified tree already an identical node in an equivalent position to the proposed node,then no new node is constructed and the value of argument ITYPE will be set to zero (in routine SAMCEL).

If in fact a new node has been constructed (within routine NTPUT) then the value of ITYPE will remain unchanged but the contents of variable IDDATA will have been altered by the action of routine LFTORT,section GP (via MACHIN,section GP called from SETCEL).The argument NTNAME is overwritten.

Subroutines Referenced NTPUT

GP: MACHIN

MP: TRNAME

Referencing Routine

I/O: SETUP

2. Identification TB2 NTPUT(LAST,ITYPE,IDDATA,LLDATA,LRDATA)

Description This routine decides via routine SAMCEL if a new tree node is to be constructed with the current node data,within the arguments;if so the node is constructed via routine SETCEL.The address of the tree node within the specified tree which is to be set with the new data or compared with the new data if already set(as well as any alternatives),is passed to this routine as the contents of argument LAST.

If the current tree node is empty (that is ,the node whose address is in LAST) then the value of its ID field (APPENDIX I) will be zero and the node is set with the values of the last four arguments,ITYPE,IDDATA,LLDATA and LRDATA,via a call to routine SETCEL, which also raises a node by means of a call to routine NUCELL (section MINISLIP) and its address is inserted in the LINK-RIGHT ADDRESS field (APPENDIX I) of the newly set node,the address of this new successor node is assigned to variable LAST and control is returned.

If the current tree node is not empty then is contents are compared with the data for the proposed new node,via a call

to routine SAMCEL,to ascertain if they are identical,in which case the value of ITYPE on return from SAMCEL will be zero and no new node need be constructed,the variable LAST is assigned the value of the LINK-RIGHT ADDRESS field of the tree node that was found to be equivalent and control is returned.

If the current tree node does not prove to be the same as the proposed node then any alternatives to the current tree node are examined in succession by routine SAMCEL until; either an equivalent node is found,in which case the action of this routine is as described above;or the tree node currently being examined is not equivalent and has no alternative.Then a new cell is raised by means of a call to routine NUCELL (APPENDIX I,MINISLIP),its address is inserted in the LINK-LEFT ADDRESS field of the current tree node and control is transferred to routine SETCEL to set the data within this new node.On return from SETCEL,the variable LAST is assigned the address of the successor node (raised and linked,as described above,within routine SETCEL) to that just constructed and control is returned.

Entry Conditions This routine contains five arguments, the last four contain data for the construction of a new node if necessary,the first argument is used to pass the root address in which the following item is to be inserted at the start of each new item,from then on it contains the address of the tree node which is in the position of insertion the current node data.

Exit Conditions If the data passed to this routine is used for the contruction of a new node then the contents of the argument IDDATA will have been altered within routine SETCEL ;if no new node is constructed the value of variable ITYPE will have been reset to zero within routine SAMCEL.

In either case,the entry value of argument LAST will have been overwritten with the address of the new current node.

Subroutines Referenced SETCEL,SAMCEL

MINISLIP: ID,INHALT,LNKL,NUCELL,SETIND,LNKR

Referencing Routines NTGROW

3. Identification TB3 SAMCEL(ITYPE,LAST,IDDATA,LLDATA, LRDATA)

Description This routines compares the contents of the node whose address is in the argument LAST with the next node data items,ITYPE,IDDATA,LLDATA and LRDATA.The details of the comparison are dependent upon the value of ITYPE,or in other words,the value of the ID field of the tree node (for node structure is ID dependent,see APPENDIX III).

If the comparison proves to be positive then the value of ITYPE is reset to zero and control is returned;else control is just returned.

Entry Conditions Of the five integer arguments,the first ITYPE must contain a valid ID field value (depends upon particular implementation,see APPENDIX III),and the second LAST must contain a tree node address.

Exit Conditions The value of ITYPE is reset as zero if the comparison proves to be positive;else none of the arguments are altered.

If on entry an invalid value of ITYPE is detected,then the program is terminated subsequent to the outputting of an error message via a call to rottine KERROR.

Subroutines Referenced

MINISLIP: ID,INHALT,LNKL,LNKR

GP: KERROR,MACHIN

Referencing Routines NTPUT

4. Identification TB4 SETCEL(LAST,ITYPE,IDDATA,LLDATA, LRDATA)

Description This routine sets the node data passed to it within the arguments,ITYPE,IDDATA,LLDATA and LRDATA,in the node whose address is received as the value of the first argument,LAST.The node structure is determined by the value of ITYPE (APPENDIX III),if the value of ITYPE is invalid (again APPENDIX III),then an error message is output via a call to routine KERROR and processing is terminated.

If the value of ITYPE is not invalid,then control is directed by the value of ITYPE to construct the tree node whose address is in variable LAST.If the value of ITYPE is not that of a terminal node,then a new cell is raised and its address is inserted in the LINK-RIGHT ADDRESS field of the node just set and control is returned;else control is just returned.

Entry Conditions The contents of the first argument should be a tree node address and the contents of the second, a valid ID value.

Exit Conditions The contents of the third argument,IDDATA may be altered dependent upon the value of ITYPE.

Subroutines Referenced

MINISLIP: NUCELL,SETIND

GP:KERROR,MACHIN

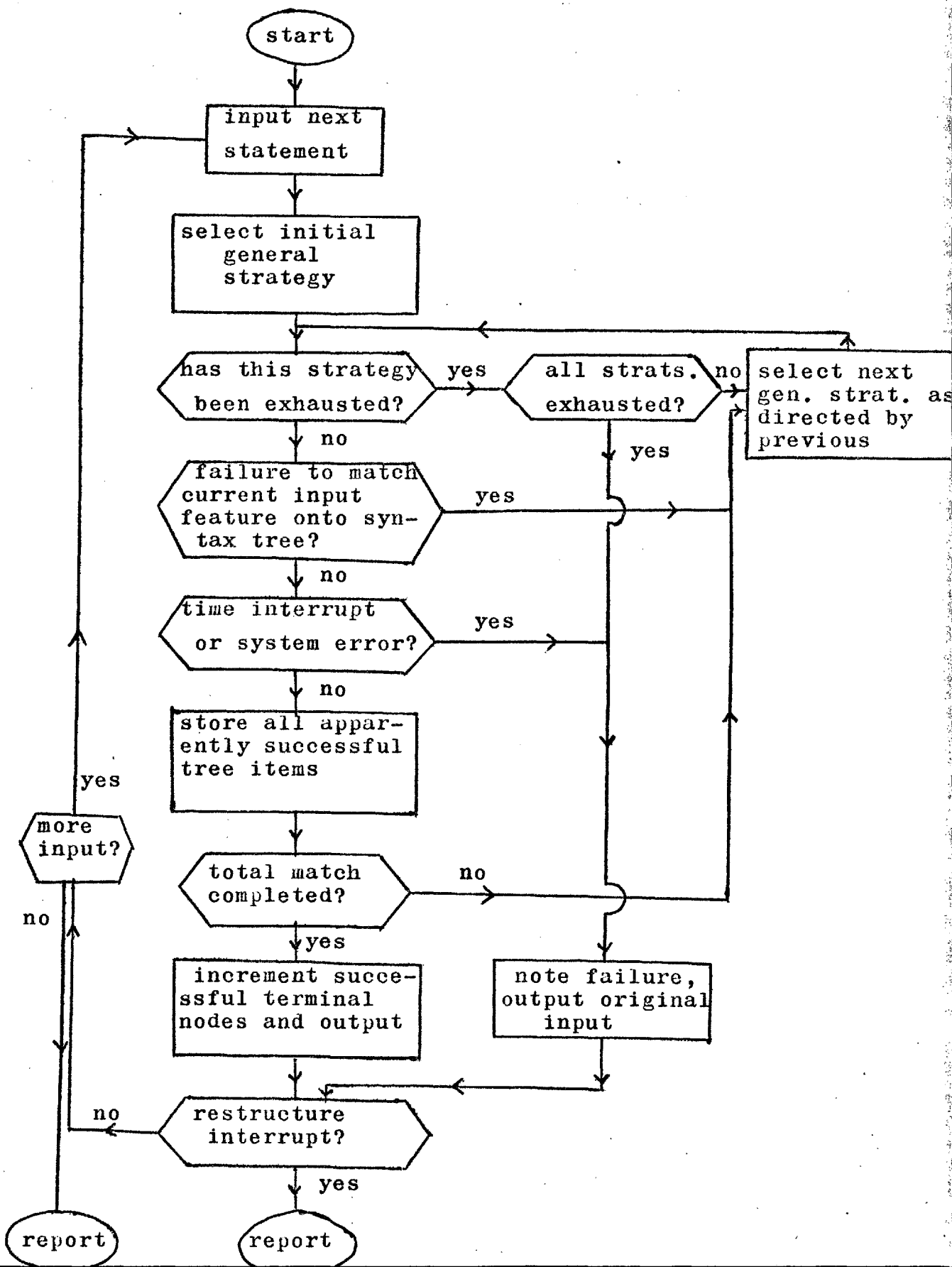
Referencing Routines NTPUT

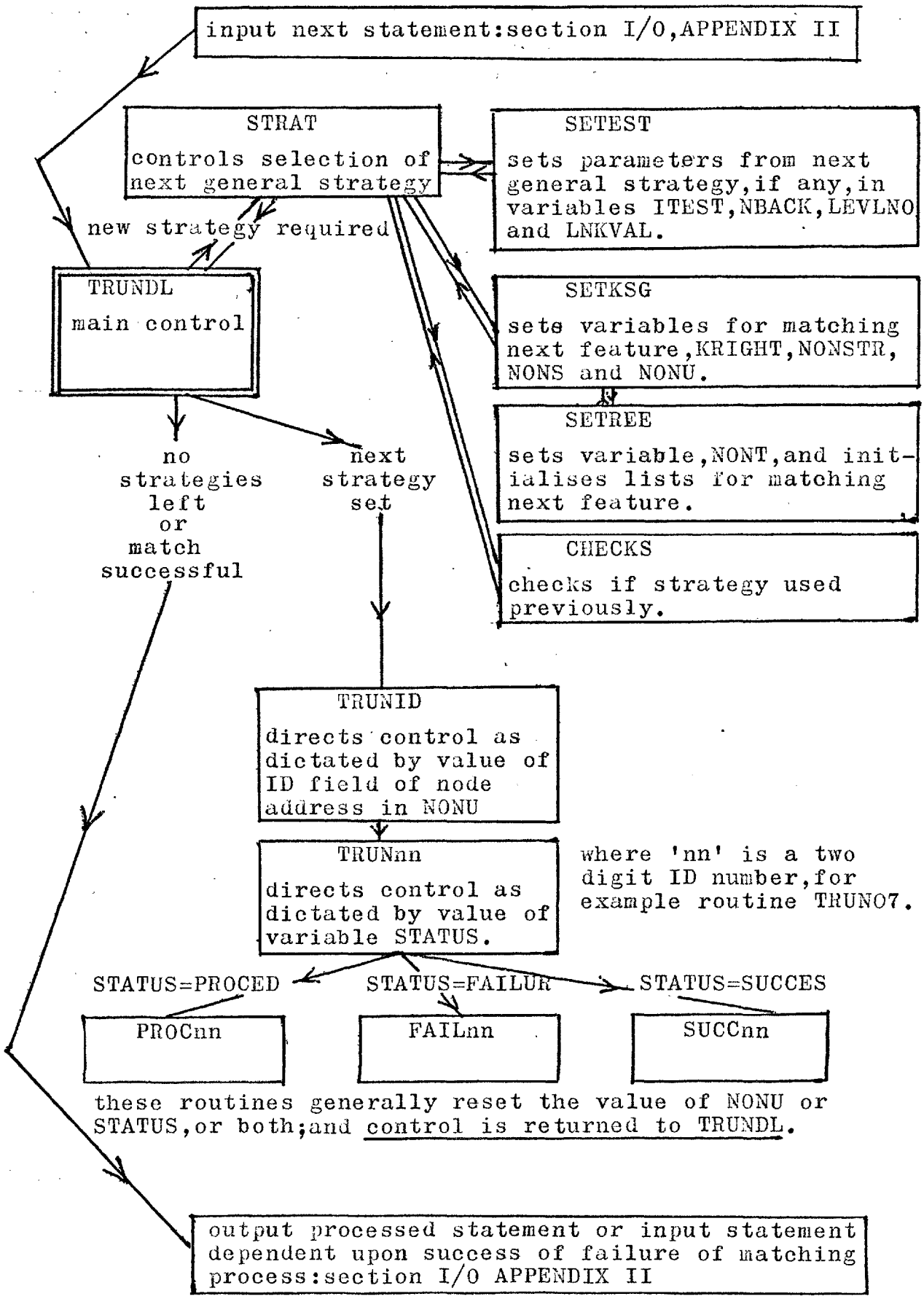
APPENDIX II - Description of the routines

The Matching Process Routines:section MP

Index

The logical structure of the analysis process





Flowchart of matching process, including main routine and variable names.

The Matching Process Routines;Section MP

1. Identification MP1 the main routine

Description An integer is read from the first data card into variable,NCARDS,if the value read is zero,the LAS (APPENDIX I) is initialised via a call to routine INITAS (in MINISLIP section) and the tree structures are constructed from card data via a call to SETUP (in I/O section).But if the value of NCARDS is not zero then the tree structures and all the associated variables are read from tape via a call to routine TAPEIN (in I/O section) and control is transferred immediately into the card processing loop to a position dictated by the value of one of the variables read,DATE.

The card processing loop obtains card images from routine READ (in I/O section) in the buffer,LINE,if this card image could validly follow the previous card image,in LINEP,then the previous card image is dealt with according toits type as follows;systems cards and data cards are just output,while the language (e.g. Fortran) cards are placed in another buffer, STRING.When a complete statement has been built up in STRING the buffer and two markers,NSTART and NSTRIN,which delimit the statement within the buffer,are passed to routine,PROCES, for processing and subsequent output (also from routine PROCES).

When the number of statements processed reaches some predefined limit (if NCARDS=0 then the value of LIMIT;else the value of NCARDS ,which has been transferred to LIMIT in routine TAPEIN),control is transferred to routine STATS,which outputs statistics and then to TAPOUT (in I/O section) which writes the current tree structure and relevant variable values on tape;finally via a call to routine SPACE (in GP section) the current number of cells in the LAS (APPENDIX I) is output.

Entry Conditions This must be the main routine,if the data card contains zero then it must be backed by a deck of

cards containing the correct layout of symbols for the construction of a set of trees (APPENDIX III);else an array containing the already initialised cells and hence tree structures plus certain other values must be available on a unit number 7 (again see APPENDIX III,for details).

Exit Conditions None,the complete matching process is terminated.

Subroutines Used PROCES,STATS

MINISLIP: INITAS

I/O: TAPEIN,TAPOUT,READ

GP: SPACE

Calling Routines None,this is main routine.

2. Identification MP2 PROCES(START,NSTRIN)

Description Routine receives a buffer,STRING,fromthe main routine containing a statement to be processed.The values of the arguments,START and NSTRIN,delimit that portion of the buffer which is to be processed.

If the end marker,NSTRIN,is marking a blank character then it is moved towards the beginning marker until either,a non-blank character is found,or it coincides with START in which case we have a null statement and control is returned.But, assuming the statement is not null,then the characters within buffer and within the delimiters ,are right adjusted (for compatability with single characters removed from tree nodes by the Function ID,see APPENDIX I).

The statement to be processed is examined and a decision made as to which main syntax tree to call on passing the buffer to routine TRUNDL for processing.

On return from TRUNDL the values of various flags are examined

to determine exactly what to output,for instance,the input statement is output unprocessed and preceeded by the appropriate messgge if an error or time trap has been activated during processing.

Finally,control is returned to the main routine.

Entry Conditions The values of the two arguments must contain integer values (note: variable START is integer) and the value of the second,NSTRIN,must be greater than or equal to the value of the first,START.

Exit Conditions None.

Subroutines Used TRNAME,TRUNDL,

MINISLIP: NOKTOP,LISTMT,ID,INHALT,RCELL

GP: EMPTY,MACHIN,SWPSTK

Calling Routine the main routine

3. Identification MP3 TRUNDL(STATUS,NSTART,NSTRIN,TNAME)

Description On entry to routine the lists (APPENDIX I) used are all emptied,the variables used during processing are initialised,the position in the buffer,STRING,of the current input string character is set in variable,NONS and the current tree node address in variable,NONU(the tree root address is obtained via routine TRNAME).

This initial enviroment (that is current character and tree node) is stored in list,NREPET,then via a call to routine STRAT,the general matching strategy to be used is obtained and used to assign the position in the buffer,STRING,of the actual character to be matched,in variable,NONSTR and the initial tree node address for this matching process in variable,NONT.Several other variables are initialised and lists emptied,then a call to routine TRUNID which directs the matching process to other routines dependent upon the

value of the ID field (APPENDIX I) of the current tree node.

On return from routine TRUNID,several flags are examined to direct control to one of several possible destinations.If the value of variable STATUS is equal to the value of variable YES (in future:if STATUS=YES),then we have a total failure to match the current input string against any tree item and control is returned;if TOTCRT=999.0 then a terminating error has occurred within the MINISLIP routines (APPENDIX I) and control is returned;if NTIMUZ $\geq$ NTIMUP,then the time limit for matching has been exceeded and again control is returned,similarly,if an error condition has occurred within the processing routines,STATUS=ERROR and control is returned.

If STATUS=ENDED but STREND$\neq$YES,then a terminal node has been found on a main tree branch but the input string has not been exhausted;so control is transferred to another section of the routine which resets the value of STATUS and the search is continued for alternative paths to examine.If STATUS=ENDED and STREND=YES then we have a total successful match and routine ICOUNT is entered the frquency count of each terminal node that was associated with this last matching of the complete input string.

Next the flag CONRAT is examined,if the value is negative then routine TRUNID is re-entered because the current input string character has not been matched and there are still nodes to be examined.If CONRAT=FAIL then a complete failure with the current general strategy is indicated and control is transferred to select a new general strategy and then continue matching;else the current string character has been matched and control is transferred to re-initialise several variables and empty several lists before continuing matching with the next current input string character and initial conditions that will have been set in routine SUUCOl.

Entry Conditions The second and third arguments, NSTART and NSTRIN, must contain integer values with NSTRIN greater than or equal to NSTART. The fourth argument must contain a tree name (APPENDIX III) and is used, via routine TRNAME, to obtain the tree root address.

Exit Conditions The first argument, STATUS, can be over-written with several possible values (see description of routine PROCES). If a processed string of characters is to be output on return to PROCES, rather than the original string, then the list ITRACE will contain the relevant information (see APPENDIX V for details of list structure) plus the Block Common variable IMARC.

Subroutines Used STEP, SETLIS, STRAT, TRNAME, TRUNID, ICOUNT

GP: EMPTY, SPACE

MINISLIP: NEWTOP

Calling Routines PROCES

4. Identification MP4 STRAT(STATUS, NONS, NSTART, NSTRIN, LNKVAL)

Description This routine controls the initialisation of various variables for the next matching of an input feature. Which variables and what values are set depends on the current general strategy (selected via call to routine SETEST) and the history of the matching process, with the current input statement.

The routine can be entered in one of two possible situations; one, the previous matching was successful and two, the matching was unsuccessful.

The list, NONENV, is used to store the recent history of the matching process when the system is 'backtracking' and trying

to re-match a feature that had been apparently successfully matched,previously.Thus if the re-match is successful this subsequent history stored in list NONENV is discarded (in routine ICOUNT) .But if the re-matching is unsuccessful then on entry to this routine (STRAT) the contents of list NONENV are restored to list NREPET,which contains the complete history of the matching process with the current input string. Also,some of NONENV's items are restored to list ITRACE which contains information for the processed string to be output on return to routine PROCES.

Then the routine SETEST is entered to obtain the next general strategy,inwhich case STATUS will be returned with the value PROCED;or signal that there are no more general strategies to be tried,then returns STATUS=YES.The third possibility on return from SETEST is that an error has occurred and in this case,on return STATUS=ERROR.Control is returned to routine TRUNDL unless STATUS=PROCED.

If control is not returned then routine SETKSG is entered. On return,the details of the string feature to be matched and the contents of the various lists (APPENDIX V for details) have been initialised,else an error condition has occurred and control is returned,to routine TRUNDL.

But if an error condition has not been activated,a check is made that the actual input string character to be matched (STRING(NONSTR)),does not preceed the first input string character;if it does not we prpceed else control is transferred to the beginning of the routine to examine the next general strategy,if any.

Finally,routine CHECKS is entered to determine if the proposed matching process has been used previously,if not,control is returned to STRAT.If the initial conditions for matching

have been,apparently successfully,used previously,then in routine CHECKS the previously final apparent success conditions are reset,STATUS is assigned the value of the variable FAILUR and control is returned to STRAT;thus the matching process will be continued from the position in which it was terminated previously.The other possibility on return from CHECKS is that the proposed matching has been attempted previously and failed,in which case STATUS is again assigned the value of variable FAILUR and KTAG is assigned the constant "9999",then control is transferred to obtain the next general strategy,if any.

Entry Conditions The real argument,STATUS,is used to pass 'success',or 'failure' information to and from the routine. The four remaining arguments are integer,NSTART and NSTRIN mark the position within the buffer STRING,of the character string to be processed,both must be non-zero and NSTRIN must not be less than NSTART.The second argument,NONS,must contain a non-zero integer which marks the position within the buffer STRING,of the beginning of the next input feature to be matched as set directly after the last feature was matched.The last argument must contain a zero or positive integer;none of the integer arguments can be negative.

Exit Conditions The returned value in the argument STATUS is the most important quantity as if it has been overwritten with the values of either ERROR or YES then complete failure to select a suitable general strategy is indicated and the processing of the current statement will be terminated.If neither of these cases holds the variables and list structures for an attempted matching will have been initialised;The value of the second argument NONS may have been overwritten and also that of the fifth argument LNKVAL,and the Block Common

variables,NONT,NONU,KRIGHT,IRIGHT,LEVLNO and NONSTR.

Routines Used CHECKS,SETEST,SETKSG

MINISLIP: LISTMT,NOKTOP,NEWTOP

GP: SWPSTK,KERROR

Calling Routines TRUNDL,SUCCOl

5. Identification MP5 SETEST(STATUS,ITEST,LEVLNO,LNKVAL, NBACK)

Description This routine selects the next general strategy to be used (which may be the same as the last) or,if there are no further general strategies to use,assigns the value of the variable YES to the first argument,STATUS,and returns control.

In the event that the tree of general strategies has not been exhausted,then the next strategy to be used is dependent upon the status of the matching process on entry to this routine.If an input feature has just been successfully matched (the value of STATUS = FAILUR) then the top strategy in the tree of strategies is selected;else the strategy whose address is in variable LASTES,is selected.

Having selected a new current general strategy,the variable NTEST is assigned its address and variable LASTES is assigned the contents of its LNKL ADDRESS field (APPENDIX I).The arguments,ITEST,LEVLNO,LNKVAL and NBACK are assigned values from the nodes of the now current general strategy and control is returned with STATUS assigned the value of PROCED.

This routine contains one error trap which will be activated and an error message output (via KERROR GP) if the ID field (APPENDIX I) of the first node of the general strategy does not contain integer '5'.

Entry Conditions The first argument,STATUS, must be real and the other four,ITEST,LEVLNO,LNKVAL and NBACK,all integer.

Exit Conditions On exit the value of STATUS will have been set to ERROR (if the error trap was activated),YES or PROCED;if the value of the latter variable then the other four arguments will have been overwritten with new values.

Subroutines Used TRNAME

MINISLIP: ID,INHALT,LNKL,LNKR

GP: KERROR,MACHIN

Calling Routine STRAT

6. Identification MP6 SETKSG(STATUS,ITEST,LEVLNO,NONS, NBACK,NSTART,NSTRIN)

Description This routine modifies the value of the variable NONS,which marks the position within the buffer,STRING, of the beginning of the next input feature proposed for matching.The value is only in fact modified if the strategy selected (by means of a prior entry to routine SETEST) is a 'back-tracking' strategy and thus the value received by this routine in the fifth argument,NBACK,is greater than zero.

Then routine SETREE is entered and on return the initial tree node variable,NONT,and the list structures (APPENDIX V) will have been initialised for the start of the next matching process;the value of the variable NONS,may again have been changed dependent upon the positions within the buffer STRING of the actual characters matched during the preceeding matching of input string features.The current tree node address is assigned to variable NONU from variable NONT.

Finally the position in the buffer,STRING,of the actual character to be matched,NONSTR,and the variable which determines how many successive specific tree node characters are to be

assumed correct during the next matching process,KRIGHT,are both assigned values dependent upon the values of ITEST,LEVLNO and NONS.

Entry Conditions The value of the second argument,ITEST, must be integer,one,two or three.The final five arguments must contain zero or positive integer values.

Exit Conditions The first argument,STATUS,will contain the value of variable PROCED,as assigned via the call to routine SETREE;or the value of variable ERROR assigned after the activation of an error trap in SETREE or SETKSG.The value of the fourth argument,NONS,will possibly have been overwritten in SETREE and/or SETKSG.

Subroutines Used SETREE

GP: KERROR

Calling Routine STRAT

7. Identification MP7 SETREE(STATUS,NONS,NSTART,NSTRIN)

Description This routine assigns the address of the tree node from which the next matching process will begin,to variable,NONT.The variable NONS,which contains the lower bound in the buffer,STRING,of the next input string feature to be matched,is set initially in routine STEP via a call from routine SUCCO1;subsequently,the value is possibly modified in routine SETKSG and possibly again in routine SETREE by the following means.

The list NREPET contains (among other things,see APPENDIX V) the past values of NONSTR (the position within buffer,STRING, of the actual characters matched),each with its initial enviroment for matching (i.e. structure of list NONUST,etc.).Routine SETREE searches down this list until value of NONSTR (NONMAT is the actual variable name used within this routine) is found

which is less than the current value of NONS.Then NONS is reset via a call to routine STEP,as the position of the first relevant character within buffer,STRING,past the position indicated by the last value of NONSTR uncovered.

The list structures are manipulated as follows;all items removed from the past history list,NREPET,while searching for the position of the last character matched which is less than the current value of NONS,are stored in list NONENV.Thus if the subsequent matching process is successful the contents of list NONENV are destroyed (in routine ICOUNT);if unsuccessful list NONENV is used to completely restore the contents of list NREPET (within routine STRAT) in preparation for the next matching to be attempted.When NONS has been reset,copies of the initial enviroment for matching,that is the contents of lists NONUST and INONUS,are stored in list ID6STK (for they will be altered in lists NONUST and INONUS during the actual matching process) and also copied into list NONENV.

Entry Conditions Variables NREPET,ID6STK,NONUST,INONUS and NONENV must contain valid list addresses (APPENDIX I),and list NREPET must have the correct,somewhat complex,structure described in APPENDIX V.The argument NONS must contain a positive integer value intermediate between the integer values of the third and fourth arguments,NSTART and NSTRIN.

Exit Conditions On exit the value of NONS will possibly be modified.If no error conditions have been activated then STATUS will contain the value of variable PROCED;else it will contain the value of ERROR.

Subroutines Used STEP

MINISLIP: NOKTOP,LISTMT,NEWTOP

GP: EMPTY,SWPSTK,STOREN,KERROR

Calling Routine SETKSG

8. Identification MP8 CHECKS(STATUS,ITEST,LEVLNO,NONS)

Description This routine determines if the matching process that we are about to try has been tried previously.

The list,CHKLST,is scanned for a set of variables (NONT, NONSTR,LEVLNO,ITEST) that are all equal to the values of the current set.If no such set is found,the current values are stored in list CHKLST,the value of the variable NUMSET is incremented by one and also stored in this list to identify the set in case of a later re-use of this set (see below).Control is returned.

If an identical set is discovered,that is,the current proposed matching has been attempted previously;then the result of the previous attempt and complete final conditions (i.e. list structures and variable values) are located within list,ICHKON,via the stored value of NUMSET (set of final conditions and appropriate value of NUMSET stored in list ICHKON by routine STOSET).

If the current set of initial conditions has been used previously with apparent success (KTAG=9000),then the stored final conditions are reset;the variable STATUS is assigned the value of FAILUR;the address of the particular strategy that gave rise to these initial conditions on the previous occassion is replaced within list ID6STK,by the address of the current strategy (from variable NTEST).The current tree node address variable,NONU,is also reset from list ICHKON, and control is returned.

But,if the current set of initial conditions has been used unsuccessfully before (KTAG=9999);the set number and value of KTAG are replaced;STATUS is assigned the value of FAILUR and control is returned to select another strategy,if any.

Entry Conditions The last three arguments plus the Common Block variable,NONT,must contain integer values.The variables ICHKON,NONUST,ID6STK,IBRNCH,ISKIP,INONUS,NID6ST, CHEX and CHKLST must all contain,integer valid list names (APPENDIX I).

Exit Conditions If the current set of initial conditions has not been used before then they are stored in list CHKLST, and control is returned.

If not,then the value of FAILUR is assigned to the first argument,STATUS and the variable KTAG is assigned the value "9000" or "9999",dependent upon whether the previous usage of the current matching conditions was terminated successfully or unsuccessfully.If the returned value of KTAG (via Common Block) is "9000" and the complete matching conditions will have been overwritten with new values.

Subroutines Used

MINISLIP: LIST,NEWTOP,LISTMT,NOKTOP,NOKBOT

GP: KERROR,EMPTY,SWPSTK,NBOTOP

Calling Routine STRAT

9. Identification MP9 TRUNID(STATUS,NSTRIN,NONS,LNKVAL, NSTART)

Description This routine examines the ID field (see APPENDIX I) of the current tree node,whose address is contained in variable NONU.

If the value is not less than one or greater than ten, control is directed to routine TRUNnn,where 'nn' is the value of the current node ID field.

If the value is outside the range specified above then an error condition is activated,a message is output (via a call to routine KERROR) and control is returned.

Entry Conditions The value of the ID field of the node whose address is the value of the Common Block variable NONU must be between one and ten.

Exit Conditions The most important of all the possible exit conditions are as follows.

The first argument STATUS may have been assigned the value of variable PROCED or SUCCES or FAILUR or ERROR or YES or ENDED (all in Common Block /MODES/),and/or the value of NONU may have been overwritten.

The variable STREND will contained the value of YES if the input string has been exhausted.

The variable CONRAT will have been assigned the value "999.0" if all the possible ways to match the current input string character (STRING(NONSTR)) using the current strategy have failed;or the value of CONRAT may be positive (it is initialised negative in routine TRUNDL) if the current input string character has been successfully matched.

Subroutines Used TRUN01,TRUN02,TRUN06,TRUN07,TRUN10

MINISLIP: ID,INHALT

GP: KERROR

Calling Routine TRUNDL

10. Identification MP10 TRUN01(STATUS,NSTRIN,NONS,LNKVAL, NSTART)

Description This routine directs control to one of three routines depending upon the value of STATUS,which should be equal to the value of variable SUCCES or FAILUR or PROCED;else an error trap is activated,a message is output (via a call to routine KERROR) and control is returned.

Entry Conditions The value of the first argument,which is real,must be equal to one of the three variables SUCCES,

PROCED or FAILUR.

Exit Conditions On exit the argument STATUS may contain the value of PROCED (assigned in FAIL01 or SUCC01),or FAILUR (assigned in FAIL01 or PROC01),or SUCCES (assigned in PROC01), or ERROR (assigned in KERROR directly or via SUCC01 or PROC01).

The variable CONRAT may have been assigned the value "999.0" in routine FAIL01 or a lesser positive value in routine PROC01.

Variable NONU may have been overwritten in SUCC01 or FAIL01.

Subroutines Used PROC01,SUCC01,FAIL01

GP: KERROR

Calling Routine TRUNID

11. Identification MP11 FAIL01(STATUS,LNKVAL)

Description This routine assigns the value of the LINK-LEFT ADDRESS field (APPENDIX I) of the current tree node,to variable NONU,as the new current tree node address.

If the value assigned is less than or equal to zero then it is not a valid node address and hence an attempt is made to obtain the next tree node address by a further searching described below;else variable NONU contains a valid tree node address.If the value of the LINK-RIGHT DATA field of this node is not less than the value of the second argument,LNKVAL the variable STATUS is assigned the value of PROCED and control is returned;else a search for a new current node address is carried out as follows.

If the current matching strategy is not matching every input and tree node data character explicitly then the addresses of the nodes which have been assumed correct,or 'skipped' (see 4.22, example analysis including the 'skipping' procedure) and contain a further node address within their LINK-LEFT ADDRESS

field,have been stored (in list IBRNCH) and are reset by a call to routine BRANCH.If NONU is not reset within BRANCH,on return the value of STATUS is equal to that of FAILUR,then the list NONUST is examined.If list NONUST is not empty,NONU is reset as the top item and control is returned;if the list is empty then list ID6STK is examined.

If list ID6STK is not empty,NONU is reset from this list, else the variable CONRAT is assigned the value "999.0",to indicate a complete failure with the current matching strategy,and control is returned.

Entry Conditions The value of the first argument STATUS must be real and equal to that of FAILUR.The second argument LNKVAL must contain an integer value.

Exit Conditions If the value of the Common Block variable CONRAT is "999.0" then the values contained in the other variables are not important as the current matching strategy has failed completely.Similarly,if the value of STATUS is equal to that of ERROR (possibly set via the call to routine BRANCH);else the value of STATUS is equal to that of PROCED or FAILUR and the current tree node address variable will have been assigned a new value.

Subroutines Used BRANCH

MINISLIP: LNKL,INHALT,LNKR,LISTMT,NOKTOP,ID,NEWTOP

Calling Routine TRUNO1

12. Identification MP12 SUCCO1(STATUS,NSTRIN,NONS,NSTART, LNKVAL)

Description This routine is entered when an input string character has been explicitly matched against the character contained in the ID data field (APPENDIX I) of the current tree node;or,the match has been tentatively assumed,that is,

the total matching of the current input feature is 'skipping' input string characters,assuming that the first ones encountered are correct,prior to a subsequent attempt at a specific match.

First the variable STATUS is assigned the value of PROCED. Then we ascertain which of the two above described cases holds by comparing the values of the two Common Block variables, IRIGHT and KRIGHT.

In the latter case,the value of IRIGHT is less than that of KRIGHT.Then the address of the current tree node is stored in list ISKIP,to be discarded if the subsequent attempt at a specific match fails and to be transferred to list ITRACE if successful.The value of IRIGHT and a marker "-2" are placed on the top of list ID6STK to delimit the count node addresses associated with each input string character processed.Then the count of the current number of tree node characters 'skipped',IRIGHT,is increased by one and control is returned.

In the case of a genuine match between the current input string character and the current tree node data character, routine IMATCH is called and entered.On return to SUCC01, the variables and lists will have been initialised ready for matching the next input string feature.Further initialisation is now performed;the position within the buffer STRING,from which the attempted matching of the next input string feature will begin is set in variable NONS,via a call to routine STEP. The address of the initial tree node for this next matching is set in variable NONT,as the LINK-RIGHT ADDRESS (APPENDIX I) field of the tree node which contained the last character matched,NONU.Then provided no error traps have been activated routine ICOUNT is entered to transfer the count node addresses associated with the previous character matched,from list

ID6STK to list NREPET.On return from ICOUNT,routine STRAT is entered to select the next matching strategy and perform the final initialisation of list and variables for the next input feature matching.Finally control is returned.

Entry Conditions On entry the value of the argument STATUS must be real and equal to that of variable SUCCES,the ID field of the current tree node,whose address is in variable NONU,has the value "1".The variable IRIGHT must contain an integer value less than or equal to that of KRIGHT.

Exit Conditions On exit the argument STATUS will contain a value equal to that of ERROR or PROCED;if the latter value then the variable NONU will have been reset to a new current tree node address.If the value of IRIGHT was equal to that of KRIGHT on entry then all the relevant variables and lists will have been initialised for matching the next input string feature,via calls to routines ICOUNT and STRAT.

Subroutines Used IMATCH,STEP,ICOUNT,STRAT

MINISLIP: NEWTOP,LNKR,INHALT

GP: KERROR

Calling Routine TRUNO1

13. Identification MP13 PROCO1(STATUS,LNKVAL)

Description By a comparison of the values of variables IRIGHT and KRIGHT a decision is made to direct control to one or other of the two halves of this routine.

If the values are equal then two specific characters are to be compared,these are,the current input statement character in variable,STRING(NONSTR),and the data character within the current tree node,whose address is in variable NONU.Before this attempted match the value of the variable STREND is

is tested;if it is equal to the value of the variable YES then the input statement has been exhausted and control is returned with the variable STATUS assigned the value of variable FAILUR; else the two characters are compared.If the characters are not equivalent control is returned with the value of STATUS reset as that of FAILUR;else control is returned with the value of STATUS reset as that of SUCCES.

If on entry the values of IRIGHT and KRIGHT are not equal, then tree node characters are currently being 'skipped',that is,the data character contained within the current tree node is tentatively assumed to be correct without examining the current input string character to determine if they are equal. First,a check is made that the value of IRIGHT is less than that of KRIGHT,if it is not,then an error message is output via a call to routine KERROR,STATUS is assigned the value of ERROR and control is returned.If the error trap is not activated then the LINK-LEFT ADDRESS field (APPENDIX I) of the current tree node is examined;if it is less than or equal to zero,control is returned with the value of STATUS assigned that of the variable SUCCES;else there is an alternative node to examine.

If the contents of the LINK-RIGHT DATA field of this alternative node (APPENDIX I) are less than the value of the argument LNKVAL,control is returned with STATUS assigned the value of variable SUCCES.Else the frequency of past successful matches with this node is greater than the minimum specified in the current matching strategy and hence a complete description of the current matching position is stored in list IBRNCH - so that if the assumed current tree node proves later to be incorrect then the alternatives can be examined.

Finally,control is returned with the value of STATUS reset as that of SUCCES.

Entry Conditions The value of the variable IRIGHT must not be greater than the value of KRIGHT.The variable NONU must contain the address of a tree node and variable IBRNCH a valid list address (APPENDIX I).

Exit Conditions The value of the argument STATUS may equal to that of SUCCES or FAILUR or ERROR;if it is equal to that of SUCCES then the list IBRNCH may have had a set of items added to it.

Subroutines Used

MINISLIP: ID,INHALT,LNKL,LNKR,NEWTOP

GP: KERROR,STOINV

Calling Routine TRUNO1

14. Identification MP14 TRUNO2(STATUS,LNKVAL)

Description This routine directs control to one of three routines dependent upon the value of STATUS.If the value is not equal to that of PROCED,SUCCES,FAILUR or ENDED then an error message is output via a call to routine KERROR,STATUS is assigned the value of ERROR (within KERROR) and control is returned.

Entry Conditions The value of the first argument STATUS must be real and equal to that of one of the four variables cited above.

Exit Conditions There are many possible combinations of variable values and list structures,dependent upon the particular routine to which control is directed.

Subroutines Used PROCO2,SUCCO2,FAILO2

GP: KERROR

Calling Routine TRUNID

15. Identification MP15 SUCCO2(STATUS)

Description The contents of the variable NONU - the address of the current tree node - are placed on the top of list ID6STK.

If the value of variable STATUS is equal to that of ENDED,or the list NONUST is empty in which case STATUS is assigned the value of ENDED;and,the value of STREND is not equal to that of YES,then the top item on list ID6STK is discarded and control is returned.If the value of STREND is equal to that of YES then control is returned without discarding this top item.

The other possibility is that the value of STATUS is not equal to that of ENDED and the list NONUST is not empty.In this case the top item is removed from list NONUST and set as the value of NONU,which should be a tree node address.If the ID field (APPENDIX I) of this node has the value "7", then the variable ID7TAG is reset as the top item in the list INONUS,and control is returned.

Entry Conditions The variables NONUST,ID6STK and INONUS must contain valid list addresses (APPENDIX I).

Exit Conditions If the value of the argument,STATUS, remains the same as that of SUCCES then variable NONU will have been assigned a new tree node address from list NONUST, and variable ID7TAG may have been assigned a new value from list INONUS.Else the value of STATUS is that of ENDED and the values of the variables and the list structures will be unchanged.

Subroutines Used

MINISLIP: NEWTOP,LISTMT,NOKTOP,ID,INHALT

Calling Routine TRUNO2

16. Identification MP16 FAIL02(STATUS,LNKVAL)

Description If value of the LINK-LEFT ADDRESS (APPENDIX I) field of the current tree node - whose address is in variable NONU - is greater than zero and the frequency of past successful matches with this node is not less than the value of the second argument LNKVAL;then the variable NONU is is assigned the value of this LINK-LEFT ADDRESS field,STATUS is assigned the value of variable PROCED and control is returned.

Else an attempt is made to reset NONU from the list ID6STK if this is unsuccessful the variable CONRAT is assigned the value "999.0" and control is returned.

If the variable NONU is re-assigned from the list ID6STK and the value of the ID field (APPENDIX I) of the node whose address is now in variable NONU is not equal to "7",control is returned;else the variable ID7TAG is assigned the top item of list NID6ST and control is returned if the list NONUST is empty.

If the list NONUST is not empty and the value of the LINK-RIGHT ADDRESS field (APPENDIX I) of the top item is equal to the value of the variable NONU,then the top item the list NONUST and from the list INONUS are both discarded and control is returned;else control is just returned.

Entry Conditions The variables ID6STK,NID6ST,NONUST, INONUS and ISKIP must contain valid list addresses (APPENDIX I).The second argument LNKVAL should contain an integer value.

Exit Conditions The variable NONU will have been overwritten with a new value and the value of STATUS will be equal to that of FAILUR or PROCED;else STATUS may be equal to that of ERROR or CONRAT may contain "999.0".

Subroutines Used

MINISLIP: LNKL,INHALT,LNKR,LISTMT,NOKTOP,ID,NEWTOP

GP: KERROR

Calling Routine TRUNO2

17. Identification MP17 PROCO2(STATUS)

Description This routine assigns the value of variable SUCCES to the argument STATUS and control is returned.

Entry Conditions None.

Exit Conditions The argument STATUS is overwritten with the value of SUCCES.

Subroutines Used None.

Calling Routine TRUNO2

18. Identification MP17 TRUNO7(STATUS,LNKVAL)

Description This routine directs control to one of four routines,dependent upon the value of the first argument STATUS and returns control immediately it is returned from the particular routine selected.

Entry Conditions The argument STATUS is expected to have a real value which is equal to the value of variable PROCED,SUCCES or FAILUR;else an error trap is activated.

Exit Conditions Many variables may have been over-written and list structures altered dependent upon which routine is entered.

Subroutines Used PROCO7,SUCCO7,FAILO7

GP: KERROR

Calling Routine TRUNID

19. Identification MP19 PROC07(STATUS,LNKVAL)

Description First,the values of the variables IRIGHT and KRIGHT are examined to determine if the matching of specific characters is to be attempted or the first tree node data character encountered will be tentatively assumed correct.

If the value of IRIGHT equals that of KRIGHT then specific characters are to be matched;the contents of variable NONU are placed on the top of list NONUST and the value of "-3333" on top of list INONUS.Then via a call to routine TRNAME the root address of the subtree,whose name was contained within the current tree node on entry,is returned in variable NONU and control is returned.

If the values of IRIGHT is not equal to that of KRIGHT and the value of the LINK-LEFT ADDRESS field (APPENDIX I) of the current tree node is greater than zero - that is,the node has an alternative - then this alternative node is examined; if the frequency of past successes with this node is not less than the value of the second argument,LNKVAL,then a copy of the current matching process is stored in list IBRNCH in case the tentatively assumed match fails and the process backtracks to this point later.The contents of variable NONU are stacked in list NONUST,etc. as described above,and control is returned.

Entry Conditions The variables IBRNCH,ID6STK,NONUST, NID6ST and INONUS must contain valid list addresses (APPENDIX I).

Exit Conditions The entry value of the variable NONU will have been placed on top of list NONUST and then over-written with a new value.The list IBRNCH may contain a copy of the current matching process.

Subroutines Used TRNAME

MINISLIP: LNKL,INHALT,LNKR,NEWTOP,ID

GP: STOINV

Calling Routine TRUNO7

20. Identification MP20 SUCCO7(STATUS)

Description The current tree node is examined for the presence of a successor node,if not found an error message is output via a call to routine KERROR and control is returned.

If a successor node is found,the current node address (in variable NONU) and its corresponding tag (in variable ID7TAG) are added to the tops of lists ID6STK and NID6ST respectively, then the variable NONU is assigned the address of the successor node,STATUS is assigned the value of PROCED and control is returned.

Entry Conditions The variable NONU should contain the address of a tree node which has a successor.

Exit Conditions The variable NONU will have been overwritten with the address of the successor node and the argument,STATUS will contain the value of the variable PROCED. The lists ID6STK and NID6ST will both contain one more item.

Subroutines Used

MINISLIP: INHALT,LNKR,NEWTOP

GP: KERROR

Calling Routine TRUNO7

21. Identification MP21 FAILO7(STATUS,LNKVAL)

Description The value of variable ID7TAG is tested to determine if any characters have been matched within the corresponding subtree (whose address is in variable NONU),

If not,that is the value of ID7TAG is not "-9999",then we can reset NONU as an alternative node address,if any are present. If no valid alternatives exist then via a call to routine BRANCH,a search is made for any for any valid alternatives that might be associated with nodes processed earlier in the attempted matching of the current input statement feature.

If an alternative node address is found,it is assigned to variable NONU,STATUS is assigned the value of PROCED and control is returned;else the list NONUST is inspected.

If this list is not empty,the top item is removed and assigned to variable NONU.If the value of the ID field (APPENDIX I) of the node whose address is now in NONU is equal to "7",then the variable ID7TAG is reset as the top item from list INONUS before control is returned.

If the list NONUST is empty or the initial test on ID7TAG proved negative,the values of variables NONU and ID7TAG are stored on top of lists NONUST and INONUS respectively;then list ID6STK is examined.

The variable NONU is reset with a tree node address from list ID6STK,if possible,if not variable CONRAT is assigned the value "999.0" and control is returned.

Entry Conditions The variables NONUST,INONUS,ID6STK, NID6ST and ISKIP must contain valid list addresses (APPENDIX I). The second argument LNKVAL should contain an integer value.

Exit Conditions There are too many combinations of exit conditions to enumerate them all specifically,some of the more important ones are as follows.If the argument STATUS contains the value of variable ERROR then an error trap has been activated.If the Common Block variable CONRAT contains the value "999.0" then the current matching strategy has failed completely.

Subroutines Used BRANCH

MINISLIP: LNKL,INHALT,LNKR,LISTMT,NOKTOP,NEWTOP,ID

GP: KERROR

Calling Routines TRUN07

22. Identification MP22 TRUN10(STATUS,NSTRIN)

Description This routine directs control to one of three routines dependent upon the value of the argument STATUS.If the value of STATUS is not equal to that of variable PROCED or SUCCES,then an error message is output via a call to routine KERROR and control is returned.

Entry Conditions The first argument STATUS should contain a real value which is equal to one of the two variables indicated above.

Exit Conditions If an error trap has been activated then the value of STATUS will be equal to that of variable ERROR. The values of the variables STREND,NONU and STATUS may be overwritten dependent upon the routine entered.

Subroutines Used PROC10,SUCC10

GP: KERROR

Calling Routine TRUNID

23. Identification MP23 PROC10(STATUS,NSTRIN)

Description The routine places the addresses of any specific character nodes that may have been 'skipped' - that is,tentatively assumed to be correct - and are therefore in list ISKIP,in list ITRACE.A marker,variable IMARC,is then set to indicate up to what point in the buffer STRING the input statement features have been processed.Control is returned.

Entry Conditions The Common Block variables ITRACE and ISKIP must contain valid list addresses (APPENDIX I).The second argument NSTRIN should contain an integer value which marks the end of the string of characters being processed within the buffer STRING.

Exit Conditions The Common Block variable STREND is assigned the value of variable YES,and STATUS is assigned the value of SUCCES.

Subroutines Used

MINISLIP: LISTMT,NOKBOT,NEWTOP

Calling Routine TRUN10

24. Identification MP24 SUCC10(STATUS)

Description The value of the LINK-RIGHT ADDRESS field (APPENDIX I) of the current tree node is examined.If it is greater than zero,variable NONU is reset as its value,STATUS is assigned the value of variable PROCED and control is returned.

If the value of this LINK-RIGHT ADDRESS field is less than or equal to zero then this node has no successor and an error trap is activated.An error message is output via a call to routine KERROR and control is returned.

Entry Conditions The Common Block variable NONU should contain the address of a tree node which has a successor node.

Exit Conditions On a normal exit the variable NONU is reset and the argument STATUS contains the value of variable PROCED.If the error trap has been activated,the value of NONU remains unchanged and STATUS contains the value of variable ERROR,assigned within routine KERROR.

Subroutines Used

MINISLIP: LNKR,INHALT

GP: KERROR

Calling Routine TRUN10

25. Identification MP25 TRUN06(STATUS,NONS,NSTART,NSTRIN, LNKVAL)

Description This routine directs control to one of four routines dependent upon the value of the first argument,STATUS. Control is returned immediately on return from the particular routine invoked.

Entry Conditions The value of the variable STATUS should be equal to that of SUCCES,FAILUR or PROCED;else an error message is output via routine KERROR.

Exit Conditions The various possible exit conditions are explicitly described within each of the referenced routines.

Subroutines Used PROC06,FAIL06,SUCC06

GP: KERROR

Calling Routine TRUNID

26. Identification MP26 PROC06(STATUS,LNKVAL,NUMJMP)

Description This routine searches list ISKIP and then if necessary list ITRACE to determine the numerical sum of the last set of successive digits matched,which is assigned to the argument NUMJMP.STATUS is assigned the value of variable SUCCES and control is returned.

If current input string character does not match the data character within the current tree node or the input string of characters is exhausted,STATUS is assigned the value of FAILUR and control is returned.

There is one error trap which is activated if the list ITRACE is unexpectedly empty,in which case a message is output via a call to routine KERROR and control is returned.

Entry Conditions The variables ITRACE and ISKIP must contain valid list addresses (APPENDIX I),and list ITRACE must have a particular structure (APPENDIX V).

Exit Conditions The first argument STATUS may have one of three possible values,the value of SUCCES,FAILUR or ERROR;if the first value then the argument NUMJMP should have been overwritten with a positive integer value.

Subroutines Used

MINISLIP: LISTMT,NOKTOP,NEWTOP,ID,INHALT,LNKL

GP: EMPTY,KERROR,SWPSTK

Calling Routine TRUNO6

27. Identification MP27 SUCCO6(STATUS,NUMJMP,NONS,NSTART, NSTRIN,LNKVAL)

Description The routine IMATCH is called to deal with certain list structures after a specific character has been matched - in routine PROCO6.The variables NONS and NONT are initialised for the next matching process,the complete initialisation of variables and list structures for matching the next input feature is accomplished by means of a call to routine ICOUNT and then routine STRAT.A check is made that the number of input statement characters to be output without alteration,that is,the value of NUMJMP,does not exhaust the input string of characters.If the value passed to this routine in the argument NUMJMP is too large then STATUS is assigned the value of variable FAILUR and control is returned.

Else a marker is inserted in list ITRACE to indicate that the two following items delimit a portion of the input string

within the buffer STRING,that is to be output unchanged. Futher items are added to list ITRACE to result in the necessary specific structure (APPENDIX V).

Finally the variables and list structures are initialised via calls to routines ICOUNT and STRAT for the attempted matching of the next input feature and control is returned.

Entry Conditions The integer value of argument NUMJMP when added to the position within buffer STRING,of the last character matched,NONSTR,should not exceed the limit of the input string to be processed,that is,NSTRIN.The variable ITRACE must contain a valid list address (APPENDIX I).

Exit Conditions If the value of STATUS is not equal to that of ERROR or FAILURE then all the necessary variables and lists will have been initialised for the next matching process.

Subroutines Used IMATCH,ICOUNT,STRAT,STEP

MINISLIP: NEWTOP,LNKR,INHALT

GP: KERROR

Calling Routine TRUN06

28. Identification MP28 FAIL06(STATUS,LNKVAL)

Description This is a dummy routine added for the sake of clarity,it consists simply of a call to routine FAIL01 and control is returned.

Entry Conditions None.

Exit Conditions None.

Subroutines Used FAIL01

Calling Routine TRUN06

29. Identification MP29 IMATCH(STATUS)

Description This routine is entered immediately after a specific character has been matched.The variable CONRAT is assigned the positive value "3".Any specific character node addresses stored in list ISKIP are transferred to list ITRACE.The current tree node address,NONU,is placed in ITRACE followed by an integer which is the number of preceding items in list ITRACE that are associated with the last input feature matched (see APPENDIX V for detailed structure of list ITRACE).Finally the position within the buffer STRING of the last input character that was actually matched,NONSTR, is placed in list ITRACE.

Then via a call to routine STOSET,a copy of the current matching process (list structures and variable values) is placed in list ICHKON,to be utilised later possibly,within routine CHECKS.

Next all the items within list INONUS,which are tags to corresponding node addresses in list NONUST,are reset to the value "-9999" to indicate that a specific character has been matched and thus the current tree node address variable,NONU, cannot be reset as the address of an alternative node to any of the corresponding nodes in list NONUST (see APPENDIX V for usage of list INONUS).

Entry Conditions The variables ISKIP,ITRACE,NONUST and INONUS must contain valid list addresses (APPENDIX I).The Common Block variable IRIGHT must contain an integer which is equal to the number of items in list ISKIP.The number of items in list NONUST must be equal to the number in list INONUS.If any of these conditions are not met an error message is output.

Exit Conditions The lists ISKIP,ITRACE,NONUST and INONUS are restructured prior to the selection of a new strategy.

The value of variable CONRAT is overwritten with the real value "3.0",and variable IRIGHT with "0".If an error trap has been activated the variable STATUS will contain the value of variable ERROR;else it will remain the same as on entry.

Subroutines Used STOSET

MINISLIP: LISTMT,NOKBOT,NEWTOP,ID,INHALT,NOKTOP

GP: KERROR,SWPSTK,EMPTY

Calling Routines SUCC01,SUCC06

30. Identification MP30 STOSET(STATUS)

Description Each time a specific character is matched, this routine is entered and used to store a copy of the complete enviroment - values of variables and list structures - in list ICHKON.These sets of enviroments are stored in case it is decided later during the matching process that some previous,apparently successful match was not correct,then the system can be restored to this previous position with one of these sets of enviroments,which is reset in routine CHECKS.

Entry Conditions The variables ICHKON,NONUST,IBRNCH,ISKIP, INONUS,NID6ST and ID6STK must all be valid list names (APPENDIX I).

Exit Conditions On exit the lists all remain unchanged, except list ICHKON which contains a copy of all the other lists,plus an integer from either the variable INUMST or NUMSET which is used to identify each set (see list structure in APPENDIX V,and routine BRANCH for use of identifying number).

Subroutines Used

MINISLIP: NEWTOP,NOKBOT

GP: KERROR,SWPSTK,STOINV,STOREN

Calling Routines IMATCH

31. Identification MP31 BRANCH(STATUS)

Description This routine resets the matching process with a complete new enviroment - that is,variable values and list structures - that have been stored in the list IBRNCH,if any.These enviroments are stored during the execution of routine PROCOl when matching process tentatively assumes that a particular tree node data character is correct and the node has an alternative.

If the subsequent attempt at an actual match proves unsuccessful then routine BRANCH is invoked to reset the matching process to take into account the first of the previously overlooked alternatives,if any.

If the list IBRNCH is empty then STATUS is assigned the value of variable FAILUR and control is returned.

If this list is not empty then the top enviroment is reset in the variables NONU,IRIGHT and in the lists INONUS,NONUST, NID6ST and ID6STK.Variable NONU is reset as the address of the first alternative node to the one that was 'skipped' during the previous matching process.STATUS is assigned the value of variable PROCED and control is returned.

There is also an error trap which is activated if the new value of IRIGHT is greater than or equal to the value on entry.

Entry Conditions The variables ISKIP,IBRNCH,NONUST, INONUS,NID6ST and ID6STK must contain valid list addresses (APPENDIX I),and list IBRNCH must contain a specific structure (APPENDIX V).

Exit Conditions The argument STATUS may have one of three values;that of variable FAILUR,ERROR or PROCED.If the latter then the structures of the lists ISKIP,IBRNCH,NONUST,INONUS, NID6ST and ID6STK will have been altered and the values of variables NONU and IRIGHT overwritten.

Subroutines Used

MINISLIP: LISTMT,NOKTOP,LNKL,INHALT

GP: KERROR,EMPTY,SWPSTK

Calling Routines FAIL01,FAIL07

32. Identification MP32 STEP(STATUS,NSTRIN,NONS)

Description This routine moves the marker NONS from its current value,marking an input string character within buffer STRING,onto the next location,unless;either,it is already marking the end of the current string of characters (i.e. NONS=NSTRIN),in which case a flag is set (STREND=YES),STATUS is assigned the value of SUCCES and control is returned;or, the contents of the next location represent a character that we wish to ignore,then the marker is moved to this pos tion and the whole process of the routine is repeated.

Entry Conditions The value of the argument NONS must be less than or equal to the value of argument NSTRIN.

Exit Conditions The values of STATUS and NONS may be altered but that of NSTRIN will remain the same as on entry.

Subroutines Used

MINISLIP: ID,INHALT

GP: MACHIN

Calling Routines SUCC01,SUCC06

33. Identification MP33 ICOUNT(STATUS,NONS)

Description This routine is entered after each input string character is successfully matched against a tree node data character.The routine is essentially divided into two sections,one that is executed after each match of an input

feature,and the other only after a total successful match of the complete input statement.

The former section is executed if the value of STATUS,on entry,is not equal to that of variable ENDED.Within this section the terminal node addresses,which contain the frequency count associated with the tree item they terminate, that relate to the previously matched input feature,followed by the initial enviroment of the feature just matched,are transferred from list ID6STK to list NREPET.The initial enviroment for the next character to be matched is stored in list NREPET and control is returned.

If,on entry,the value of the argument STATUS is equal to that of ENDED,then a complete input statement has been successfully matched.List ID6STK and list NREPET are emptied and the terminal nodes found therein - that is,addresses of nodes whose ID field has the value "2" - are passed to routine ADDONE for the incrementation of the count field that each contains.When incrementing the terminal nodes that are associated with the various matching strategies (see,the strategy tree,2.6),if the address of the initial node of the strategy is not that of the topmost strategy, contained in variable NORMAL,then variable IFLAG is set to "999",to indicate that this statement has probably been modified.

Entry Conditions The variables ID6STK,NREPET,NONUST, NONENV,INONUS and NID6ST must contain valid list addresses (APPENDIX I).Furthermore the lists ID6STK and NREPET have somewhat complex structures which must be as described elsewhere (APPENDIX V) or an error trap will be activated. There are two slight variations upon the structure of list NREPET,dependent upon whether or not the last matching

strategy involved backtracking or not,and distinguished by testing the relevant field within the terminal node associated with this still current strategy (the terminal node is the successor of the initial node whose address is in variable NTEST).

The variable NONS must contain an integer which is the position within the buffer STRING of the beginning of the next feature to be matched.

Exit Conditions All the list structures will have been radically altered (see APPENDIX V for details).

Subroutines Used ADDONE

MINISLIP: LISTMT,NOKTOP,NEWTOP,ID,INHALT,LNKR

GP: SWPSTK,EMPTY,UNSTAK,KERROR

Calling Routines TRUNDL,SUCCO1

34. Identification MP34 ADDONE(STATUS,NCELL)

Description The ID field of the node whose address is received as the contents of the argument NCELL,or a node directly right of it,must have the value of "2".If such an ID value is not found within the sequence of nodes directly right of the node whose address is in NCELL,then an error trap is activated,a message is output via a call to routine KERROR and control is returned.

But if a node with an ID field value of "2" is found,then the 'count' field of the data word of this node is inspected. (APPENDIX III for details of tree node structure).If the value of this 'count' is less than the maximum (determined by the field size) then it is incremented by one and control is returned;else the contents of the count field are reset to "1" and the message "COUNT LOCATION 'node address' FULL", is output.Control is returned.

Entry Conditions The second argument,NCELL,must be a tree node address which has an ID field value of "2",or else such a node address must be obtainable via link-right addresses.

Exit Conditions The value of the first argument,STATUS, will be equal to the value of variable ERROR if an error condition has occurred;else it will be the same as on entry. The value of the first parameter will be overwritten with the address of the node which had an ID field value of "2", and which therefore may be the same as on entry.

Subroutines Used

MINISLIP: ID,INHALT,LNKR,LNKL,SETIND

GP: KERROR

Calling Routine ICOUNT

35. Identification MP35 TRNAME(NAME,NADD)

Description This routine searches for the name of a tree within the Fortran list,NTREES,which is equal to the contents of the first argument NAME.If such a name is not found then an error message is output,"BRANCH NAME 'contents of variable NAME' NOT ON TREE LIST,STOPPING",and the program is terminated.

If the equivalent name is found,then the contents of the corresponding location within Fortran list,ITREES,are assigned to the second argument,NADD;thus NADD should contain the root address of the tree whose name w s passed to this routine within the first argument.

Entry Conditions The Fortran lists NTREES and ITREES (within the Common Block /TREES/) shouln contain all the tree names currently in use by the system and their root addresses,respectively - these lidts are initialised by

the routine from section TB; SETUP.

Exit Conditions If the program is not terminated for the above reason,then the root address of the tree whose name is in variable NAME,is returned in the second argument NADD.

Subroutines Used None.

Calling Routines PROC07,SETEST

TB: NTGROW

36. Identification MP36 SETLIS

Description This routine initialises the lists (see APPENDIX I) used,via Common Block /LISTES/,during the matching process.The initialisation involves removing a cell from the List of Available Space (APPENDIX I) and creating with it a 'list header cell',for each list required,plus structuring the Fortran variables to be used as list names such that in future they will be recognised as valid list names.

Entry Conditions None.

Exit Conditions On exit the variable names used as the arguments in calls to routine LIST will be structured as valid list names.

Subroutines Used

MINISLIP: LIST

Calling Routine

I/O: TAPEIN

37. Identification MP37 RLISTS

Description This routine is entered when the complete matching process is terminated.All the lists initialised in routine SETLIS are emptied and their 'header cells' are

returned to the List of Available Space (APPENDIX I).Control is returned.

Entry Conditions All the variables within Common Block /LISTES/ whose lists are to be emptied and header cells returned must contain valid list names.

Exit Conditions The Fortran variables used as list names will no longer be valid list names.

Subroutines Used

GP: EMPTY

MINISLIP: RCELL

Calling Routines

I/O: TAPOUT

38. Idehtification MP38 TRUNO8 (STATUS,NSTRIN,NONS)

Description Directs control to one of three routines dependent upon the value of argument STATUS.

Entry Conditions None

Exit Conditions The values of the first and last argument may have been overwritten and list L16 may have been assigned three items.

Subroutines Referenced PROCO8,FAILO8,KERROR

Referencing Routine TRUNID

39. Identification MP39 PROCO8 (STATUS,NSTRIN,NONS)

Description If the value of variable NTHEAD is less than that of NBHEAD then the 'confidence jump' is not activated, but STATUS is assigned the value of variable FAILUR and control is returned.

Else,the tree node character at the end of the 'jump' is unpacked as variable ICHAR,and the number of times that this character must be found in the input string before the 'jump' is terminated,is unpacked from the IDDATA field of

the current tree node and placed in variable NUMTIM.The input string is then scanned for NUMTIM occurrences of character ICHAR,if the end of the input string is encountered first then STATUS is assigned the value of FAILUR and control is returned;else the current values of variables NONSTR,NONU and NONS are stored in list L16,NONSTR is reset as the final occurrence of the character ICHAR,NONU is reset as the address of the tree node at the end of the jump,and control is returned.

Entry Conditions Arguments NSTRIN and NONS should mark the positions within buffer STRING of the final and current input statement characters respectively.

Exit Conditions The value of argument STATUS or the COMMON block variables NONSTR and NONU will have been overwritten.

Subroutines Referenced

MINISLIP:ID,INHALT,LNKL,NEWTOP

Referencing Routine TRUNO8

40. Identification MP40 FAILO8 (STATUS)

Description This routine transfers control to the successor of the jump node when the jump has failed or is not activated.

Entry Conditions COMMON block variable NONU should contain the address of a 'jump' node (ID=8,see APP. III).

Exit Conditions The variables STATUS and NONU are overwritten.

Subroutines REferenced

MINISLIP: LNKR,INHALT

Referencing Routine TRUNO8

41. Identification MP41 TRUNO3 (STATUS)

Description Routine directs control to one of three routines dependent upon the value of argument STATUS.

Entry Conditions None

Exit Conditions The value of STATUS will have been overwritten and the COMMON block variable NONU or items added to the output buffer NTRACE.

Subroutines Referenced PROC03,SUCC03,KERROR

Refrencing Routine TRUNID

42. Identification MP42 PROC03 (STATUS)

Description The value of the IDDATA field of the current tree node is added to the output buffer NTRACE, in the position indicated by variable IPLACE.IPLACE is incremented by one,STATUS is assigned the value of variable SUCCES and control is returned.

Entry Conditions None.

Exit Conditions Argument STATUS and COMMON block variable IPLACE will have been overwritten as described above.

Subroutines Referenced

MINISLIP: ID,INHALT

Referencing Routine TRUN03

43. Identification MP43 SUCC03(STATUS)

Description This routine transfers control to the successor of the current tree node and resets the value of STATUS to that of PROCED,or that of ENDED if the current tree node has no successor.

Entry Conditions None.

Exit Conditions The values of argument STATUS and COMMON block variable NONU will have been overwritten.

Subroutines Referenced

MINISLIP: LNKR,INHALT

Referencing Routine TRUN03

44. Identification MP44 TRUNO4 (STATUS)

Description Routine directs control to one of three routines dependent upon the value of argument STATUS.

Entry Conditions None.

Exit Conditions The value of STATUS will have been overwrittan and items added to the output buffer NTRACE.

Subroutines Referenced PROCO4,SUCCO4,KERROR

Referencing Routine TRUNID

45. Identification MP45 PROCO4 (STATUS)

Description This routine unpacks a parameter number from the IDDATA field of the current tree node and places the value in variable IPARAM.The list ITRACE is then scanned for the occurrence of this parameter number,if it is not found then an error message is output via a call to routine KERROR and control is returned.If the parameter is found then items are placed in the output buffer as dictated by this parameter,control is then returned.

Entry Conditions List ITRACE must be structured correctly (see APP. V).

Exit Conditions Argument STATUS will have been overwritten and if the error trap has not been activated items will have been added to the output buffer NTRACE.

Subroutines Referenced

MINISLIP:ID,INHALT,NOKBOT,NEWTOP

GP: KERROR

Referencing Routine TRUNO3

46. Identification MP46 SUCCO4 (STATUS)

Description Simply a call to SUCCO3 and return.

Entry Conditions None.

Exit Conditions see SUCCO3

Subroutines Referenced SUCCO3

Referencing Routine TRUNO4

APPENDIX II - Description of the routines

The Restructuring Process Routines:section RP

Index

1. Identification RP1 the main routine

Description This routine is used to control the restructuring process,its precise structure will depend upon exactly what restructuring is required at any particular time.Thus we will content ourselves with simply indicating certain sequences of control that must be observed.

Prior to restructuring a system of trees containing 'jump nodes' or 'patterns',the routines JMPOUT and PATOUT (both in this section) must be used to remove the jump nodes and patterns.Similarly the routines JUMPIN and PATERN (again both in this section) are used after restructuring,if we require to replace these features removed.

The 'branch swopping' process (see 2.4) requires the utilization of routine SETJNC,to place the relative frequencies within each node as dictated by the current frequency counts; then routine RESRTC actually does the 'swopping' from an examination of these relative frequencies.

The 'promotion mechanism' (see 2.4) is controlled by routine PROMOT,after the above branch swopping mechanism has been activated.

Entry Conditions None.

Exit Conditions None.

Subroutines Referenced JMPOUT,JUMPIN,PATOUT,PATERN, NODOUT,PROMOT,NUMTRE,NODECT,SETJNC,RESTRC,NUMODD
MINISLIP: LIST,RCELL,ID,INHALT
GP: EMPTY,
I/O: TPRINT,TAPEIN,TAPOUT

Referencing Routines None.

2. Identification RP2 PATERN (STATUS,NLIST,NUMPAT, NUMNOD,NROOT)

Description This routine 'patterns' the items of the tree whose root address is contained within the fifth argument,NROOT;only items in which the terminal node has the LNKR ADDRESS field of less than or equal to zero (that is items that currently do not have a pattern) are patterned.

The number of patterns added and the number of nodes added are returned within the arguments,NUMPAT and NUMNOD, respectively.

The pattern currently added is that for a syntax checking system.

Entry Conditions The argument NROOT must contain a valid tree node address.The argument NLIST must contain a valid list address (APPENDIX I).

Exit Conditions The values of the arguments NUMPAT and NUMNOD will have been overwritten as indicated above.

Subroutines Referenced

MINISLIP:ID,INHALT,NEWTOP,NOKBOT,NUCELL,SETIND,LNKL,LNKR,LISTMT GP: KERROR

Referencing Routine the main routine RP1

3. Identification RP3 PATOUT (STATUS,NLIST,LNKLST, NUMPAT,NROOT)

Description This routine deletes all the patterns from all the terminal nodes governed by the node address within argument NROOT.If the value of the LNKR ADDRESS field (see APPENDIX III,i) is less than or equal to zero it is assumed that no pattern is present and the next terminal node is sought.The number of patterns deleted is returned as the

value of argument,NUMPAT.

There is one error trap that is activated if a non-terminal node has no 'link-right' address,an error message is output via a call to routine KERROR,variable STATUS will have been assigned the value "ERROR"(within routine KERROR) and control is returned.

Entry Conditions The argument NROOT must contain a tree node address and the arguments NLIST and LNKLST must contain valid list addresses (APPENDIX I).

Exit Conditions The argument NUMPAT will have been overwritten as described above,the argument STATUS will contain the characters "ERROR" if the error trap has been activated.

Subroutines Referenced RTREE
MINISLIP:LNKL,INHALT,NEWTOP,LNKR,SETIND,LISTMT,NOKTOP
GP: KERROR,EMPTY

Referencing Routine the main routine RP1

4. Identification RP4 JUMPIN (NROOT,MINFRQ,JMPLST, LNKLST,JMPTOT,NODTOT,STATUS,NODLST)

Description This routine inserts 'jump nodes' (ID=8, see APPENDIX III,i for details) prior to every sequence of two or more 'redundant' specific character nodes (i.e. ID =1),where a node is considered to be'redundant' if the relative frequency of its first alternative (i.e. the node addressed by the contents of the current node's LNKL ADDRESS field) is less than the value of argument MINFRQ; the important special case is when the current node has no alternative and is hence structurally redundant within the language definition.

The number of jump nodes inserted and the number of nodes that they jump (which can be simply the number of nodes

comprising each redundant sequence or,in addition any defined alternatives and the nodes they might govern,that these 'redundant' nodes might have:the value of variable LEVFRQ determines which),are returned as the values of the arguments JMPTOT and NODTOT,respectively.

Entry Conditions The arguments JMPLST,LNKLST and NODLST must contain valid list names (APPENDIX I).The argument NROOT must contain a tree node address.

Exit Conditions The values of arguments JMPTOT and NODTOT will have been overwritten as described above.The value of argument STATUS may have been overwritten as "ERROR".

Subroutines Referenced KNODES

MINISLIP:LISTMT,LNKL,INHALT,LNKR,NUCELL,SETIND,ID,NOKTOP,NOKBOT

GP: EMPTY,KERROR

Referencing Routine the main routine RP1

5. Identification RP5 JMPOUT (STATUS,NROOT,MAXFRQ, NUMNOD,LNKLST)

Description This routine deletes all jump nodes (ID=8, see APPENDIX III,i) from the items with relative frequencies less than or equal to the value of argument,MAXFRQ,in the tree governed by the node address in argument NROOT.

The number of nodes deleted is returned as the value of argument NUMNOD.

There is the usual error trap for non-terminal nodes which do not have a successor.

Entry Conditions The argument LNKLST must contain a valid list address (APPENDIX I).Argument NROOT must contain a tree node address.

Exit Conditions The value of the argument NUMNOD is overwritten as described above or if the error trap has been activated STATUS will contain the value "ERROR".

Subroutines Referenced

MINISLIP:ID,INHALT,LNKR,RCELL,SETIND,LNKL,LISTMT,NOKTOP

GP: EMPTY,KERROR

Referencing Routines the main routine RP1

6. Identification RP6 RTREE (NODADD,LNKLST)

Description This routine returns all the nodes governed by the node whose address is in NODADD,to the List of Available Space (APPENDIX I).

Entry Conditions The first argument must contain a tree node address and the second,LNKLST must contain a valid list address (APPENDIX I).

Exit Conditions The argument NODADD is overwritten as zero or a negative value.

Subroutines Referenced

MINISLIP: LNKL,INHALT,LNKR,RCELL,NEWTOP,LISTMT,NOKTOP

GP: EMPTY

Referencing Routines PATOUT,INSTRC,NODOUT

7. Identification RP7 KNODES (NROOT,LNKLST,KOUNT,MINFRQ)

Description This routine searches the tree governed by the node address in argument NROOT,whilst the relative frequency is greater than or equal to the value of argument MINFRQ.The number of nodes encountered is returned as the value of argument KOUNT.

There is the usual error trap for a non-terminal node that has no successor.

Entry Conditions The argument NROOT must contain a tree node address and argument LNKLST must contain a valid list address (APPENDIX I).

Exit Conditions The argument KOUNT is overwritten as described above.If the error trap has been activated STATUS

will contain the characters "ERROR".

Subroutines Referenced

MINISLIP:ID,INHALT,LNKL,LNKR,NEWTOP,NOKTOP,LISTMT

GP: EMPTY,KERROR

Referencing Routines JUMPIN

8. Identification RP8 NUMTRE (NROOT,LNKLST,NTOTAL)

Description This routine is used in conjunction with routine NODECT (RP9,following) to check that the tree frequency counts are self-consistent.

This routine searches the tree governed by the node address in argument NROOT,accumulates the sum of all the frequency counts and returns the total in argument NTOTAL.

If a terminal node does not have ID=2 the message," NO LINK-RIGHT &node address' ",is output and control is returned.

Entry Conditions The first argument NROOT must contain a tree node address and the second LNKLST a valid list address.

Exit Conditions The argument NTOTAL is overwritten as described above,unless it is the error exit.

Subroutines Referenced

MINISLIP:LNKL,INHALT,NEWTOP,NOKTOP,LNKR,LISTMT

GP: EMPTY

Referencing Routines the main routine RP1 ,SETJNC,PROMOT, INSERT I/O: NUMODD

9. Identification RP9 NODECT (NROOT,ITYPE,IDDATA,NDUMMY ,LNKLST,NODLEV,KOUNTS,KUMSUM)

Description This routine searches the tree governed by the node address within the argument,NROOT,for any occurrence of the node specified by the contents of arguments ITYPE and IDDATA.Each time an occurrence is located,if any,the number of times that it has been successfully matched

in the past (i.e. its associated frequency count) is added to a cumulative total within the argument,KUMSUM.

The type of node sought is specified by the ID field value (in argument ITYPE) and IDDATA field value (in argument IDDATA field,these fields are illustrated in APPENDIX III,i fig.I).

The error message," NO LNKR 'node address' ",is output and control is returned if a non-terminal node is found that has no successor.

Entry Conditions The arguments NDUMMY,LNKLST,NODLEV and KOUNTS must all contain valid list addresses(APPENDIX I), and the argument NROOT a tree node address.

Exit Conditions The argument KUMSUM is overwritten as described above,unless the error trap is activated.

Subroutines Referenced

MINISLIP:ID,INHALT,LNKL,LNKR,NEWTOP,NOKTOP,LISTMT

GP: EMPTY,SWPSTK

Referencing Routines the main routine RP1

10. Identification RP10 SETJNC (STATUS,NROOT,NODEST,KSUMST)

Description This routine first obtains the cumulative total of all the frequency counts associated with the tree governed by the node address in argument NROOT;this is done by means of a call to routine NUMTRE (RP 8 , above),the total being returned within the argument NTOTAL.

This routine now traverses the current tree itself and by examination of the frequency counts places the appropriate relative frequency (currently as a fraction of 10,000) in the LINK-RIGHT DATA field (see APPENDIX III,i) of every node in a filial set of greater than one.

There are several error traps,two for unexpectedly empty

lists in which an error message is output via a call to routine KERROR (GP2) and control is returned;the third trap is activated when a terminal node is encountered that does not have an ID value of "2",in which case an error message is output," TERMINAL CELL HAS ID= 'ID value found'", and control is returned.

Entry ConditionsThe arguments NODEST and KSUMST must contain valid list addresses (APPENDIX I),and argument NROOT a tree node address.

Exit Conditions The tree nodes are altered as described or,if either of the first two error traps has been activated the returned value of STATUS will be "ERROR".

Subroutines Referenced NUMTRE

MINISLIP:ID,INHALT,LNKL,LNKR,LISTMT,NEWTOP,NOKTOP,SETIND

GP: EMPTY,KERROR

Referencing Routine the main routine RP1

11. Identification RP11 RESTRC (STATUS,NROOT,NODEST,LNKLST)

Description This routine re-structures the tree governed by the node address in argument NROOT.The re-structuring is simply an ordering of the nodes in each filial set in descending order of the vaues of their LINK-RIGHT DATA fields (i.e. their relative frequencies, see routine SETJNC RP10,above).

There is an error trap which is activated if list,LNKLST is unexpectedly found to be empty,a message is output via routine KERROR (GP2) and control is returned.

Entry Conditions The arguments NODEST and LNKLST must contain valid list names (APPENDIX I),and NROOT a tree node address.

Exit Conditions The tree nodes in each filial set of the tree will have been reordered as described above (i.e. 'branch swopping',see 2.4,will have taken place).

If an error condition has been activated then argument,STATUS will contain the characters "ERROR".

Subroutines Referenced ASORT

MINISLIP:ID,INHALT,LNKL,LNKR,LISTMT,NEWTOP,NOKTOP

GP: EMPTY,KERROR

Referencing Routines the main routine RP1

12. Identification RP12 ASORT (STATUS,ILIST)

Description This routine reorders the nodes whose addresses are the contents of list ILIST,in ascending order of the values of their LINK-RIGHT DATA fields (see APPENDIX III,i for node structure).

Entry Conditions The argument ILIST must be a valid list name (APPENDIX I) and further,the list must contain at least one item else an error condition is activated,a message output via a call to routine KERROR (GP2) and control is returned.

Exit Conditions The contents of list ILIST will have been reordered as described above,unless the error condition was activated in which case STATUS will contain "ERROR".

Subroutines Referenced

MINISLIP:LISTMT,NEWTOP,NOKBOT,NOKTOP,LNKR,LIST,RCELL

GP: EMPTY,KERROR

Referencing Routines RESTRC

13. Identification RP13 PROMOT (MINENT,PROPIM,PROPER, LNKLST,NODEST,NDUMMY)

Description This routine sets the parameters for the'promotion mechanism' (see 2.4) and passes control to routine DISCOV (RP15,below).

Each tree is examined to determine how many substructures, if any,are the maximum to possibly be promoted from it.

If the number of times that items have been matched on the current tree is less than the value of argument MINENT then control is transferred to consider the next tree,or returned if no further trees.

If not,then by means of a call to routine TOTITM (RP14, following),the number of substructures on level one of the current tree is obtained as the value of variable NOITEM. The maximum number of substructures that can be promoted from the current tree is calculated as ITMPRM.Next the minimum frequency that this number of substructures must contain MINFRQ,is calculated,and the then the minimum average per substructure,NAVFRQ.

An information message is output," IN TREE 'tree name' NOITEM 'value of NOITEM' ITMPRM 'values of ITMPRM,MINFRQ and NAVFRQ' ".If the number of items to be promoted is less than one control is transferred to consider the next tree;else the tree structure is output via a call to routine TPRINT (I/O2),control is then transferred to routine DISCOV (RP15,below).On return from DISCOV the tree structure is again output via a call to TPRINT and control is transferred to consider the next tree,or return if no further trees to consider.

Entry Conditions Arguments LNKLST,NODEST and NDUMMY must be valid list names (APPENDIX I).

Exit Conditions None.

Subroutines Referenced TOTITM,DISCOV,NUMTRE

I/O: TPRINT

Referencing Routines the main routine RP1

14. Identification RP14 TOTITM (NROOT,NITEMS)

Description This routine returns the number of sub-structures on level one of the tree governed by the node

whose address is in argument NROOT;which can also be described as the number of lower precedence nodes in the filial set of the node whose address is in NROOT,plus one for node NROOT itself.

Entry Conditions Argument NROOT must contain a tree node address.

Exit Conditions The argument NITEMS is overwritten as described above.

Subroutines Referenced

MINISLIP: LNKL,INHALT

Referencing Routines PROMOT

15. Identification RP15 DISCOV (NTROOT,MINFRQ,NAVFRQ, ITMPRM,LNKLST,NODEST,NAME,NDUMMY)

Description This routine examines the tree governed by the node whose address is passed in argument,NTROOT,and determines if a substructure (composed of the sequence of level one nodes and their associated substructures,if any) is to be promoted from the current tree using the criteria passed as arguments.The criteria for possible promotion amount to,if at most the substructure composed of the highest precedence ITMPRM,number of level one nodes (and associated structures,if any),contain a total frequency count of greater than or equal the value of argument MINFRQ;level one nodes are no longer considered if their individual frequency counts are not less than the value of argument NAVFRQ.

If these criteria are satisfied then control is passed to routine INSERT (RP16,below) which determines at which positions the current tree is referenced amongst all the trees and continues with the promotion mechanism.

But,if not,an information message is output," NO PROMOTION FROM TREE 'tree name' ,ROOT ADDRESS &value of NTROOT' ",

then control is returned.

Entry Conditions Arguments LNKLST,NODEST and NDUMMY must contain valid list names (APPENDIX I),and argument NTROOT a valid tree node address,argument NAME should contain the corresponding tree name.

Exit Conditions None.

Subroutines Referenced INSERT

MINISLIP: LNKR,INHALT,LNKL

GP: EMPTY

Referencing Routines PROMOT

16. Identification RP16 INSERT (NROOT,ITYPE,IDDATA, LNKLST,NODEST,NEND,NDUMMY)

Description This routine examines all the trees in the system consecutively,in order to determine if the substructure rooted at NROOT and ending at the node whose address is in argument NEND (which may be equal to NROOT),is to be inserted in each tree and if so it inserts it,possibly in a modified form.

The criteria for insertion are;first that the tree that the substructure originated in,specified by the values of arguments ITYPE and IDDATA,is referenced from the tree currently under consideration for insertion.Second that the reference node has an associated frequency count of greater than zero,i.e. the referenced tree has actually been successfully matched from this particular referencing node.

Then from the number of referencing nodes within the tree currently under consideration and from the number of nodes that comprise the substructure to be promoted a decision is made,by a comparison with the value of variable MINUSE, whether to proceed with thw promotions or abandon because too many nodes will be required.

If the substructure itself is too large the information message," NO PROMOTION FROM TREE,AS TOO MANY NODES IN SUBSTRUCTURE,NUMBER IS 'value of NUMNOD' ",is output and control is returned.

Else the information message," SUBSTRUCTURE TO BE PROMOTED TO 'number of places' PLACES IN TREE 'name of tree' " is output.

Then if the total number of nodes that would be required to place a copy of the substructure at each of these positions is too large (i.e. greater than MINUSE) the information message," NO PROMOTION INTO THIS TREE AS TOO MANY NODES NEEDED ",and control is transferred to consider the next tree.

Else control is transferred to insert the substructure,or a modification of it.The information message," SUBSTRUCTURE TO BE INSERTED IS ",is output followed by the actual substructure.

Next control is passed to routine COPY (RP17,below) which returns a copy of the substructure which is to be inserted in the first reference position with slightly higher precedence than the referencing node.But before the actual insertion,routine INSTRC (RP18,following) is invoked to determine if any,or all of the substructure already occurs at the proposed position of insertion,if so then these parts are deleted from the copy to be inserted.On return from INSTRC,if the complete substructure has been deleted then control is transferred to examine the next position of possible insertion,else the message," AFTER DELETIONS, SUBSTRUCTURE TO BE INSERTED IS ",is output followed by the remaining substructure.

This substructure is then inserted,there are several error traps via routine KERROR (GP2),except the one which detects

null tree items (i.e. terminal node on level one) when the message "ERROR,NULL BRANCH ADDRESS 'node address' ",is output and control is transferred to consider the next tree.

Entry Conditions Arguments LNKLST,NODEST and NDUMMY must contain valid list addresses (APPENDIX I),argument NROOT a tree node address and NEND the address of the same node or a lower precedence member of the same filial set.

Exit Conditions None.

Subroutines Referenced COPY,INSTRC,NUMTRE
MINISLIP:LNKL,INHALT,LISTMT,NOKTOP,NEWTOP,LNKR,ID,SETIND,RCELL
GP: KERROR,EMPTY
I/O: TPRINT

Referencing Routines DISCOV

17. Identification RP17 COPY (NROOT,LNKLST)

Description This routine copies the tree structure rooted at the node address in the argument NROOT and returns the address of the root node of the copy in argument NROOT.

Entry Conditions Argument LNKLST must contain a valid list address (APPENDIX I),and argument NROOT a tree node address.

Exit Conditions The value of the argument NROOT will be overwritten as described above.

Subroutines Referenced
MINISLIP:NUCELL,ID,INHALT,LNKL,LNKR,SETIND,LISTMT,NEWTOP,NOKTOP
GP: EMPTY,KERROR

Referencing Routines INSERT

18. Identification RP18 INSTRC (ICOPY,NEND,NTOP,ICURAD, KUMTTM,LNKLST)

Description This routine examines all,if any,of the substructures in higher precedence positions than that of

the intended insertion.If any of these substructures occur within the substructure to be inserted than they are deleted from the substructure to be inserted.

Entry Conditions Argument LNKLST must contain a valid list name (APPENDIX I),and arguments ICOPY,NEND,NTOP and ICURAD must contain tree node addresses;ICOPY and NEND refer to the substructure to be inserted as described above,whilst ICURAD is the address of the referencing node within the current tree and NTOP is the highest precedence member of the filial set containing ICURAD (which may be ICURAD,in which case they will be equal).

Exit Conditions The value of argument ICOPY may be overwritten and although NEND will remain unchange it may no longer refer to a node in the substructure,that is,the lowest precedence item in the substructure may be deleted.

Subroutines Referenced RTREE

MINISLIP:ID,INHALT,LNKL,LNKR,SETIND

GP: EMPTY

I/O: TPRINT

Referencing Routine INSERT

19. Identification RP19 NODOUT (IROOT,NLIST,LNKLST)

Description This routine examines the tree whose name is passed as the value of argument IROOT and deletes any items that are redundant and have not been utilised sufficiently.More specifically it deletes items which are referenced by higher precedence nodes in the same filial set;and also items that are referenced by lower precedence nodes of the same filial set,if the redundant item has no record of past successful usage (i.e. frequency count zero).

If an item is deleted an information message is output,

" LOWER PRECEDENCE SUBSTRUCTURE 'tree structure deleted'

DELETED FROM TREE 'name of tree' ",or " HIGHER PRECEDENCE SUBSTRUCTURE 'tree structure deleted' DELETED FROM TREE 'name of tree' ".

Just prior to the return of control the information message," 'number of deletions' DELETIONS FROM TREE 'name of tree' ",is output.

Entry Conditions Arguments NLIST and LNKLST must contain valid list names (APPENDIX I) and argument IROOT a valid tree name (see routine SETUP I/01).

Exit Conditions None.

Subroutines Referenced RTREE,INTREE

MINISLIP:ID,INHALT,LNKL,LNKR,NEWTOP,NOKTOP,LISTMT,SETIND

GP: EMPTY

I/O: TPRINT

Referencing Routines the main routine RP1

20. Identification RP20 INTREE (NROOT,ITYPE,IDDATA,IANSWR)

Description This routine searches for the type of node specified by the arguments ITYPE (value of ID field) and IDDATA (value of IDDATA field,see APPENDIX III,i) in the tree whose root address is the value of argument NROOT,If the type of node is found in level one of the tree then argument IANSWR is returned with the value "99",else it is returned with the value "0".

Entry Conditions The argument NROOT must contain a tree node address.

Exit Conditions Argument IANSWR is overwritten.

Subroutines Referenced

MINISLIP: ID,INAHLT,LNKL

Referencing Routines NODOUT

21. Identification RP21 ZEROTR (NROOT,LNKLST,KOUNT,JUNCTN)

Description This routine assigns the value of zero to the frequency counts in the terminal nodes and the relative frequency counts within each node in every filialset of grrater than one;the nodes so processed are those governed by the node whose address is passed to the routine within the first argument,NROOT.

The frequency count fields are zereed if the value of the third argument,KOUNT is not "-1",and the relative counts are zeroed if the value of argument JUNCTN is not "-1".

Entry Conditions The argument LNKLST must contain a valid list name(APPENDIX I),and argument NROOT must contain a tree node address.

Exit Conditions None.

Subroutines Referenced

MINISLIP:ID,INHALT,LNKR,LNKL,LISTMT,SETIND,NEWTOP,NOKTOP

GP: EMPTY

Referencing Routines the main routine RP1

APPENDIX III

The data structures;

i) details of tree node structures,

ii) details input source structure required by routine SETUP I/01 and the subsequent Tree Building Process, APPENDIX II, section TB, also required by ADD I/07 and DELETE I/08

iii) details of input format required by routine CHANGE I/09

i) details of tree node structures

The particular node structures that we employ have six fields as illustrated generally in Fig. I of APPENDIX II. A node is simply a cell (as described in APPENDIX I) that in association with other cells forms the tree structures utilised by the Matching Process.

The most important field is the ID field which can have one of a small number of possible values. The value of the ID field is then used to interpret the remaining structure of the particular node.

Fig. I - the generalised node structure

ID field	LINK-LEFT ADDRESS field	LINK-RIGHT ADDRESS field
ID DATA field	LINK-LEFT DATA field	LINK-RIGHT DATA field

ID=1; then LINK-LEFT ADDRESS field (LNKLAD) contains, the address of an alternative node or zero;

LINK-RIGHT ADDRESS field (LNKRAD) contains, the address of a successor node;

ID DATA field (IDDATA) contains a single specific character to be interpreted as an immediate symbol;

LINK-LEFT DATA field (LKLDAT) contains zero.

LINK-RIGHT DATA field (LKRDAT) contains the relative frequency,wihtin its particular tree,of past successful matches of the data character contained in the node agains input characters,at the time when the routine which sets this field SETJNC (section RP) was last entered - stored as a fraction of 10,000.

ID=2;indicates a terminal node,and:

LNKLAD contains the address of an alternative node or zero.

LNKRAD contains zero in System I,but will contain the address of a successor node in System II if the tree item terminated by this node has an associated pattern (for description of pattern see section 4.2).

IDDATA contains zero.

LKLDAT contains a count of the number of times that this node has been successfully processed during the analysis of incoming statements (that is contains the frequency count associated with this item,section 4.4).

LKRDAT as for ID 1 node above with the single exception of when the ID 2 node is directly right of an ID 5 node and is thus part of a general strategy within the strategy tree (section 2.6),in which case this field contains the value of the variable NBACK.

ID=3;this node can only be part of the pattern associated with a tree item and thus can only occur within System II.

LNKLAD contains zero.

LNKRAD contains the address of a successor node or zero.

IDDATA contains a single specific character which is to be interpreted as an immediate symbol.

LKLDAT contains zero.

LKRDAT contains zero.

ID=4; this node is the second (the first is ID 3 described above) of the two types of node that comprise the pattern associated with a tree item (see section 4.2 for description of pattern). As with the ID 3 node, the ID 4 node only occurs within a pattern and is thus retricted to main trees.

LNKLAD contains zero.

LNKRAD contains the address of a successor node or zero.

IDDATA contains a positive integer which is used to reference and output a specific string of characters that has been constructed during the processing of an input statement.

LKLDAT contains zero.

LKRDAT contains zero.

ID=5; this node is used exclusively in the strategy tree (section 4.22) as the initial node of a general strategy.

LNKLAD contains the address of an alternative strategy (and hence ID 5 node), or zero.

LNKRAD contains the address of a terminal node with a special strature (described above under ID 2 nodes).

IDDATA contains a positive integer which specifies the type of strategy by dictating the setting of several variables within routine SETKSG (section MP); the value is extracted as the variable ITEST within routine SETEST (section MP, APPENDIX II).

LKLDAT contains the value of variable LEVLNO which is extracted within routine SETEST (section MP, APPENDIX II)

LKRDAT contains the value of variable LNKVAL which is again extracted within routine SETEST (section MP, APPENDIX II).

ID=6; then LNKLAD contains the address of an alternative node or zero.

LNKRAD contains the address of a successor node.

IDDATA contains a single character which is to be interpreted as an immediate symbol.

LKLDAT contains a positive integer.

LKRDAT as for ID 1 node.

ID=7; this type of node is used to reference a subtree and is thus a subroutine 'call'.

LNKLAD contains the address of an alternative node or zero.

LNKRAD contains the address of a successor node.

IDDATA contains a single character which is to be interpreted as the name of a subtree.

LKLDAT contains zero in System I, but can contain a positive integer if the item(s) of which this node is part, have an associated pattern; that is within System II only. The number is the parameter number and is used in conjunction with the IDDATA value of an ID 4 node to output a string of processed characters in a desired position.

LKRDAT as for ID 1 node.

ID=8; This ID value designates a confidence jump node and thus is restricted to main trees (for description of confidence jump see section 4.21).

LNKLAD contains the address of the node to which control is transferred if the jump mechanism is activated.

LNKRAD contains the address successor node, that control is transferred to if the jump mechanism is not activated or fails.

IDDATA contains an integer which controls how many times the jump terminating input string characters must be 'jumped' so that the new current input string character should correspond to the new tree node, after the action of the jump mechanism.

LKLDAT contains zero.

LKRDAT contains zero.

ID=10; this node is used to output the remainder of the input string unprocessed and hence unaltered.

LNKLAD contains zero.

LNKRAD contains the address of a terminal node.

IDDATA contains zero.

LKLDAT contains zero.

LKRDAT contains zero.

The value of the ID field of a cell is the tree node 'strategy code' of chapter 4.

The detailed action of each of these strategy codes is defined by the three routines associated with each (occasionally only two routines), SUCCnn, FAILnn and PROCnn (all in section MP, APPENDIX II), where 'nn' is the particular strategy code such as '02', etc.

Thus the three routines which specify the action of terminal nodes which have a strategy code 2 or 02, are; PROC02, FAIL02 and SUCC02. The value of variable STATUS (APPENDIX IV, no.80) determines which of the three routines specifies the action at any particular time (see flowchart preceding section MP, APPENDIX II).

ii) details of the input source structure required by routine SETUP (section I/O).

The information used in the Tree Building Process is input on the 80 column Hollerith cards.

First all the tree names to be used are input as single characters in the first columns of successive cards,thus process is terminated by the one obligatory tree name which is "E".

Next tree items are input in one of the two possible formats,which are;

1) the condensed format,in which we can specify up to 22 nodes per card but can only specify the ID and IDDATA fields (the LNKLAD and LNKRAD fields are specified automatically by the context of the node).

The tree name in which the subsequent item is to be constructed is given in column 1;and then from column 7 to column 72 inclusive,that is 66 columns,divided into 22 sets of three,each set of which can specify a node.The first two columns of each set is expected to contain the ID field value as an integer and the third column a character for the IDDATA field.If the ID field is found to be blank then the preceding field is considered to be the terminal node of the item and control is transferred to the next card for the next tree item. If the two columns of the ID field contain "99" then the specification of the current tree item is expected to continue on the following data card in exactly the same manner as above, confirmation of the continuation card must be provided by the characters "KONTIN" in the first six columns of any such cards.(See example deck setup below).

2) the expanded format,in which we can only specify one node per card but can now explicitly specify the ID,IDDATA, LKLDAT and LKRDAT fields for each node.

The series of cards which specifies each tree item is now sequenced with "1" on the initial card of a new item and "99" on the card specifying the terminal node, the intermediate cards must contain an ascending series of sequence numbers between "1" amd "99". The first card for each new tree item must contain the name name of the tree within the subsequent item is to be constructed, in column 1.

The general structure of a card in expanded format is;

card column numbers

1 2 3 4 5 6 7	8 9	10 11 12	13 14 15 16 17 18 19	20 21 22 23 24 25	26 27 28 29 30 31
	seq. num.	ID	IDDATA	LKLDAT	LKRDAT

After reading in the tree names control is switched to expect tree item data in the condensed format but can be switched to expanded format by the occurrence of a blank card; another blank card will switch control back again, and so on.

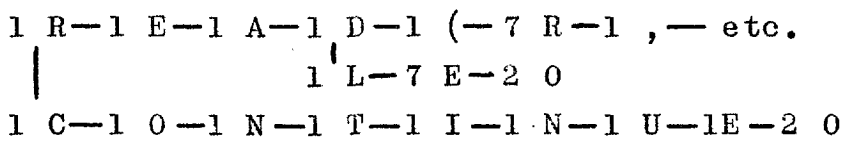
The input and subsequent construction of tree items is terminated by the occurrence of the characters "00000E" in the first six columns of a data card.

Fig. II - an example deck setup for the initialisation and construction of two trees.

1 2 3 4 5 6	7 8	9 10 11	12 13 14	15 16 17	18 19	20 21 22 23	24 25	26 27	28 29 30 31
FREE									
FREE	1	R 1	E 1	A 1	D 1	(7	R 1	,	etc.
RR	1	1 9	2 0						
RF	1	5 2	0						
FF	1	R 1	E 1	A 1	L 7	E 2	0		
KONTIN	1	C 1	0 1	N 1	T 9 9				
	1	I 1	N 1	U 1	E 2	0			
			blank card						
R	1	1	6				0		0
	5	1	0				0		0
	9 9	2	0				0		0
000000E									

the resultant trees would be:-

tree F



tree R

tree E

Empty in this example but in a practical situation this would cause system errors.

Note: only ID and IDDATA values of each node illustrated as all LKLDAT and LKRDAT fields are zero.

iii) details of the input format required by routine CHANGE I/09

Card column 1 is left blank, column 2 contains a single character which will be interpreted as a tree name, then columns 3 and 4, 5and 6, etc up to 71 and 72, must contain the successive values of the ID and IDDATA fields of the item whose count location is to be altered.

The last eight card columns (that is columns 73 to 80) must contain an integer value which will be inserted in the count location associated with the item specified.

APPENDIX IV - important program variables

	name	occurrence	usage
1.	A(7500)	RP1 transferred as argument	Fortran list which is initialised as MINISLIP cells - also known as ICELLS(7500),see below
2.	AVSL	MCOPS,TAPEIN, TAPOUT connected via EXTERNAL statement	special location used in MINISLIP package to monitor LAS(APPENDIX I)
3.	CHEX	COMMON block /LISTES/	a list name variable (APPENDIX I) used in routine CHECKS(MP8)
4.	CHKLST	COMMON block /LISTES/	as above
5.	CONRAT	COMMON block /MAINBK/	to indicate if current character matched or current strategy to be abandoned
6.	CRPLET	COMMON block /MAINBK/	value controls if non-storage processing or not
7.	DATAR	argument in RP1,MP1,TAPEIN TAPOUT	contains information necessary to restart matching process at position of previous termination
8.	DATE	as above	
9.	ENDED	COMMON block /MODES/	value assigned to variable STATUS when matching completed
10.	ERROR	COMMON block /MODES/	value assigned to variable STATUS by routine KERROR (GP2)
11.	FAILUR	COMMON block /MODES/	value assigned to STATUS when action dictated by current node strategy code cannot be completed
12.	IBRNCH	COMMON block /LISTES/	a list name variable (APPENDIX I)

13. ICELLS(7500) MP1,TAPEIN, TAPOUT — other name for A(7500) above

14. ICHKON COMMON block /LISTES/ — a list name variable (APPENDIX I), list structures and usage APP. V

15. ID6STK COMMON block /LISTES/ — a list name variable

16. ID7TAG COMMON block /TAGID7/ — if current tree node has ID=7 then this variable contains value of associated tag,see list INONUS APP. V

17. IFLAG COMMON block /AFLAG/ — value tested in routine PROCES(MP2) to determine if the statement just processed has been modified,value assigned in routine ICOUNT (MP33)

18. IMARK MP1,TAPEIN, TAPOUT — same as DATAR above no.7

19. IPLACE COMMON block /BUFOUT/ — indicates current position in output buffer NTRACE(200) below

20. IRIGHT COMMON block /MAINBK/ — current input character compared with current tree character when value equal to that of KRIGHT,below

21. INONUS COMMON block /LISTES/ — a list name variable,see APP. I,for structure and usage see APP. V

22. INUMST COMMON block /CHKON/ — contains a number to identify the current set of matching characteristics if set has been used previously;else zero.Assigned in routine CHECKS(MP8) and tested in STOSET (MP30)

23. ISKIP COMMON block /LISTES/ — a list name variable,see APP. I,for structure and usage see APP. V

24. ISTRIN MP1,TAPEIN, TAPOUT — same as DATAR above no.7

25. ITOTIM	COMMON block /TIMES/	THe total time for processing the current batch of statements is accumulated within routine PROCES (MP2) and output by STATS (I/06)
26. ITREES(64)	COMMON block /TREES/	contains the tree root addresses, assigned in routine SETUP (I/01)
27. KRIGHT	COMMON block /MAINBK/	assigned in routine SETKSG(MP6) contains number of specific character nodes (i.e. ID=1) to be assumed correct before a match is attempted
28. KSUCES	COMMON block /TIMES/	contains the number of statements successfully processed of the current batch,set in PROCES (MP2)
29. KTAG	COMMON block /SETTAG/	indicates in routine STRAT(MP4) that current matching conditions have been used unsuccessfully before, assigned in routine CHECKS(MP8)
30. KTREES	COMMON block /TREES/	contains the number of trees currently being used,set in SETUP(I/01)
31. KTRUN	COMMON block /TIMES/	contains current number of statements processed,set in PROCES(MP2)
32. KUMSUC	COMMON block /TIMES/	contains cumulative total of successfully processed statements for all batches since last zeroed
33. KUMTIM	COMMON block /TIMES/	as above but times for all batches processed
34. KUMTRN	COMMON block /TIMES/	as above but total number of statements processed
35. L16	COMMON block /LISTES/	a list variable name,see APP. I, structure and usage see APP. V
36. L17	COMMON block /LISTES/	not used

37. LIMIT COMMON block /ISTATS/ — contains the number of statements to be processed as the current batch, input as NCARDS in routine MP1 and transferred to LIMIT in routine TAPEIN (I/05).

38. LINE(80) MP1, TAPEIN, TAPOUT argument — first input buffer for one card image

39. LINEP(80) MP1, TAPEIN, TAPOUT argument — second input buffer for one card image

40. LINK COMMON block /LISTES/ — a list variable name, see APP. I, structure and usage, see APP. V

41. LNKVAL argument — assign from current matching strategy in routine SETEST, passed on to other routines and determines the lower limit of relative frequency which the search will proceed to.

42. MARK MP1, TAPEIN, TAPOUT argument — as DATAR above no.7

43. MAXCEL COMMON block /SELLS/ — COntains the maximum number of available cells, assigned in routine SPACE (GP8)

44. MINCEL COMMON block /SELLS/ — contains the minimum number of cells that were available since initialisation, assigned in SPACE(GP8)

45. MTEST COMMON block /TESTKT/ — contains the address of the matching strategy that gave rise to an apparently match but will be discarded if the current backtracking strategy is successful; if not strategy address restored in routine STRAT(MP4),

		initially assigned in routine SETREE (MP7)
46. MTRACE	COMMON block /LISTES/	not used
47. NBHEAD	COMMON block /MAINBK/	contains value which is compared with values of variables CRPLET to determine if non-storage processing and NTHEAD to determine if confidence jumps to be activated or not.Value assigned in routine TRUNDL (MP3)
48. NCARDS	COMMON block /ISTATS/	receives number of statements to be processed in next batch,input in routine MP1,then is assigned number of cards processed in all previous batches,in routine TAPEIN(I/O (I/O5)
49. NDUMMY	COMMON block /LISTES/	a list variable name,see APP. I, structure and usage see APP. V
50. NID6ST	COMMON block /LISTES/	as above
51. NONBLK	COMMON block /TIMES2/	contains the total number of characters processed,excluding blanks
52. NONENV	COMMON block /LISTES/	a list name variable,see APP. I, structure and usage see APP. V
53. NONS	argument in many routines of matching process MP APP. II	contains the number which refers to the position within input buffer STRING of the current character being processed
54. NONSTR	COMMON block /MAINBK/	as above but position of character to be actually matched to complete matching of current input feature

			assigned in routine SETKSG (MP6)
55.	NONT	COMMON block /MAINBK/	contains address of the tree node from which the matching of the current input started,assigned in routine SUCCO1 (MP12)
56.	NONU	COMMON block /MAINBK/	contains address of current tree node during the matching process
57.	NONUST	COMMON block /MAINBK/	a list variable name,see APP. I, structure and usage see APP. V
58.	NORMAL	COMMON block /AFLAG/	contains address of the highest precedence general strategy (see 2.6 for introduction to the strategy tree),assigned in routine PROCES (MP2) and tested in routine ICOUNT (MP33) to signal a modification of input statement.
59.	NREORD	COMMON block /RORDE/	tranfers number of nodes reordered in each filial set during the branch swopping process,from routine ASORT (RP12) to main routine RP1
60.	NREPET	COMMON block /LISTES/	a list name variable,APP. I, structure and usage see APP. V
61.	NSIZE	main routines, TAPEIN,TAPOUT argument	contains dimension of Fortran list A no. 1 above,or ICELLS no.13
62.	NSTART	MP1 and many routines of matching process	indicates position in buffer STRING of first character in character string currently being processed
63.	NSTRIN	MP1 and many routines of matching process	as above but indicates last position
64.	NTEST	COMMON block /TESTKT/	contains address of current general strategy,assigned in SETEST (MP5)

65. NTHEAD COMMON block /MAINBK/ — if value not less than value of variable NBHEAD, then confidence jump mechanism activated, assigned in routine TRUNDL(MP3) and tested in routine PROC08(MP38)

66. NTIMUP COMMON block /TIMES2/ — contains time limit set in routine PROCES(MP2) for processing one statement in 1/60 ths second, tested in TRUNDL (MP3)

67. NTMTRP COMMON block /REJECS/ — number of statements time trapped

68. NTRACE(200) COMMON block /BUFOUT/ — output buffer constructed in routines PROC03(MP42) and PROC04(MP45)

69. NTREES(64) COMMON block /TREES/ — contains all the valid tree names, set in routine SETUP (I/01)

70. NUCARD COMMON block /CARD/ — set to one in PROCES(MP2) and used in CHECKS(MP8) to signal first entry with a new statement

71. NUMCEL COMMON block /SELLS/ — contains current number of cells available for use, set in SPACE(GP8)

72. NUMCHG COMMON block /REJECS/ — contains the number of statements modified

73. NUMCHR COMMON block /TIMES2/ — contains the number of characters processed including blanks

74. NUMJMP PROC06, SUCC06 argument — set in PROC06(MP26) as number of successive characters to be output unaltered, used in SUCC06(MP27)

75. NUMPOS COMMON block /RORDE/ — the number of nodes in the currentt filial set, set in ASORT(RP12) used in main routine RP1

76. NUMSET COMMON block /CHKON/ — contains identification number stored in list CHKLST with last set of initial matching conditions, CHECKS(MP8

77. NXS200 COMMON block /REJECS/ — the number of statements containing more than 199 characters

78. PROCED COMMON block /MODES/ — set in BLOCKDATA(GP1) and assigned to variable STATUS when control transferred to a new tree node

79. START argument equivalent to NSTART, used in routine PROCES(MP2) — see NSTART no. 62 above

80. STATUS argument in matching process — assigned values from variables in COMMON block /MODES/ and directs control via routines TRUN01, TRUN02, etc

81. STREND COMMON block /SENDED/ — initialised to zero in TRUNDL(MP3) assigned value of variable YES when current input string exhausted

82. STRING(200) COMMON block /MAINBK/ — input buffer

83. SUCCES COMMON block /MODES/ — value set in BLOCKDATA(GP1) and assigned to variable STATUS when the action dictated by the strategy code of the current tree node is terminated successfully

84. TOTCRT COMMON block /MAINBK/ — assigned value by routine ERROR(MINISLIP) when error condition detected within MINISLIP routines (see APP.I)

85. XS200 — same as for DATAR no.7 above

86. YES COMMON block /MODES/ — set in routine BLOCKDATA(GP1) and assigned to variables STATUS and STREND

APPENDIX V

List structures and usage.

i) The list structures and usage during 'storage' processing of an incoming statement (see 5.222 for description of 'storage' and 'non-storage' processing').

ii) The list structures and usage during 'non-storage' processing of an incoming statement.

list structure notation:

"=" indicates the bottom of a list

"." is used to separate list items, or a repeated series of items within parentheses

"( )$^{n=0}_{N}$" indicates that the item(s) inside the parentheses are repeated n times where n can take a value from 0 to N

list items may be described using lower case letters, or the program variables used may be named using upper case characters, or both; specific values may also be used, such as "-3"

i) Storage processing

1. On return from routine STRAT (section MP,APPENDIX II), the list structures are initialised for the matching of the next input string feature against the experience tree.

list NONUST - contains in order (that is,address of last subtree reference is on top),the return addresses of the reference nodes to all the subtrees currently entered to obtain the address of the node from which the next matching will start,stored in variable NONT.

$=$(a subtree reference node address.)$_{N}^{n=0}$

list INONUS - contents have a one to one correspondence with those of list NONUST,each item has the value "-3333" (set by routine PRO607,section MP),or "-9999" (set by routine SUCC01, section MP);if the latter value then a specific character has been matched within the subtree referenced by the corresponding node in list NONUST and the current tree node variable,NONU,cannot be reset as an alternative to this node, if any.

$=$("-3333" or "-9999".)^{n} where "n" is the number of items in list NONUST,above.

list ID6STK - contains a copy of the initilised variables and lists NONUST and INONUS,terminated by a marker "-1".This list is constructed within routine SETREE (section MP).

$=$NONS.NONT.contents of list NONUST'contents of list INONUS'the number of items copied from list NONUST'

address of current strategy node."-1".

list NREPET - contains a copy of the complete matching process so far, which consists of, a copy of all initial conditions (as in ID6STK above) plus the addresses of 'count' nodes (that is nodes with ID=2) encountered during matching and subtree reference nodes discarded from list NONUST with corresponding 'tags' (that is, items from list INONUS which are processed within variable ID7TAG) from list NID6ST.

=(address of strategy node·contents of list INONUS in reverse order·contents of list NONUST in reverse order·number of items from list NONUST·the relevant value of NONT·the relevant value of NONS·contents of list NID6ST in reverse order·number of items from list NID6ST·"-3".((addresses of count nodes and subtree reference nodes from list ID6STK·)$^{i=0}_{I}$ (relevant value of variable IRIGHT ·"-2".)$^{j=0}_{J}$)$^{k=0}_{K}$ · relevant value of variable NONSTR ·)$^{l=0}_{L}$ where l+1 is the number of previous features matched with current statement being processed, in addition, there are items associated with the last feature matched, which are;

address of ·strategy node·contents of list INONUS in reverse order·contents of list NONUST in reverse order·number of items from list NONUST·previous value of variable NONT ·previous value of variable NONS·(contents of list NID6ST in reverse order·number of items from list NID6ST·"-3".(contents of parentheses i as above)(contents of parentheses j as above))$^{m=0}_{M}$ ·previous value of variable NONSTR ·

where "m" is usually zero, but might be greater than zero if the current strategy involves bactracking, and which if it proves successful will cause the contents of parentheses m to be deleted (in routine ICOUNT, section MP) and replaced by new items.

list NONENV - contains a copy of all the items removed from list NREPET whilst the current matching process was initilised (items removed and stored in list NONENV in routine SETREE, section MP) as dictated by the current strategy,which was set by an earlier call to routine SETEST (section MP).Usually this is simply a copy of the current contents of lists NONUST and INONUS plus the current values of variables NONS and NONT.But,if the current strategy demands backtracking, then the list NONENV will contain copies of all the matching processes that were apparently successful subsequent to the input string character to be matched,STRING(NONSTR),set by this backtracking strategy.

These copies of the matchings of input features from list NREPET,have embedded within them between markers,"-99",the corresponding items from list ITRACE.

If the current backtracking strategy proves to be successful, then the contents of list NONENV are discarded (within routine ICOUNT,section MP);if unsuccessful,the contents of list NONENV are restored to lists NREPET and ITRACE,on entry to routine STRAT (section MP) prior to assigning the next strategy (or signaling failure) and initialising the next matching of an input feature.

=(a set of items from list NREPET associated with the matching of one input feature,in reverse order . "-99" .

relevant value of NONMAT,the position in buffer STRING of character matched . relevant value of NONS, position of the first

character in feature matched . node addresses from list ITRACE . number of node addresses .

"-99".)$^{p=0}_{p}$ where p is greater than zero only if currently backtracking;plus a copy of the current matching process,

.NONS.NONT. number of items from list NONUST . contents of list NONUST . contents of list INONUS .

list NID6ST - is used to contain the tags (that is,values "-3333" or "-9999") that are removed from list INONUS each time a subtree reference node is tentatively discarded from list NONUST and placed amongst the count nodes in list ID6STK during the matching of a feature from within an input statement being processed.This list is emptied prior to the matching of the next feature,by routine SETREE(APPENDIX II,MP7).

list ITRACE - contains the positions within the input buffer, STRING,of the characters actually matched during the analysis of a statement;and the appropriate the parameter values are als stored within this list.The parameters are stored as addresses of nodes that contain immediate symbols or,as the beginning and end markers to a string of characters within the input buffer,STRING.

=.("-98".parameter number.n node addresses."n+2".position within buffer STRING of actual character matched,

NONSTR.(m node addresses."m".relevant value of NONSTR.)$^{i=0}_{I}$.("-90".initial marker to buffer STRIN(

.final marker."3".NONSTR.)$^{j=0}_{J}$)$^{k=0}_{K}$ where k,the number of parameters is determined by the main tree item that is directing the matching.

When a complete statement has been successfully analysed the list is restructured within routine TRUNDL (APPENDIX II,MP3) prior to the stored pattern directing the construction of the output buffer (see section 4.31),NTRACE.The restructured list is;

=.("-98".parameter number.node addresses.("-90".initial marker.final marker.)$^{p=0}_{P}$)$^{k=0}_{K}$

list IBRANCH - is emptied prior to each matching of an input statement feature (in routine TRUNDL,section MP).It is used and structured as described in the following section (APPENDIX V,i, 2.).

list CHKLST - is used in conjunction with list CHEX to store copies of the initial conditions of each of the matching processes attempted with the current input statement so far, Each of these stored sets has an associated number which is unique and is used to connect each set with a corresponding enviroment stored in list ICHKON (see below).Thus once the proposed next initial conditions for matching have been selected (by routines SETREE,SETEST and SETKSG,all section MP), all the previous initial conditions are examined (within routine CHECKS,section MP) to determine if they have been used previously and if so,altered or a new set selected. During this checking process sets are transferred to list CHEX for temporary storage.

$$=\Big(\begin{array}{l}\text{relevant strategy code variable ITEST}\\ \text{relevant value of variable LEVLNO}\\ \text{relevant value of variable NONS}\\ \text{relevant value of variable NONT}\\ \text{set identification number variable INUMST or NUMSET}\end{array}\Big)_{R}^{r=0}$$

where list CHEX contains s sets of the same variables but in reverse order and r+s is the number of different attempts to match input statement features that have been tried with the current input statement,including the current attempt.

list CHEX - see list CHKLST above.

list ICHKON - contains sets of the complete enviroments (list structures and variable values) that,in conjunction with the four variable values stored in list CHKLST (see above),completely specify the final state of each of the matching processes which terminated in successfullly matching an input statement feature.Each of these sets has an associated identification number which is equal to the identification number of the corresponding set in list CHKLST (described above).The list ICHKON also contains null sets which have a similar identification number which is equal to an identification number in list CHKLST but corresponds to a matching process which terminated in failure and therefore should not be retried.Each set in list ICHKON has an associated tag to indicate whether the corresponding matching process was terminated with a successful match (tag "9000" and set stored by routine STOSET,section MP),or with a failure (tag "9999" and identification number stored by routine SETEST,section MP).

=(relevant identification number variable INUMST or NUMSET . "9000" . number of items following in this set

number of following items from list ISKIP . contents of list ISKIP in reverse order . number of following

items from list IBRNCH . relevant contents of list IBRNCH in reverse order . number of items following from list NID6ST

relevant contents of list NID6ST in reverse order . number of following items from list INONUS . contents of list INONUS

in reverse order . number of following items from list ID6STK . contents of list ID6STK in reverse order .

number of following items from list NONUST . it contents of list NONUST in reverse order . relevant value of variable NONU .

or relevant identification number variable INUMST . "9999" .)$_{T}^{t=0}$ where t=r+s-1,see list CHKLST above

list ISKIP - is used to temporarily store the addresses of specific character nodes that are tentatively assumed to be correct during the matching of an input feature.If the assumption proves to be incorrect the contents of ISKIP may be reduced by routine BRANCH (APPENDIX II,MP31) or the matching might fail completely and list ISKIP is emptied in routine TRUNDL (APPENDIX II,MP3);else the assumptions are confirmed by a specific maych and the contents of list ISKIP are transferred to list ITRACE (above) within routine IMATCH (MP29).Thus list ISKIP is empty at the beginning of the matching of an input statement feature.

i)

2. During the attempted matching of the next input statement feature,which is delimited within the buffer STRING by the values of the variables NONS and NONSTR,starting from the tree node whose address is contain in variable NONT,several list structures are altered as follows;

list NONUST - the contents of this list may be increased and/or decreased,even to zero,during any attempted matching. Subtree reference node addresses are added to this list each time a subtree is entered.Each time a terminal node is encountered within a subtree then the return address (that is to the correct position within the referencing tree) is the successor to the top node address in list NONUST ,which is removed and placed on top of list ID6STK in case needed during any later backtracking.

list INONUS - the contents of this list as described earlier (i,1. above) will maintain a one-to-one correspondence with the items within list NONUST,thus if an item is added to list NONUST then an item is added to list INONUS,similarly if an item is removed from list NONUST and placed in list ID6STK then the top item is removed from list INONUS and placed in list NID6ST.

list ID6STK - the structure as described above (i,1.) will remain the same but,in addition terminal nodes (or 'count' nodes) that are processed during the matching will be added as will any subtree referencing nodes tentatively discarded from list NONUST (see i,2. above).If the matching fails,then these count and subtree reference node addresses will be discarded;else they will be utilised as described later(i,3.).

The count node addresses are placed on top of list ID6STK by routine SUCC02 (section MP) and the subtree reference node addresses by routine SUCC07(section MP).If the matching process is 'skipping' specific character nodes,that is assuming them tentatively to be correct without checking,then the current value of variable IRIGHT and a marker "-2" is also added to list ID6STK by routine SUCC01 (section MP).

=NONS.NONT.contents of list NONUST initially.initial contents of list INONUS.number of items copied from list

NONUST.address of current strategy node."-1".((a count node address or a subtree reference node address)$^{v=0}_{V}$ (relevant value of variable IRIGHT . "-2" .)$^{w=0}_{1}$)$^{x=0}_{X}$

where x-1 is the number of specific nodes 'skipped' and if x=1 then w=0.

list CHKLST - remains unchanged (see i,1. above).

list CHEX - remains unchanged (see i,1. above).

list ICHKON - remains unchanged (see i,1. above).

list NDUMMY - is used locally within routines STOINV (section GP) and PROC06 (section MP),for temporary storage.

list NREPET - remains unchanged(see i,1. earlier).

list NONENV - remains unchanged(see i,1. earlier).

list NID6ST - initially this list is empty but tags (that is items with the value "-3333" or "-9999") will be added if any subtree reference node addresses are transferred from list NONUST to list ID6STK during the matching.Similarly the top tag will be removed if any subtree reference node addresses are removed from list ID6STK (in routine FAIL01 or FAIL02 or FAIL07,all section MP APPENDIX II).

list ITRACE - remains unchanged(see i,1.).

list ISKIP - initially this list is empty,as described earlier (i,1.),during the matching of input feature it is used to store the addresses of any specific character nodes currently considered to be tentatively correct if a subsequent match confirms the correctness of the contents of this list it is dealt with as described within the following section (i,3.).Node addresses may be added by routine SUCC01 (section MP) and removed by routines BRANCH,FAIL01,FAIL02 and /or FAIL07 (all section MP).

list IBRNCH - this list is emptied prior to the matching of an input statement feature (emptied by routine TRUNDL,section MP).During the matching any specific nodes that have valid (that is have frequencies not less than the current minimum, in variable LNKVAL) alternatives are stored in this list by routine PROC01,along with various current list structures and variable values that would be required if we should later wish to examine these alternatives.

=(contents of list ID6STK in reverse order • number of preceding items from list ID6STK • contents of list NONUST

in reverse order • number of preceding items from list NONUST • contents of list NID6ST in reverse order •

number of preceding items from list NID6ST • contents of list INONUS in reverse order • number of preceding

items from list INONUS • relevant value of variable NONU • relevant value of variable IRIGHT •)$_{U}^{u=0}$

where u is the current number of specific character nodes that have been assumed to be correct and have valid alternatives

<u>list L16</u> - this list is utilised by the confidence jump mechanism for the temporary storage of variable values that will be reset if the 'jump' fails and discarded if it succeeds.

=.NONSTR.NONU.NONS.

The values are set within PROC08 (APPENDIX II,MP39),and utilised or discarded within PROC01 (MP13).

i)

3. The attempted matching of a feature from the input statement against the tree structures can be terminated with a specific match of the final character of the feature and hence success;or,with a failure to match in which case control is returned to routine STRAT (section MP,APPENDIX II) and the lists are re-initialised with the next matching strategy or a complete failure is returned with the current input statement.

But,in the case of success when a specific input character STRING(NONSTR),is matched against the character contained within the node whose address is in variable NONU,the lists are re-structured prior to the selection of the next input feature and matching strategy,as follows;

list ICHKON - is used to store a copy of the complete list structures and variable values which specify the terminating condition of the matching process such that if necessary, the matching could be continued from this point to search for a further match.The set of successful termination conditions is placed in list ICHKON by routine STOSET (section MP),a tag is added (that is the value "9000") which indicates that this particular set is of successful termination conditions - the sets of unsuccessful termination conditions (empty sets) is the value "9999" and is added to list ICHKON by routine SETEST (section MP).Both types of set stored also have ah associated identification number which connects each set with a set of variable values in list CHKLST (see list CHKLST i,2. above).

The detailed structure of the set of items stored in list ICHKON is given above (list ICHKON i,2.).

list ITRACE - the contents of list ITRACE are as described and illustrated earlier (i,1. above).After a specific character and hence the input statement feature which it terminates has been matched,then one of the two sets of possible items is added to this list.

generally:-

a series of specific character node addresses from list ISKIP	current value of variable NONU	a count of the number

of preceding items just added,KRIGHT+1 | the position in buffer of character matched,NONSTR

added by routine IMATCH (section MP)

or possibly:-

first position in buffer of feature matched	final position in buffer of feature	"3"	.NONSTR.	."-90"

STRING

added by routine SUCC06 (section MP)

list ISKIP - may contain specific character node addresses which are transferred to list ITRACE,thus list ISKIP is emptied.

list INONUS - which may contain a series of tags (that is, the value "-3333" or "-9999",explained above i,1.) is restructured (in routine IMATCH,section MP) such that each tag now has the value "-9999" .

list NONUST - remains unchanged (see i, 2. and i,1. above).

list NREPET - contains the previous history of the matching of the current input statement as described earlier (i,1.). Routine ICOUNT (section MP) adds first the tags from list NID6ST (if any) and a count of how many are added to list NREPET.

Then the count nodes,subtree reference nodes and markers (if any) are removed from list ID6STK and added to list NREPET,within the items associa ed with the last feature matched (N.B. not the current feature just matched). Then the remaining items from list ID6STK (except the marker "-1") are added to list NREPET;these represent the initial conditions for the feature just matched.Finally the initial conditions for the next feature to be matched,that is,the contents of lists NONUST and INONUS plus the current values of variables NONT and NONS,are added to list NREPET (these are removed in routine SETREE,section MP).

= up to and including parenthesis
1 described earlier (i,1.),the same°
items associated with the last feature matched:-

address of strategy node | contents of list INONUS in reverse order | contents of list NONUST in reverse oder

number of items from list NONUST | relevant value of variable NONT | relevant value of variable NONS

current contents of list NID6ST in reverse order | number of items added from NID6ST | "-3" | (count node addresses

and subtree reference node addresses | and markers | from current contents of list ID6STK (see i,2.)).

relevant value of variable NONSTR

plus initial conditions of feature just matched,again transferred from list ID6STK:-

relevant contents of list INONUS in reverse order | relevant contents of list NONUST in reverse order | number of preceding

items from list NONUST | relevant value of NONT | relevant value of NONS | relevant value of NONSTR

plus items associated with next feature to be matched:-

current contents of list current contents of list number of
INONUS in reverse order 'NONUST in reverse order 'items from

list NONUST. current value of variable NONT ' current value of variable NONS '

ii) Non-storage processing

The only difference from storage processing described above (in i) is that the storage lists,NONENV,ICHKON,CHEX, CHKLST,NREPET are not utilised.Similarly,a copy of the initialised variables preceding the marker "-1" (as described earlier in i,1) is omitted.

Control is directed to exhibit this method of processing when the value of variable CRPLET is greater than or equal to that of variable NBHEAD (APPENDIX IV,variables number 6 and 47,respectively).

APPENDIX VI - input and output index

i) input, ii) error messages, iii) information messages

i) input routines

ADD I/07 — accepts card images of items to be added to trees, card format in APPENDIX III, ii

CHANGE I/09 — expects card images which specify a tree item and a numeric value which is assigned to the frequency count of the specified item, card format in APPENDIX III, iii

DELETE I/08 — accepts card image of an item which is then deleted from the specified tree, card format in APPENDIX III, ii

READ I/010 — accepts card images of statements to be processed

SETUP I/01 — accepts tree names and then item which are used to construct the trees, card format APPENDIX III, ii

TAPEIN I/05 — reads total tree structure and associated variable values from output medium as output by routine TAPOUT(I/04)

ii) output routines - in alphabetical order of message output

KERROR GP2 ******* SYSTEM ERROR ******* SIGNALED FROM 'routine name'

plus one of the following:

INVALID ID = value of invalid ID field

INVALID VALUE OF PARAMETER = invalid value encountered

INVALID STRING POSITION, NONS= invalid value found

INVALID VALUE OF STATUS value as a character string

INVALID VARIABLE HAS VALUE numeric value encountered

LIST address of list header (APP.I) EMPTY

NO LINK-RIGHT FROM CELL cell address

NO SET FOUND WITH NUMSET= numeric value encountered

PARAMETER NUMBER numeric value NOT IN LIST ITRACE

STACK list name character string EMPTY

TRNAME MP35 BRANCH NAME name in A6 format NOT ON TREE LIST,STOPPING

DELETE I/08 BRANCH NAME NOT ON TREE LIST,MOVE TO NEXT CARD

ADDONE MP34 COUNT LOCATION node address FULL
occurs when value of frequency count is 32,767 location reset to 1 and processing continues

INSERT RP16 ERROR,NULL BRANCH ADDRESS node address
occurs when a null item (consisting only of a terminal node) encountered

SETUP I/01 ERROR IN DATA SETUP,CONTINUATION CARD EXPECTED, NOT FOUND...CARD IS card image in input format
occurs when card format as specified in APPENDIX III,ii is not correct

DELETE I/08 ERROR IN DELETE DECK STRUCTURE CONTINUATION CARD EXPECTED card image on following line
occurs as above message

NTGROW TB1 INCORRECT DATA SETUP,CARD READ IS *** card image in input format
message occurs when data incorrectly sequenced,see APPENDIX III,ii

DELETE I/08 NODE WITH ID= value of ID field AND DATA CHARACTER value of IDDATA field NOT ON TREE name of tree
occurs when input information specifies an item to be deleted which is not on the tree specified

NUMTRE RP8 NO LINK-RIGHT node address

NODECT RP9 occurs when a non-terminal node (i.e. not ID=2) has no successor

SETJNC RP10	TERMINAL CELL HAS ID= value of ID field found
	a node with no successor and not ID=2 found
DELETE I/08	TREE tree name IS NOW EMPTY
	all items have been deleted from above named tree

iii) self explanatory information messages are output from:

PROMOT RP13	IN TREE etc.
DISCOV RP15	NO PROMOTION FROM TREE etc.
INSERT RP16	AFTER DELETIONS SUBSTRUCTURE TO BE INSERTED etc
	NO PROMOTION INTO THIS TREE etc.
	NO PROMOTION FROM TREE etc.
	SUBSTRUCTURE TO BE INSERTED IS etc.
	SUBSTRUCTURE TO BE PROMOTED TO etc.
INSTRC RP18	FOLLOWING ITEM DELETED DUE TO NON USAGE
NUMODD I/011	NO ITEMS MATCHED ON THIS TREE
	IN TREE ADDRESS etc.
	AVERAGE ODDITY PER SYMBOL MATCHED etc.
	PERCENTAGE SAVING IN ODDITY PER TREE etc.

APPENDIX VII - system exits

RP1,MP1	normal termination of execution
SETUP I/01	the program can be terminated by either of two exits which are activated by faulty data conditions:an error message is output
NTGROW TB1	the error exit is activated by a faulty data setup:an error message is output
TRNAME MP35	if the character contained in this first argument transferred to this routine does not match any location in FORTAN list NTREESS then a message is output and execution terminated
SETCEL TB4	if the value of the ID field of the node to be constructued is invalid then a message is output,via a call to routine KERROR (GP2), and execution is terminated

APPENDIX VIII

Key to figures.

General features of experience tree hierarchy.

Fig.I initial experience tree for the analysis of Data Set I

Fig.II final,optimised experience tree from the analysis of Data Set I

Fig.III initial experience tree for the analysis of Data Sets II to V,illustrating confidence levels

Fig.IV final,optimised experience tree from the analysis of Data Set II

Fig.V as fig. IV after analysis of Data Set III

Fig.VI as fig. IV after analysis of Data Set IV

Fig.VII as fig. IV after analysis of Data Set V

Key to experience tree figures following:

The column of figures on the left hand side of each page is the percentage occurrence of the tree item that terminates on the same level and to the right.

Node representations:

ID=1,an immediate symbol; the symbol illustrated

ID=2,a terminal node; the value of the frequency count from within the node is illustrated in the terminal position

ID=6,deals with Hollerith; an immediate symbol preceded by the symbol '

ID=7,a subtree reference; the subtree name preceded by the symbol £,with paramter number subscript if pattern illustrated

ID=8,a jump node; the jump path is drawn over the nodes jumped and the number of times that the jump terminating symbol is to be skipped in the input string

ID=10,remainder of statement to be output unprocessed; illustrated as an arrow head

ID=3,immediate symbol pattern node;illustrated as the immediate symbol

ID=4,the number of a parameter in the pattern; the number is illustrated within the pattern

General features of the tree hierarchy defining FORTRAN

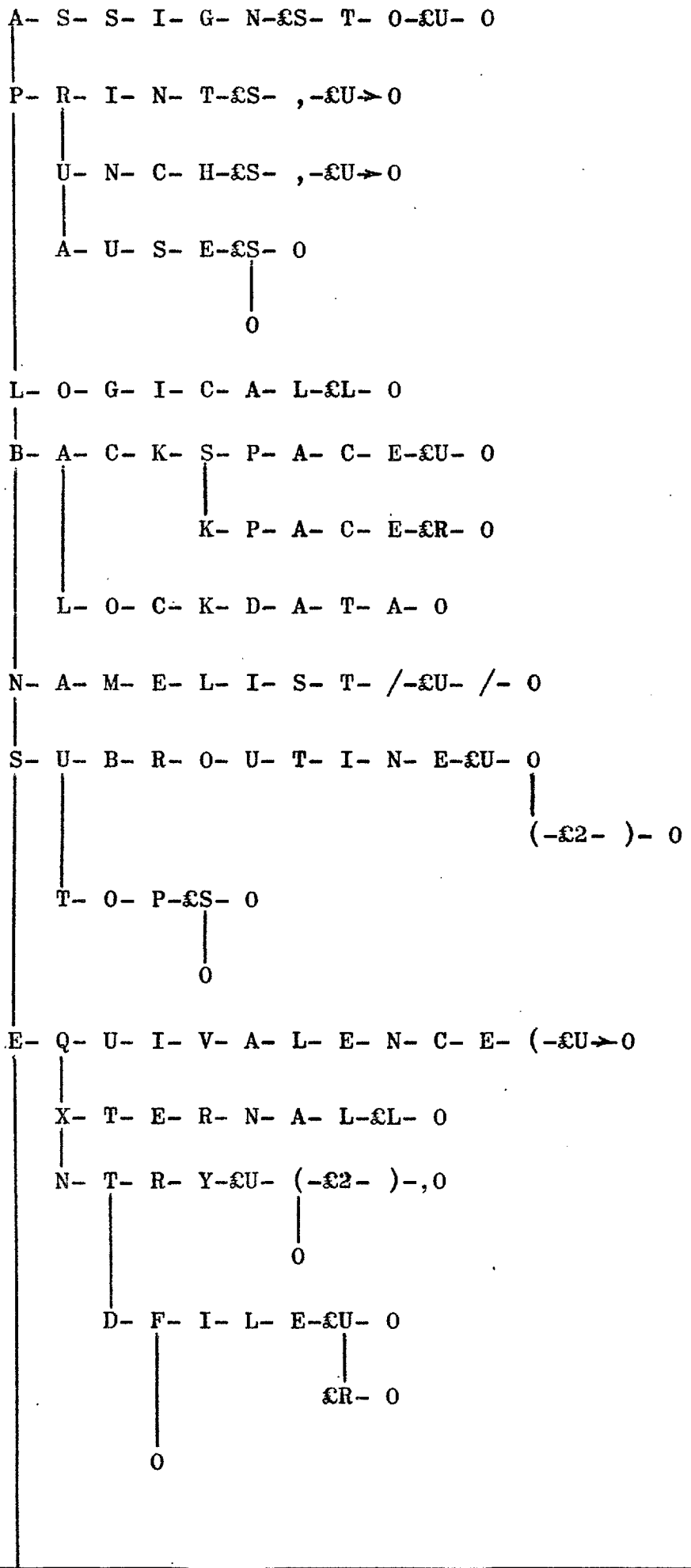
The total tree hierarchy is composed of approximately 1,700 nodes,including 600 for the patterns.

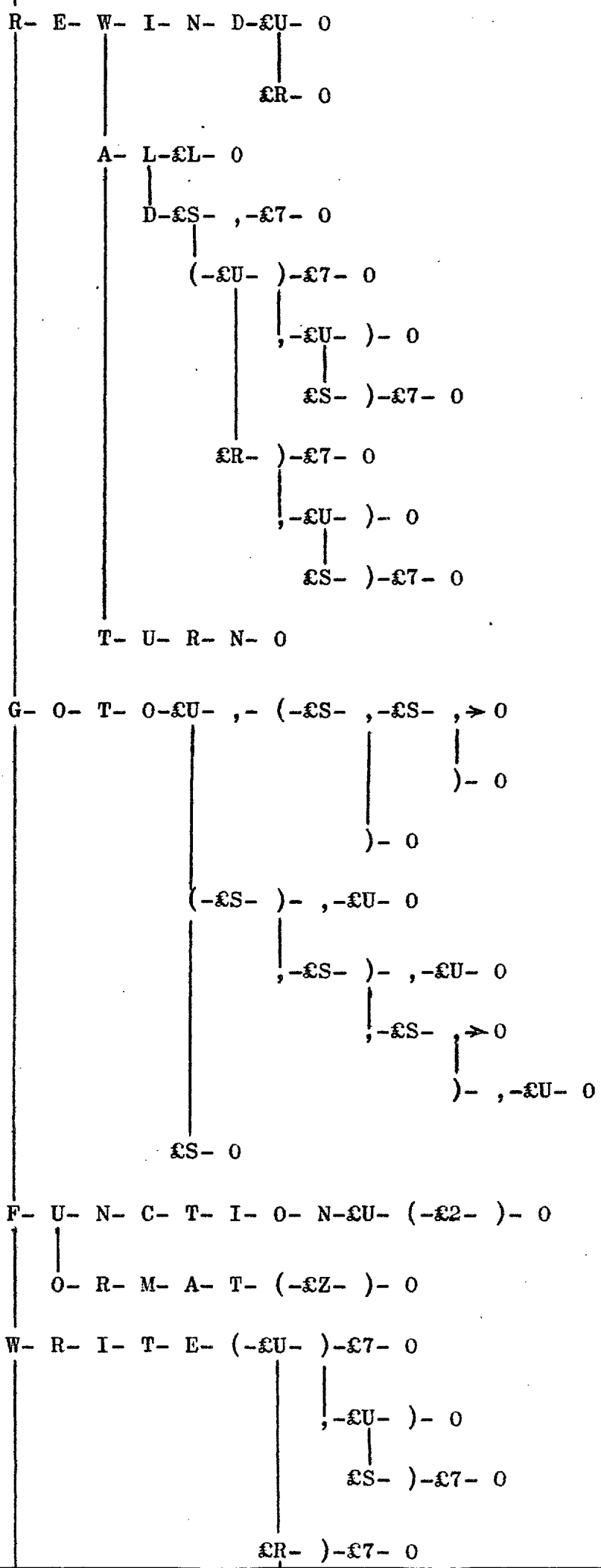
The structure is processed as two main trees (tree F for non-assignment,and tree H for assignment statements) and 30 (figs.I and II) or 31 (figs.III to VII) subtrees.

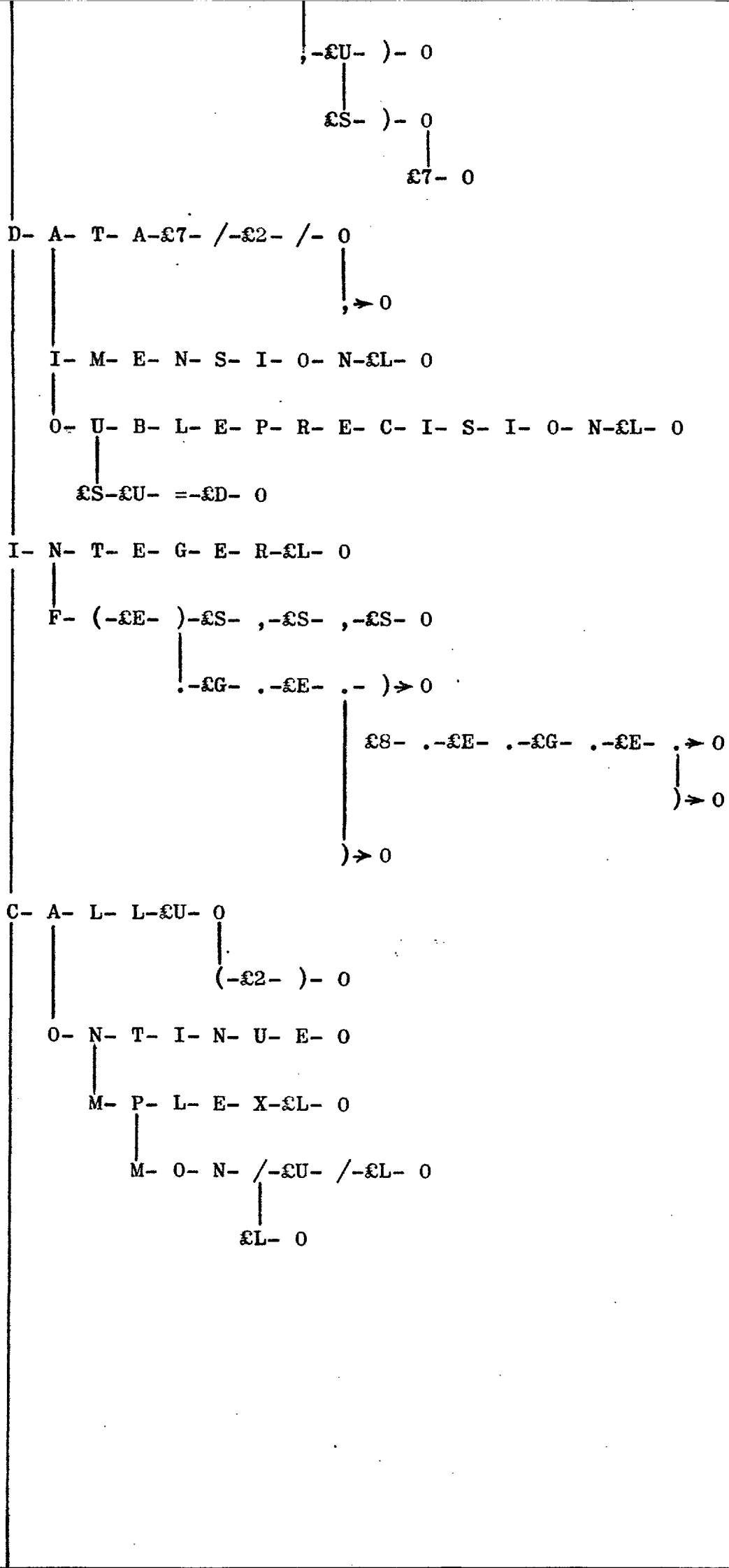
After any period of analysis the self-consistency of the resulting frequency counts is checked using routines,NUMTRE and NODECT (APP.II,RP8 and RP9 respectively).

Fig. I

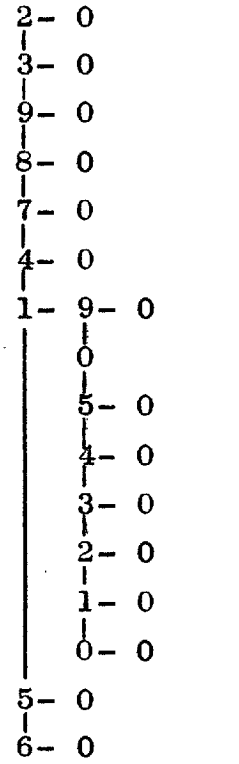
the main tree F - the initial experience tree used in the analysis of Data Set I.



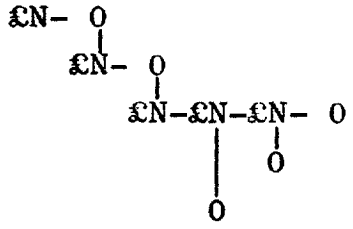

R- E- W- I- N- D-£U- 0
£R- 0
A- L-£L- 0
D-£S- ,-£7- 0
(-£U-)-£7- 0
,-£U-)- 0
£S-)-£7- 0
£R-)-£7- 0
,-£U-)- 0
£S-)-£7- 0
T- U- R- N- 0
G- O- T- O-£U- ,- (-£S- ,-£S- ,→0
)- 0
)- 0
(-£S-)- ,-£U- 0
,-£S-)- ,-£U- 0
,-£S- ,→0
)- ,-£U- 0
£S- 0
F- U- N- C- T- I- O- N-£U- (-£2-)- 0
O- R- M- A- T- (-£Z-)- 0
W- R- I- T- E- (-£U-)-£7- 0
,-£U-)- 0
£S-)-£7- 0
£R-)-£7- 0


,-£U-)- 0
£S-)- 0
£7- 0
D- A- T- A-£7- /-£2- /- 0
,→0
I- M- E- N- S- I- O- N-£L- 0
O- U- B- L- E- P- R- E- C- I- S- I- O- N-£L- 0
£S-£U- =-£D- 0
I- N- T- E- G- E- R-£L- 0
F- (-£E-)-£S- ,-£S- ,-£S- 0
.-£G- .-£E- .-)→0
£8- .-£E- .-£G- .-£E- .→0
)→0
)→0
C- A- L- L-£U- 0
(-£2-)- 0
O- N- T- I- N- U- E- 0
M- P- L- E- X-£L- 0
M- O- N- /-£U- /-£L- 0
£L- 0

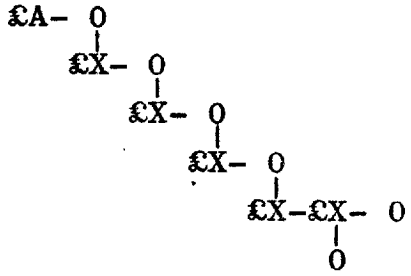
the subtree R



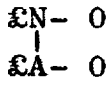
the subtree S



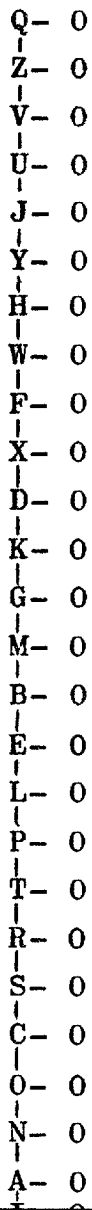
the subtree U



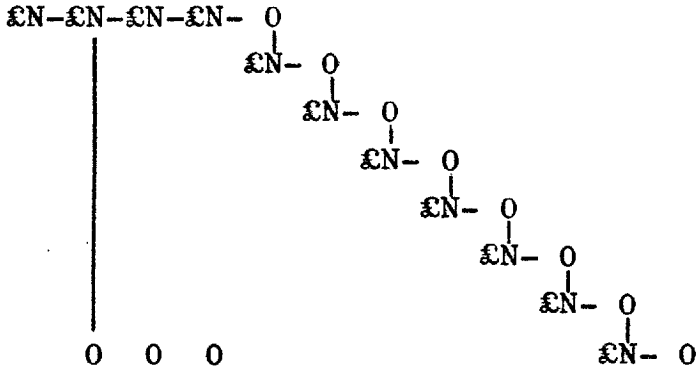
the subtree X



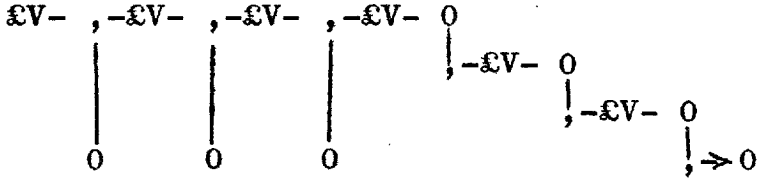
the subtree A



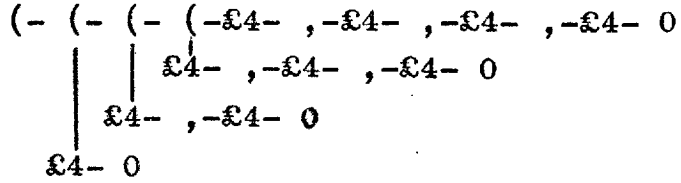
the subtree K



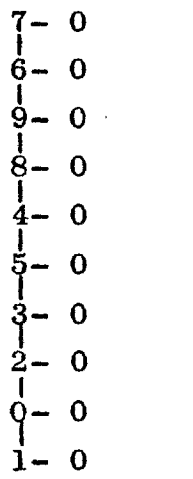
the subtree L



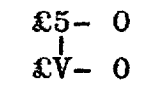
the subtree 5



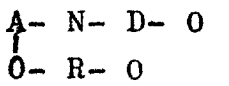
the subtree N



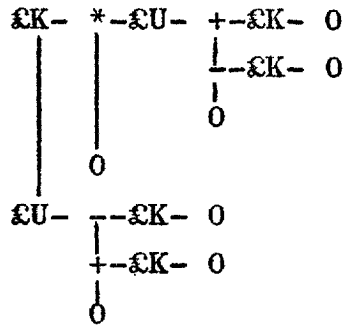
the subtree 6



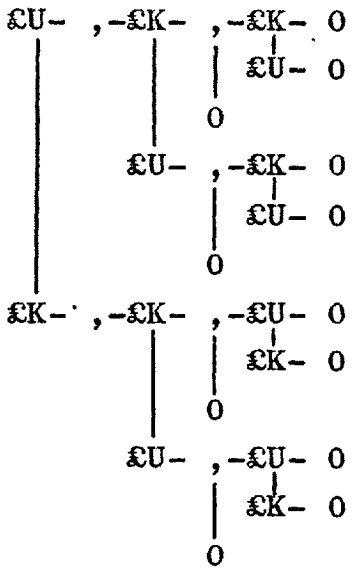
the subtree 8



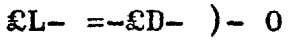
the subtree C



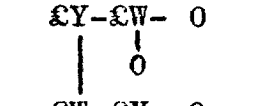
the subtree D



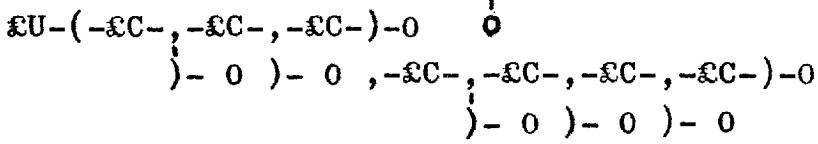
the subtree 4



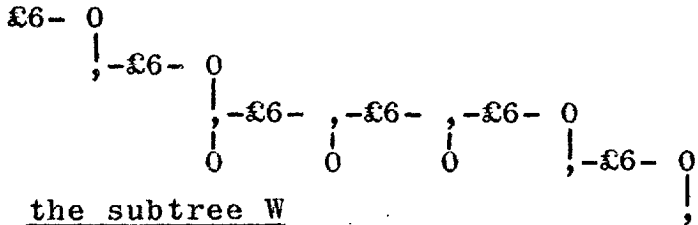
the subtree Q



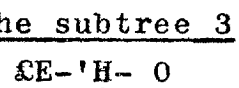
the subtree V



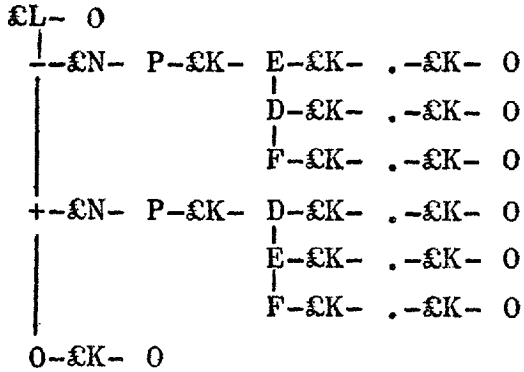
the subtree 7



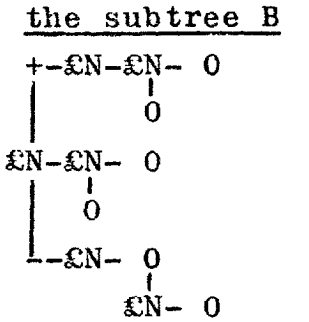
the subtree 3



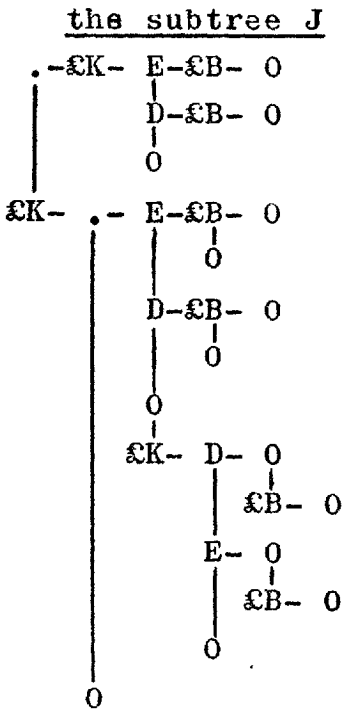
the subtree W

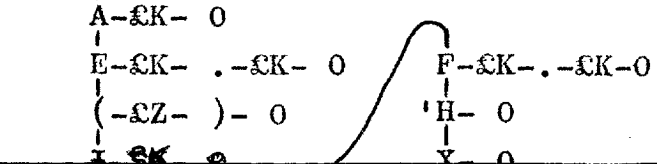


the subtree B

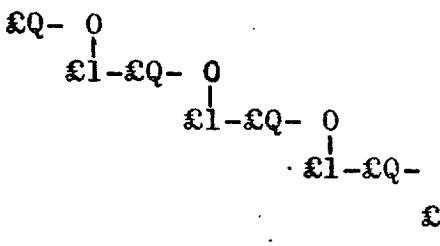


the subtree J

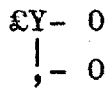




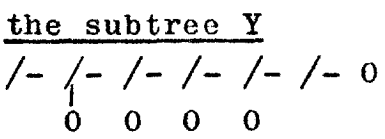
the subtree Z

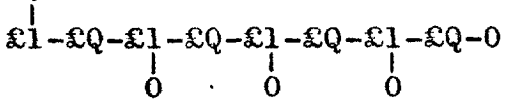


the subtree 1

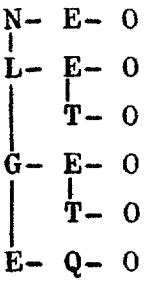


the subtree Y

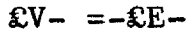




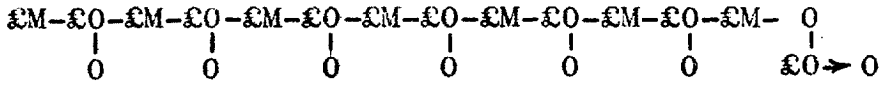
the subtree G



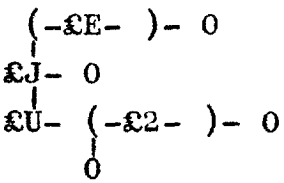
the main tree for assignment statements H



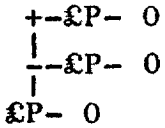
the subtree E



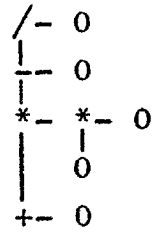
the subtree P



the subtree M



the subtree 0



the subtree 2

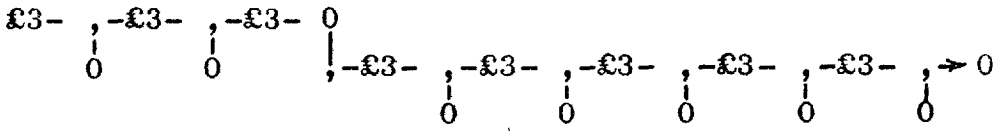
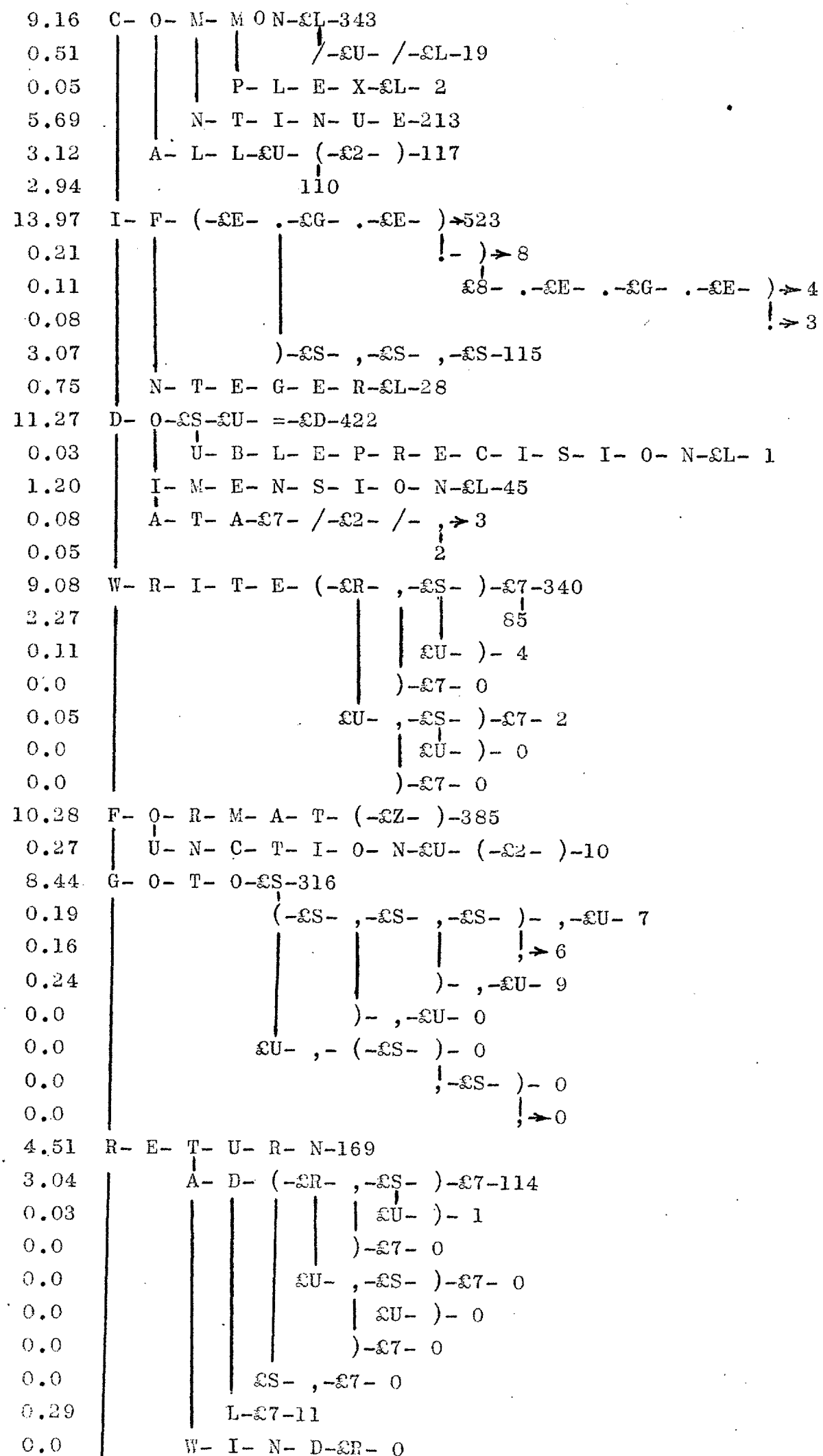


Fig. II

the main tree F - Imperial College student jobs,1969



```
0.0  |                £U- 0
4.94 E- N- D-185
0.0  |  |  |  F- I- L- E-£R- 0
0.0  |  |  |              £U- 0
0.0  |  |  T- R- Y-£U- 0
0.0  |  |              (-£2- )- 0
0.05 |  X- T- E- R- N- A- L-£L- 2
0.6  |  Q- U- I- V- A- L- E- N- C- E- (-£U➤ 0
1.95 S- T- O- P-73
0.0  |  |     £S- 0
0.91 |  U- B- R- O- U- T- I- N- E-£U- (-£2- )-34
0.80 |                                30
0.11 N- A- M- E- L- I- S- T- /-£U- /- 4
0.0  B- L- O- C- K- D- A- T- A- 0
0.0  |  A- C- K- S- P- A- C- E-£R- 0
0.0  |                        £U- 0
0.0  L- O- G- I- C- A- L-£L- 0
0.0  P- A- U- S- E- 0
0.0  |  |        £S- 0
0.0  |  U- N- C- H-£S- ,-£U➤ 0
0.0  |  R- I- N- T-£S- ,-£U➤ 0
0.0  A- S- S- I- G- N-£S- T- O-£U- 0
```

the subtree R

```
77.57 6-422
21.32 5-116
 0.18 1- 0- 1
 0.18 |  1- 1
 0.18 |  2- 1
 0.18 |  3- 1
 0.18 |  4- 1
 0.18 |  5- 1
 0.0  |  0
 0.0  |  9- 0
 0.0  4- 0
 0.0  7- 0
 0.0  8- 0
 0.0  9- 0
 0.0  3- 0
 0.0  2- 0
```

the subtree S

```
29.21 £N-£N-£N-491
16.78       |  £N-282
 0.0        |     £N- 0
37.66       633
16.36  275
```

the subtree U

```
14,12 £A-£X-£X-£X-£X-2000
11.62       |  |  |  |  £X-1646
12.73       |  |  |  1803
13.04       |  |  1848
18.87       |  2673
29.62  4196
```

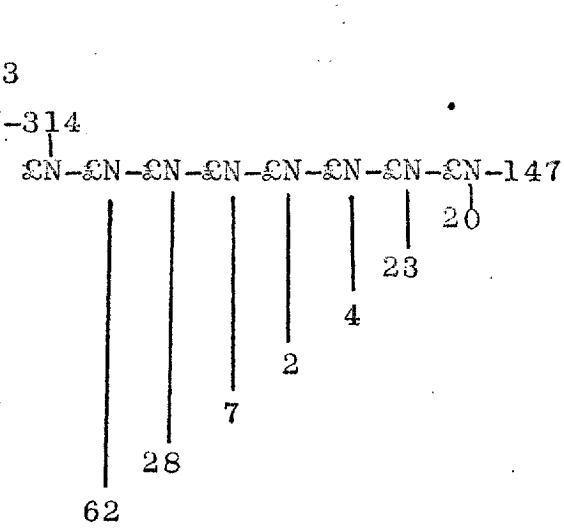
the subtree X

```
93.67 £A-26236
 6.33 £N-1772
```

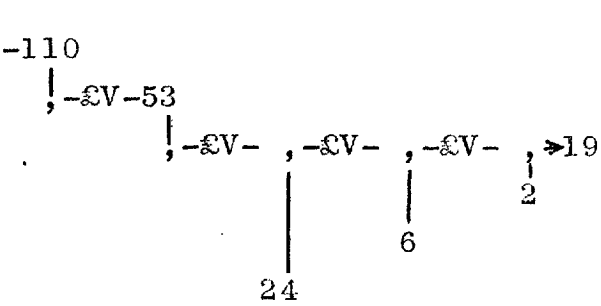
the subtree A

13.86 I-5601
7.58 A-3061
6.89 N-2783
5.89 O-2379
5.18 C-2095
5.05 S-2041
4.61 R-1861
4.56 T-1844
4.18 P-1688
4.16 L-1680
3.91 E-1579
3.58 B-1445
3.11 M-1258
2.83 G-1143
2.79 X-1127
2.69 K-1088
2.60 D-1052
2.59 F-1046
2.48 Y-1003
2.34 W-946
2.28 H-923
2.24 J-907
1.76 U-710
1.59 V-642
0.75 Z-303
0.49 Q-197

the subtree K



the subtree L



the subtree 5

100.00 (-£4-45
0.0 (-£4- ,-£4- 0
etc. no occurrences of nested implied DO loops

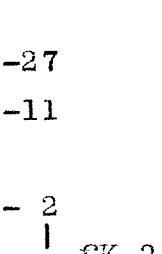
the subtree N

28.30 1-4553
21.67 0-3486
15.71 2-2527
7.93 3-1276
6.47 5-1041
5.84 4-939
4.26 8-686
3.51 9-564
3.37 6-543
2.95 7-474

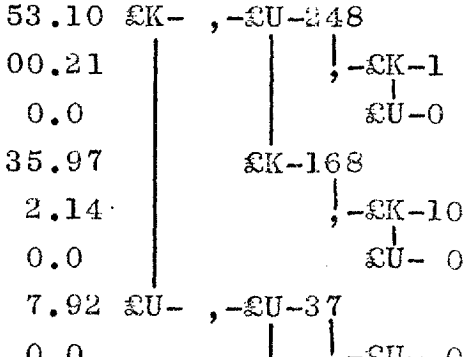
the subtree 4

100.0 £L- =-£D-)-45

the subtree C



the subtree D



the subtree 6

96.08 £V-1103
3.92 £5-45

the subtree 8

71.43 0- R- 5
28.57 A- N- D- 2

the subtree V

```
77.50 £U-4289
16.63      (-£C- )-920
 5.82           ¦-£C- )-322
 0.05                ¦-£C- ,-£C-)-3
                          etc.
```

no occurrences of three or more
than four subscripts

the subtree 7

```
28.20 £6- ,-£6- ,-£6-130
17.35       |       |      ¦-£6-80
 2.39       |       |          ¦-£6- ,-£6- ,-£6-11
 0.0        |       |          |     |     |    ¦-£6-0
 0.0        |       |          |     |     |        ¦-£6-0
 0.0        |       |          |     |     |            ¦-£6-0
 1.52       |       |          |     |     7
 0.65       |       |          |     3
18.22       |      84          |
31.67      146
```

the subtree W

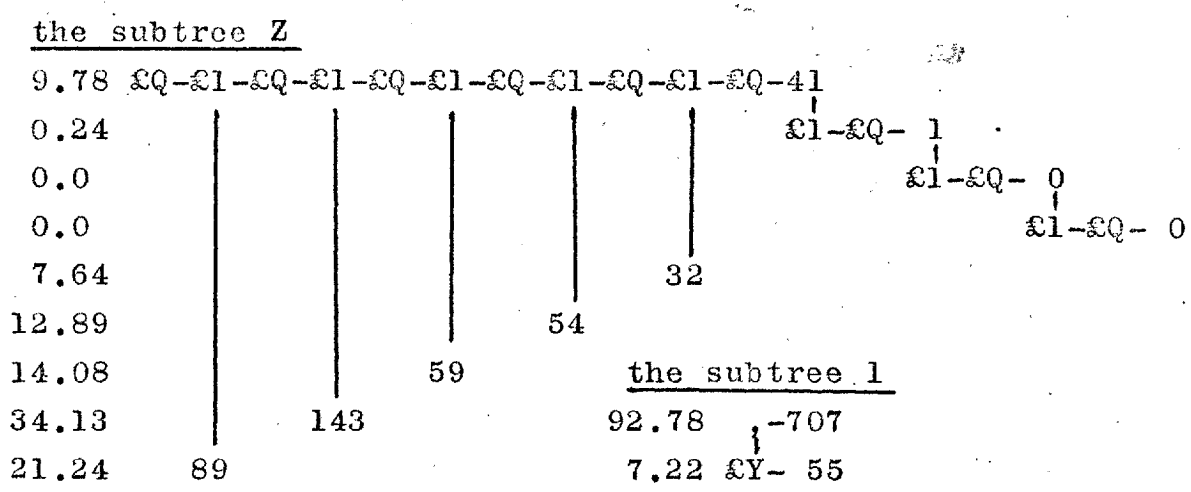
```
31.23 £K- X-351
21.80     'H-245
 5.87       F-£K- .-£K-66
 3.20       I-£K-36
 2.85       (-£Z- )-32
 1.69       E-£K- .-£K-19
 1.33       A-£K-15
 0.09       D-£K- .-£K- 1
 0.0        0-£K- 0
15.92   I-£K-179
13.97   F-£K- .-£K-157
 0.98   E-£K- .-£K-11
 0.80   A-£K- 9
 0.0    (-£Z- )- 0
 0.0    D-£K- .-£K- 0
 0.0    0-£K- 0
 0.0    +-£N- P-£K- F-£K- .-£K-0
 0.0    |              E-£K- .-£K-0
 0.0    |              D-£K- .-£K-0
 0.0    --£N- P-£K- F-£K- .-£K-0
        |              etc. as above
 0.0    L-£K- 0
```

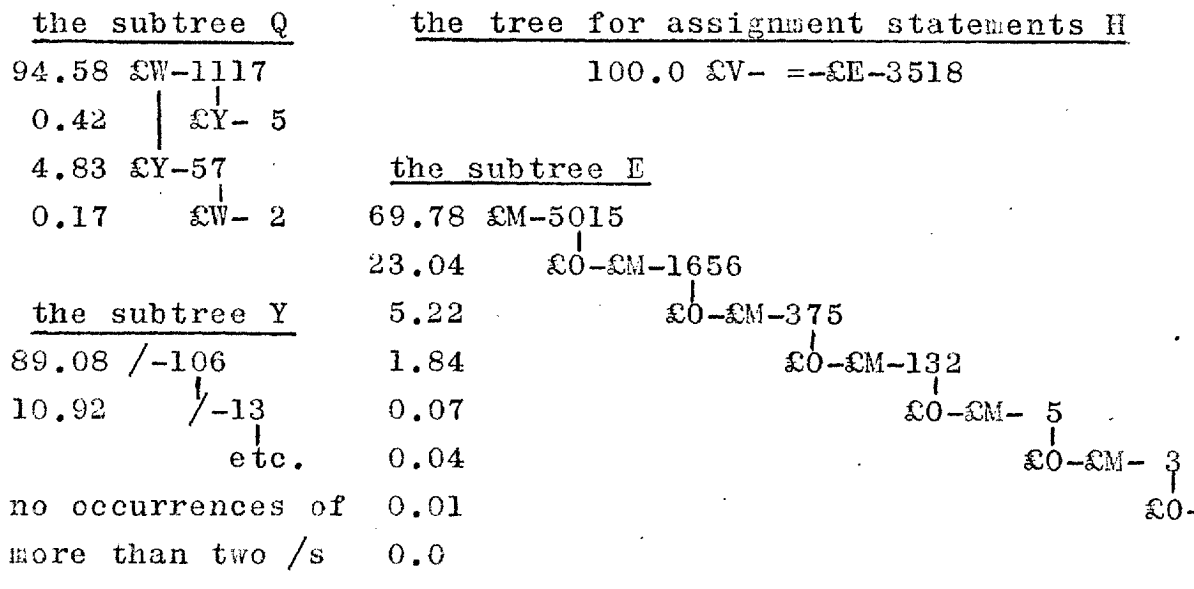
the subtree J

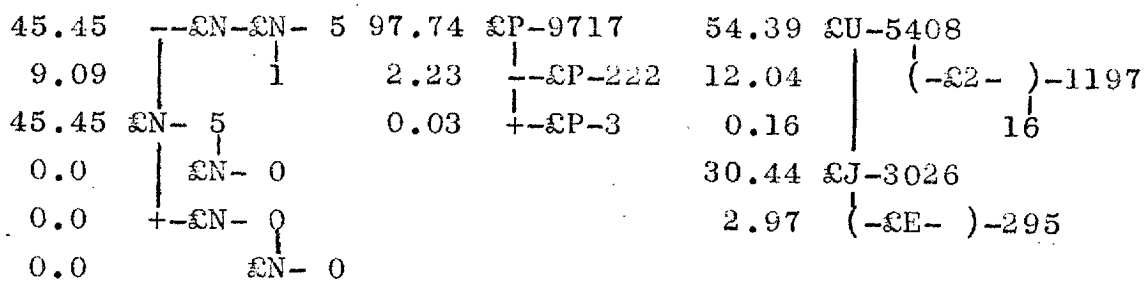
```
72.97 £K-2208
17.08    ¦-£K-517
 0.23        |   E-£B- 7
             |     0
 0.0         |
 0.13        |   D-£B- 4
             |     0
 0.0         |
      288    |
       D- 0  |
       |  £B- 0
       E- 0  |
          £B- 0
 0.07 .-£K- 2
 0.0      D-£B- 0
 0.0      E-£B- 0
```

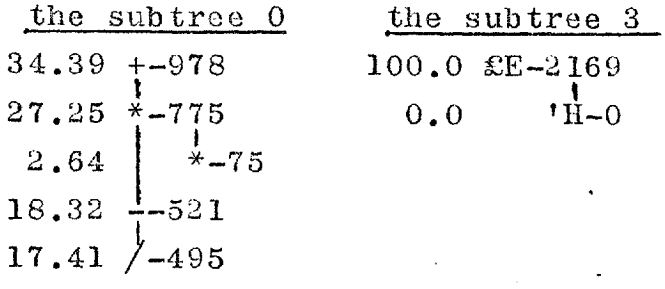
the subtree G

```
48.26 E- Q-263
20.92 G- T-114
 2.02 |  E-11
11.93 L- T-65
 4.22 |  E-23
12.66 N- E-69
```

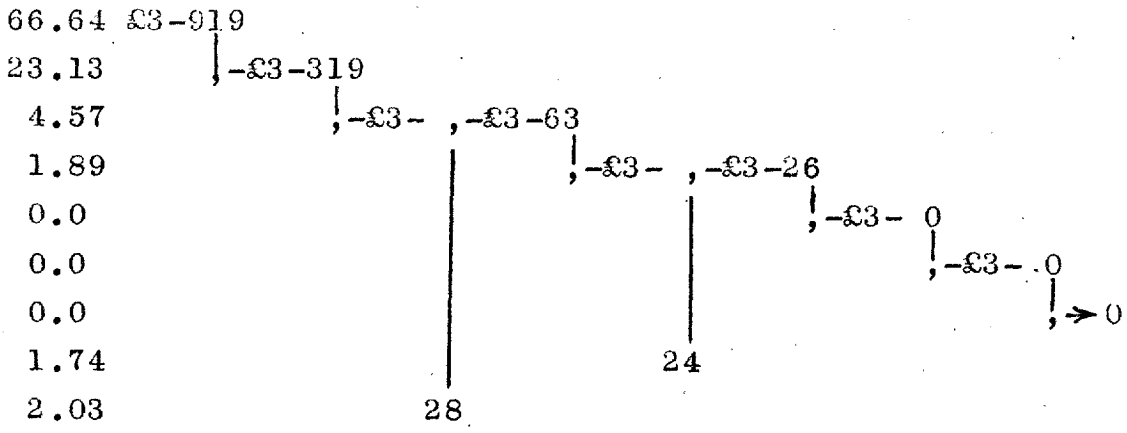

the subtree Z
9.78 £Q-£1-£Q-£1-£Q-£1-£Q-£1-£Q-£1-£Q-41
0.24 £1-£Q- 1
0.0 £1-£Q- 0
0.0 £1-£Q- 0
7.64 32
12.89 54
14.08 59
34.13 143
21.24 89
the subtree 1
92.78 -707
7.22 £Y- 55


the subtree Q
94.58 £W-1117
0.42 £Y- 5
4.83 £Y-57
0.17 £W- 2
the subtree Y
89.08 /-106
10.92 /-13
etc.
no occurrences of
more than two /s
the tree for assignment statements H
100.0 £V- =-£E-3518
the subtree E
69.78 £M-5015
23.04 £0-£M-1656
5.22 £0-£M-375
1.84 £0-£M-132
0.07 £0-£M- 5
0.04 £0-£M- 3
0.01
0.0


the subtree B
45.45 --£N-£N- 5
9.09 1
45.45 £N- 5
0.0 £N- 0
0.0 +-£N- 0
0.0 £N- 0
the subtree M
97.74 £P-9717
2.23 --£P-222
0.03 +-£P-3
the subtree P
54.39 £U-5408
12.04 (-£2-)-1197
0.16 16
30.44 £J-3026
2.97 (-£E-)-295


the subtree 0
34.39 +-978
27.25 *-775
2.64 *-75
18.32 --521
17.41 /-495
the subtree 3
100.0 £E-2169
0.0 'H-0

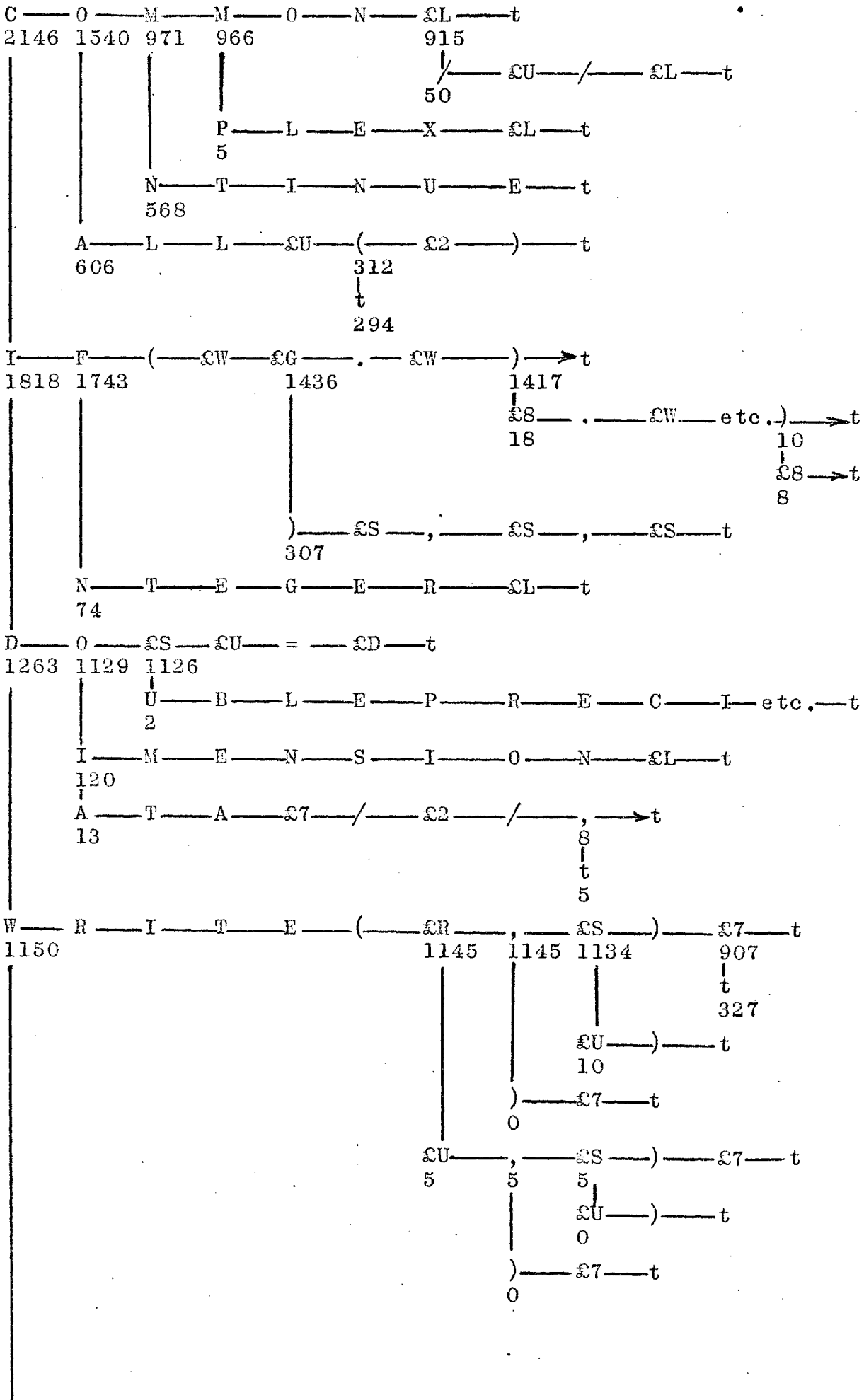
the subtree 2

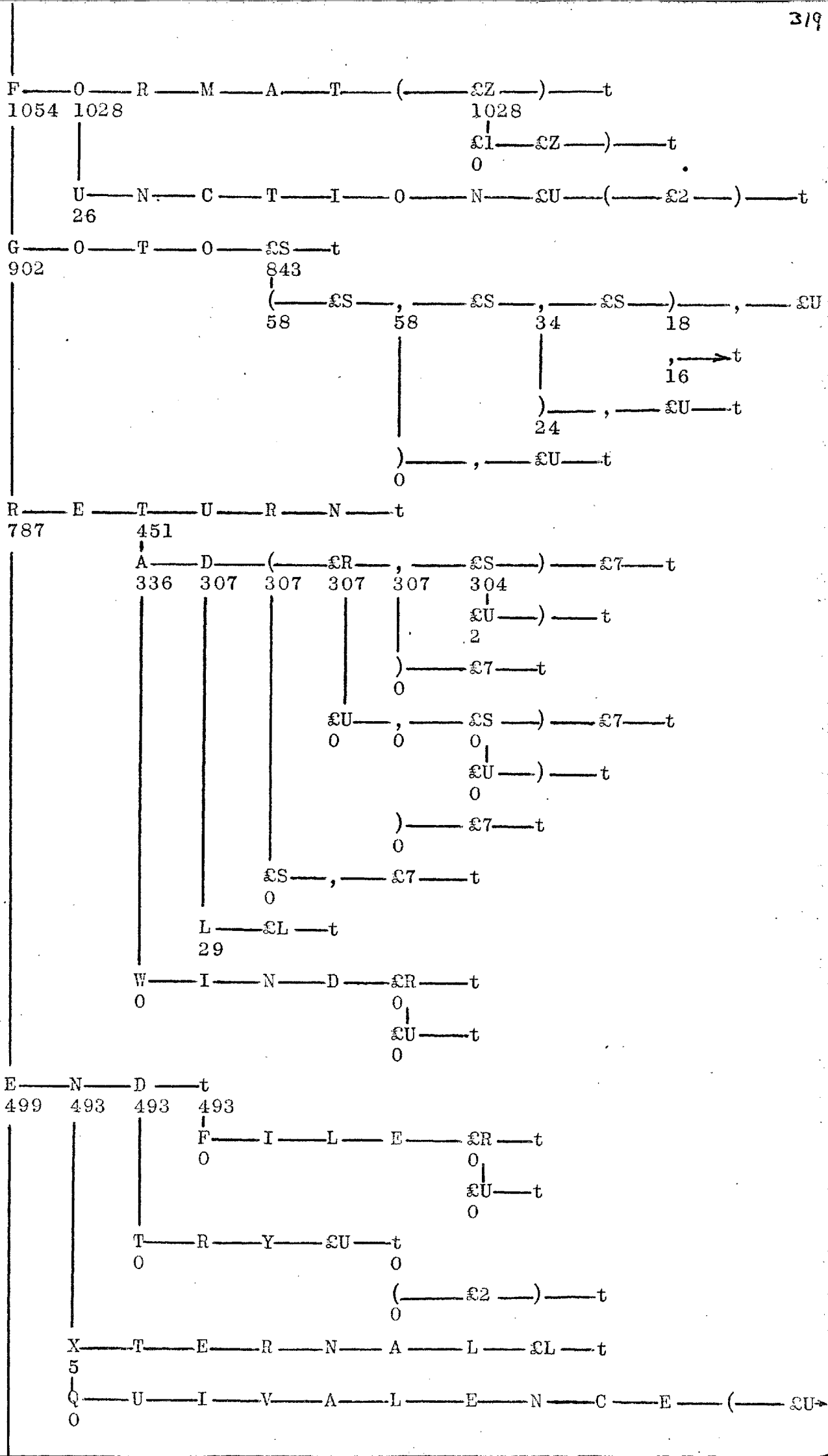


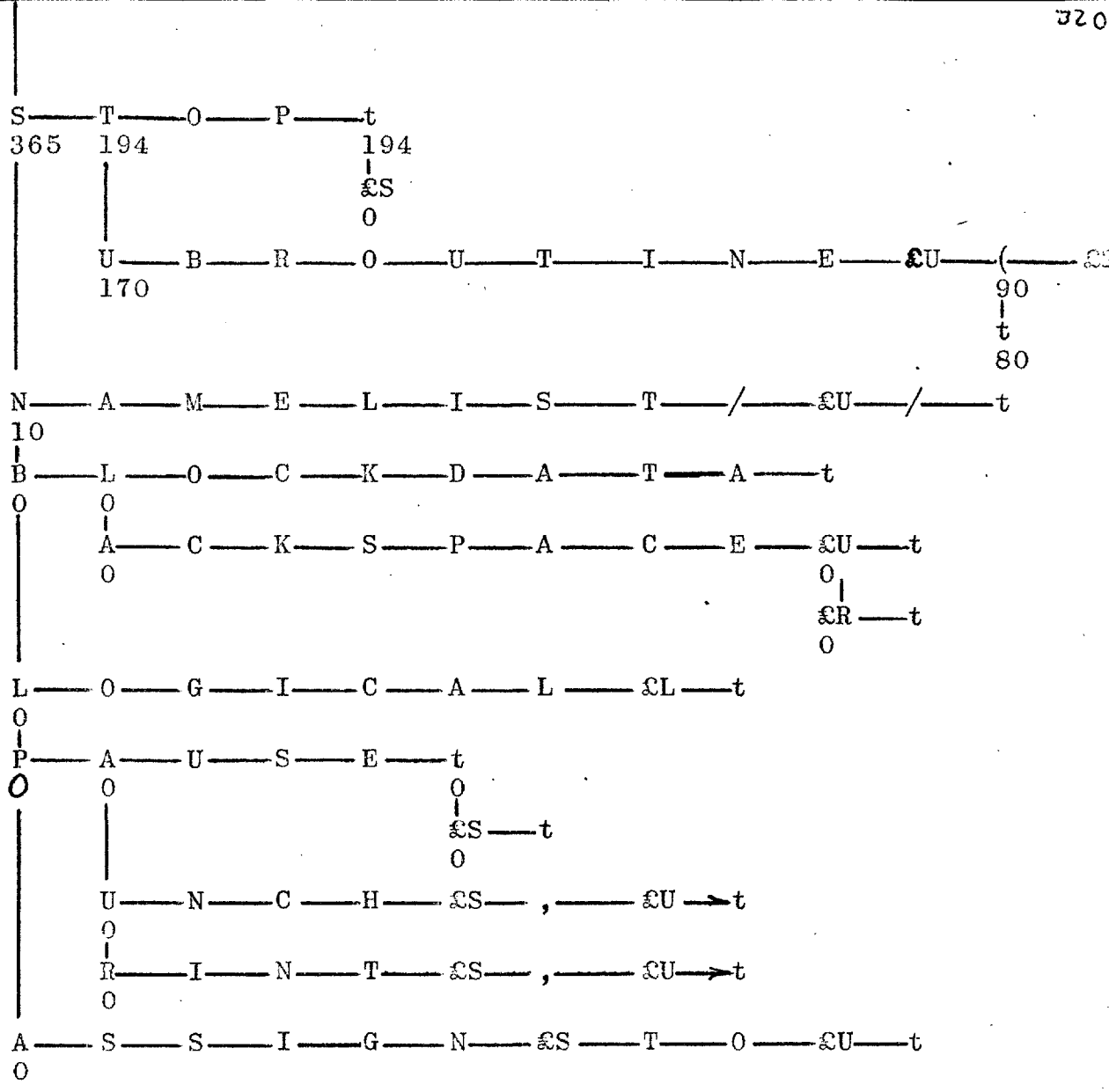
Note: the above set of trees do not form a completely self-consistent set - the subtree 6 is referenced 1158 times and entered only 1148,and the subtree M is referenced 10,030 times yet entered only 9942.This slight inconsistency is due to two minor alterations that were made to the trees during processing.

Fig. III

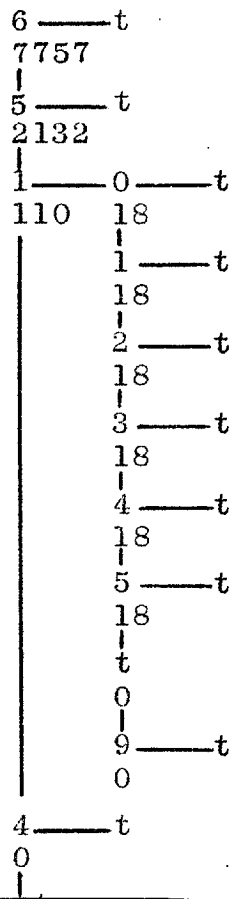
the main tree,F - Imperial College 1969,illustrating the relative frequency counts associated with each node in each filial set



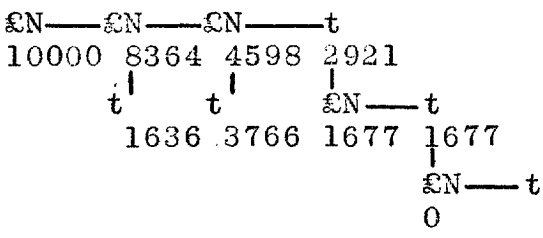

F 1054
O 1028
R
M
A
T
(£Z 1028 £1 0 £Z) t
U 26
N
C
T
I
O
N
£U
(£2) t
G 902
O
T
O
£S 843
(£S 58 , £S 58 , £S 34) 18 , £U
, t 16
) 24 , £U t
) 0 , £U t
R 787
E
T
U
R
N
t
451
A 336
D 307
(£R 307 , £S 307) £7 t
304
£U) t
2
) £7 t
0
£U 0 , £S 0) £7 t
£U) t
0
) £7 t
0
£S 0 , £7 t
L 29 £L t
W 0
I
N
D
£R 0 £U t
0
E 499
N 493
D 493
t 493
F 0
I
L
E
£R 0 £U t
0
T 0
R
Y
£U t
0
(£2) t
0
X 5
T
E
R
N
A
L
£L t
Q 0
U
I
V
A
L
E
N
C
E
(£U



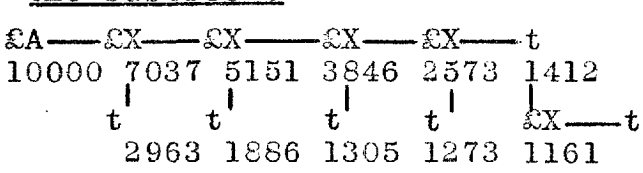
the subtree R



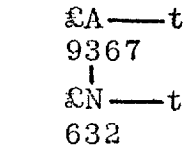
the subtree S



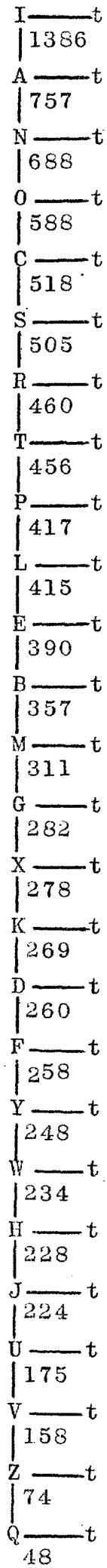
the subtree U



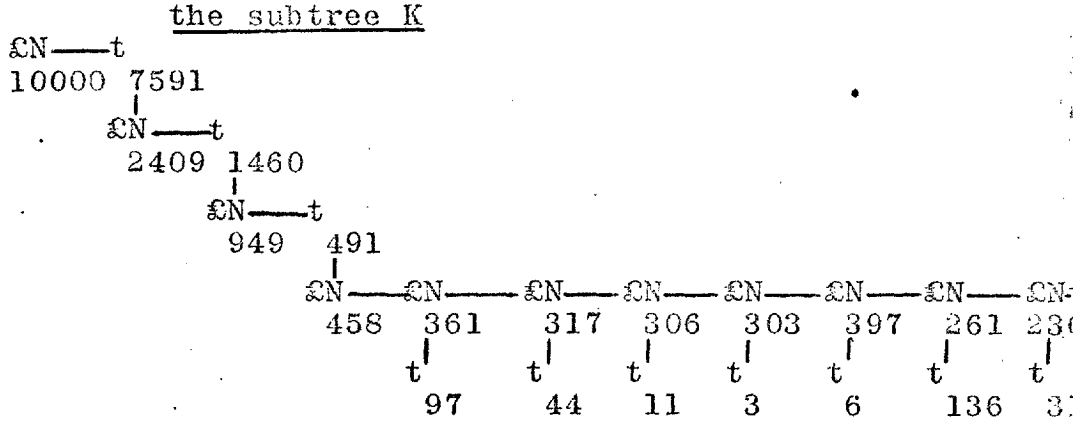
the subtree X



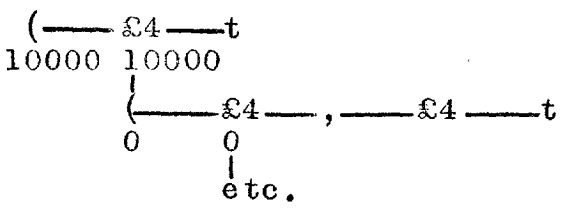
the subtree A



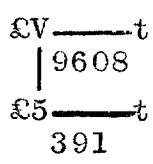
the subtree K



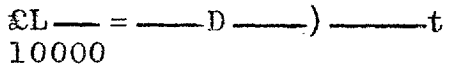
the subtree 5



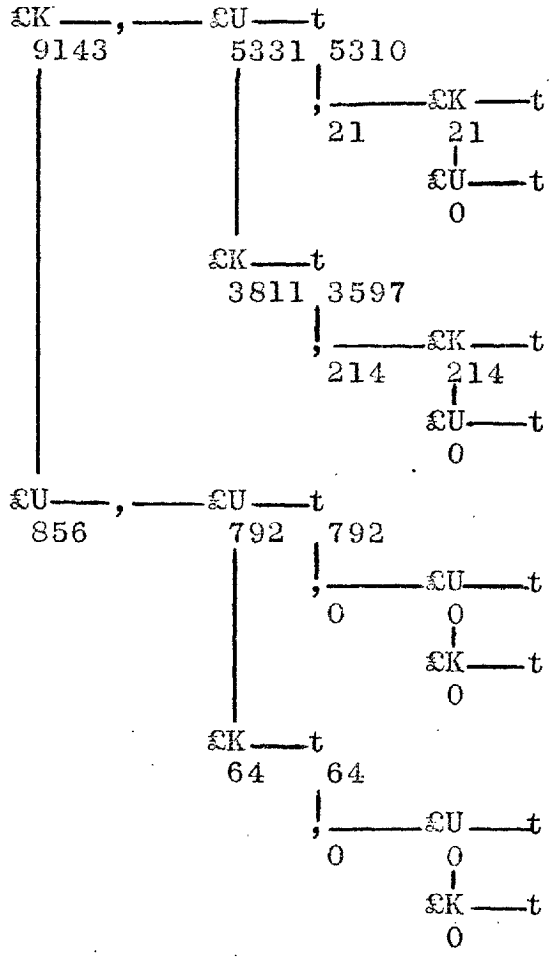
the subtree 6



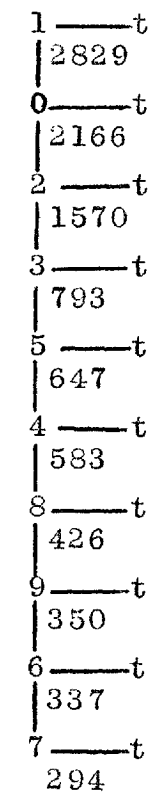
the subtree 4



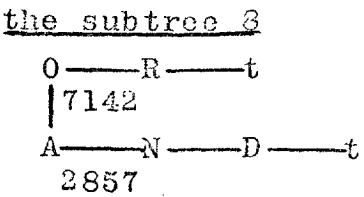
the subtree D



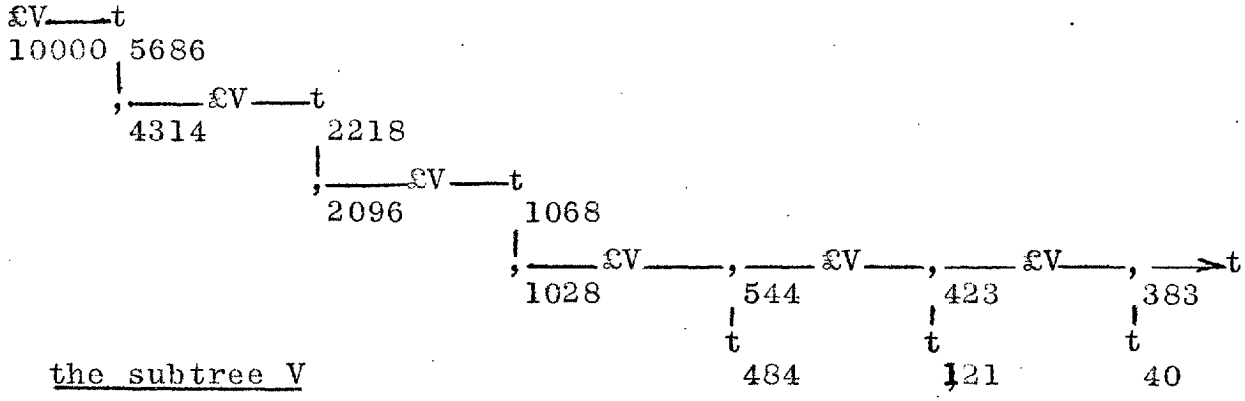
the subtree N



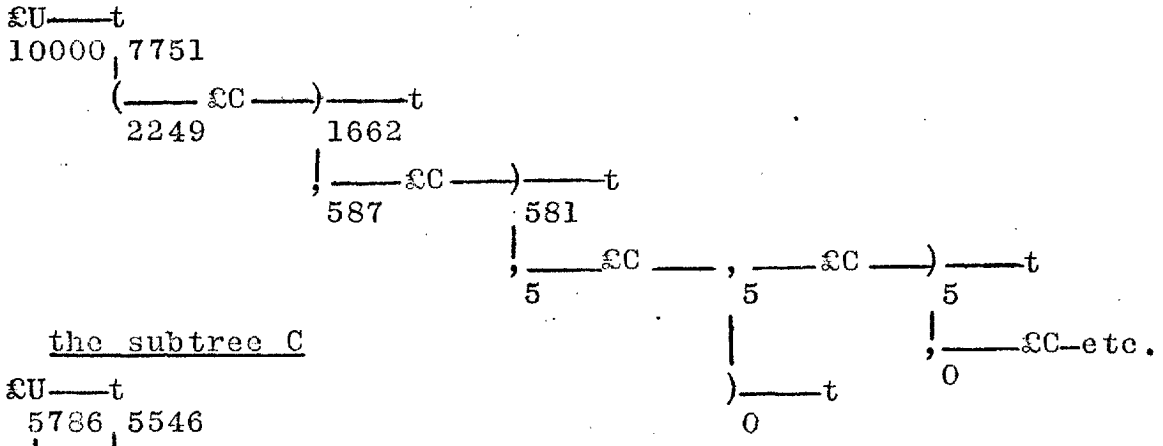
the subtree 3



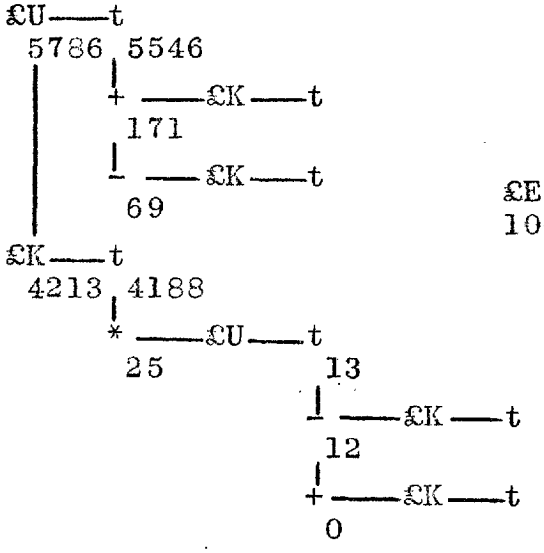
the subtree L



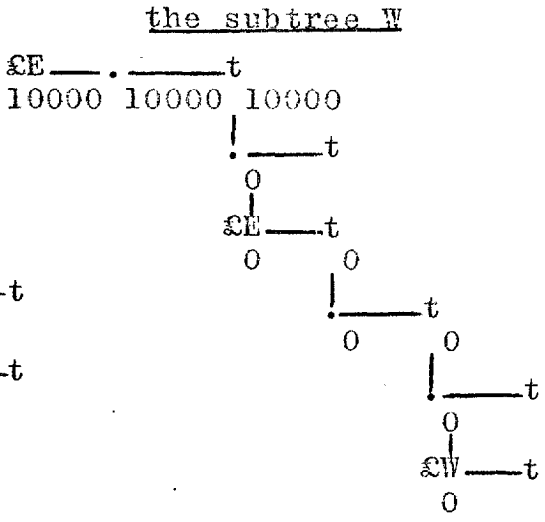
the subtree V



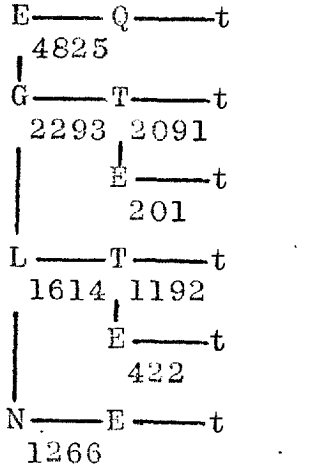
the subtree C



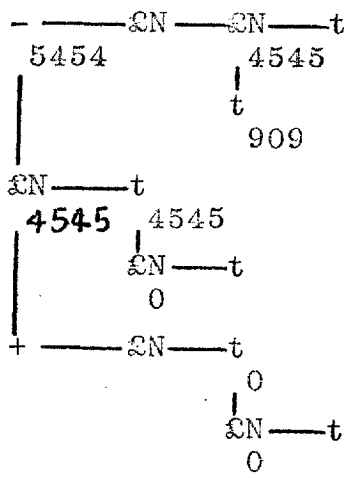
the subtree W

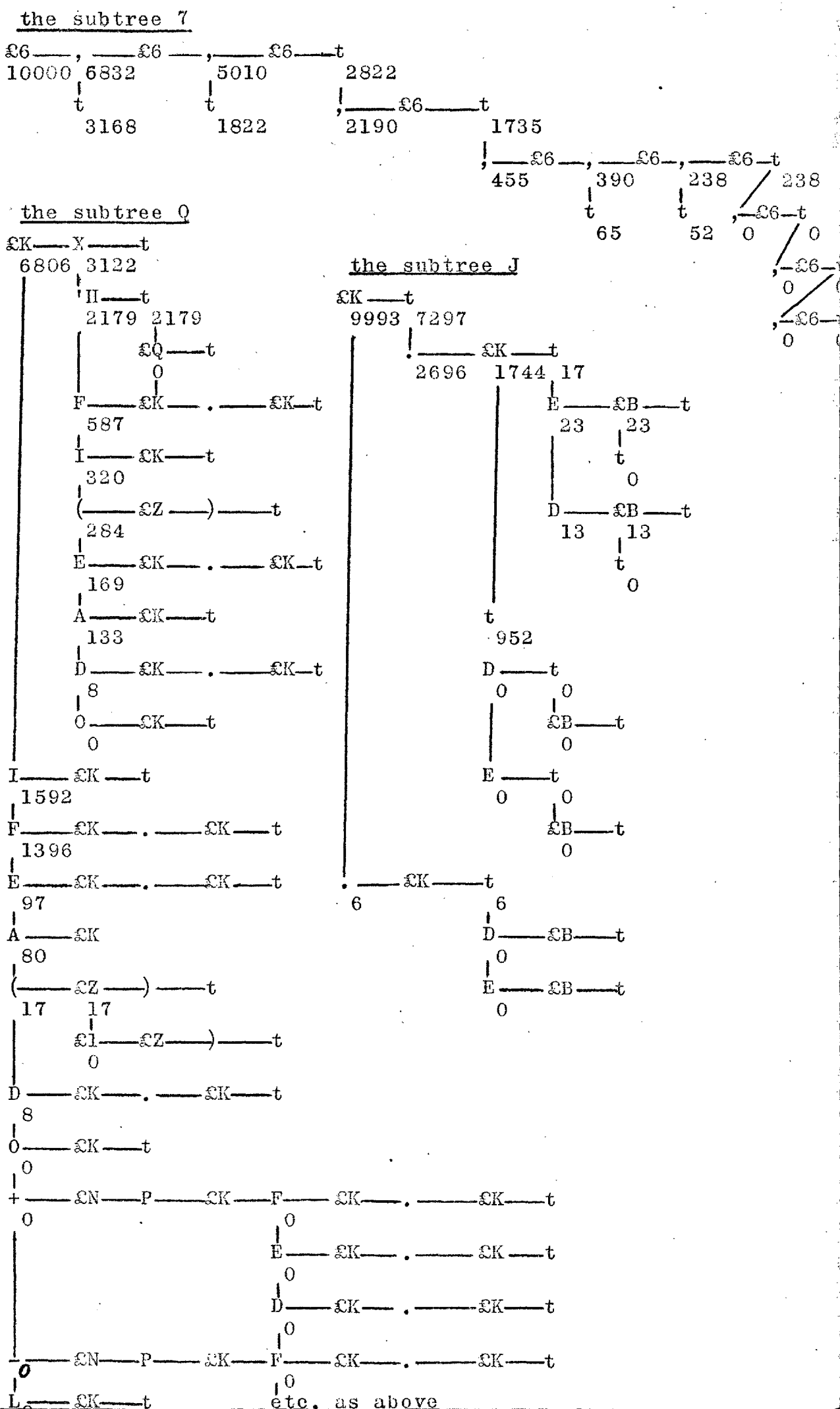


the subtree G

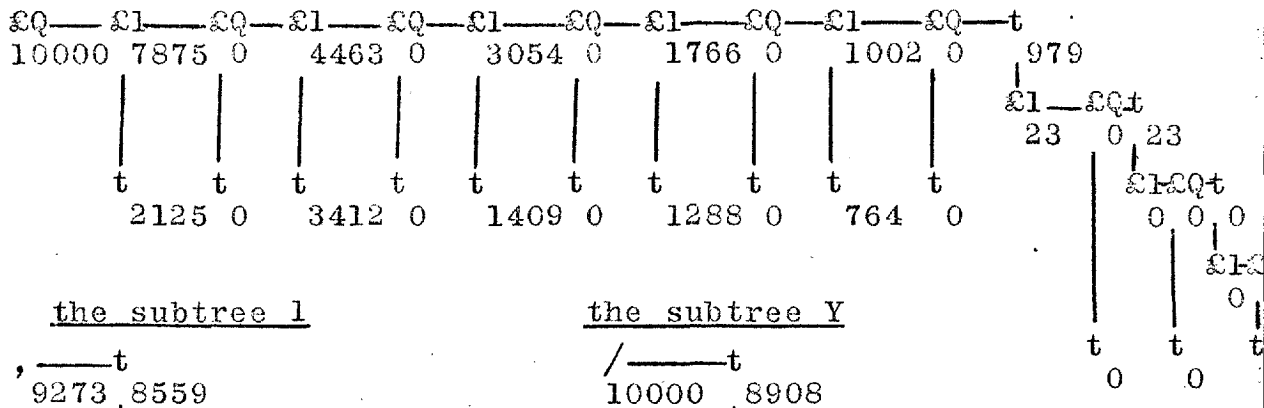


the subtree B

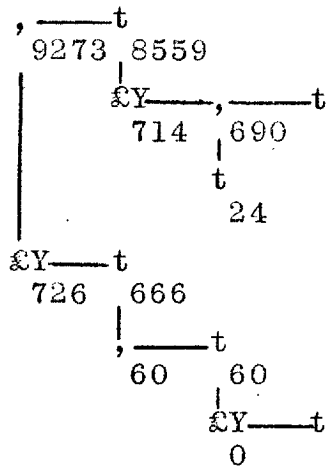



the subtree 7
the subtree 0
the subtree J
etc. as above

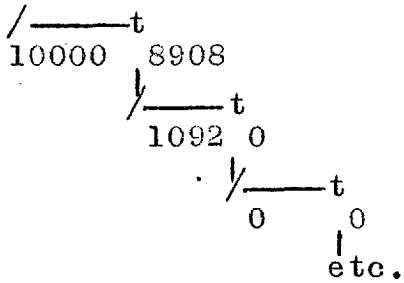
the subtree Z



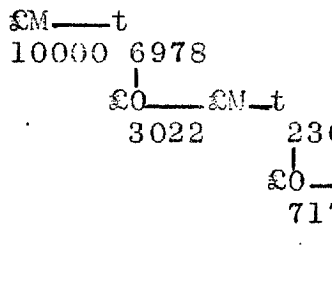
the subtree 1



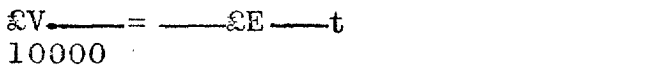
the subtree Y



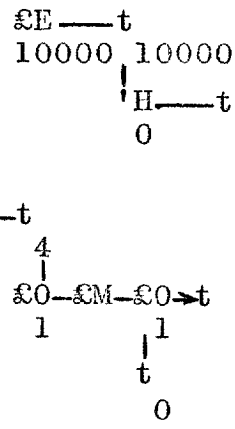
the subtree E



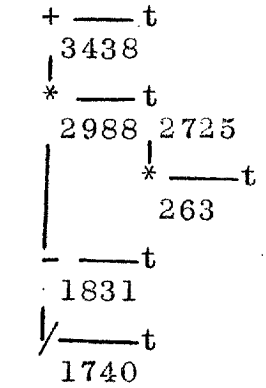
the main tree for assignment statements H



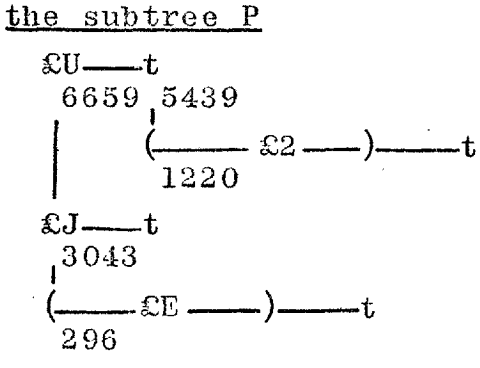
the subtree 3



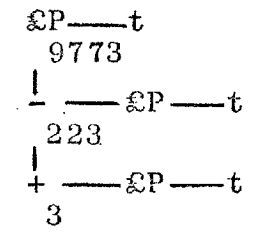
the subtree 0



the subtree P



the subtree M



the subtree 2

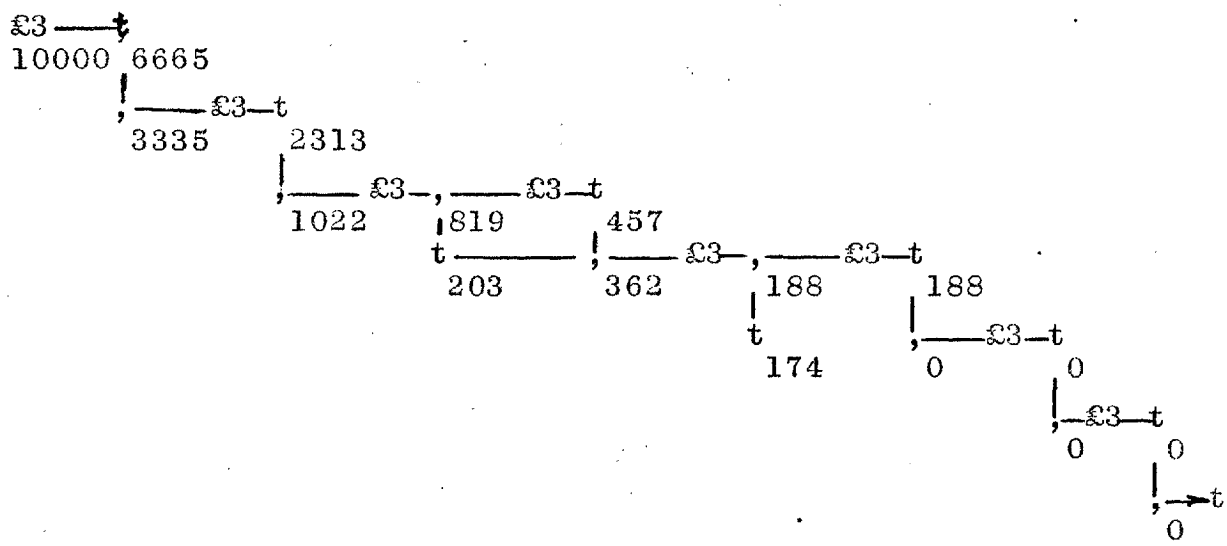
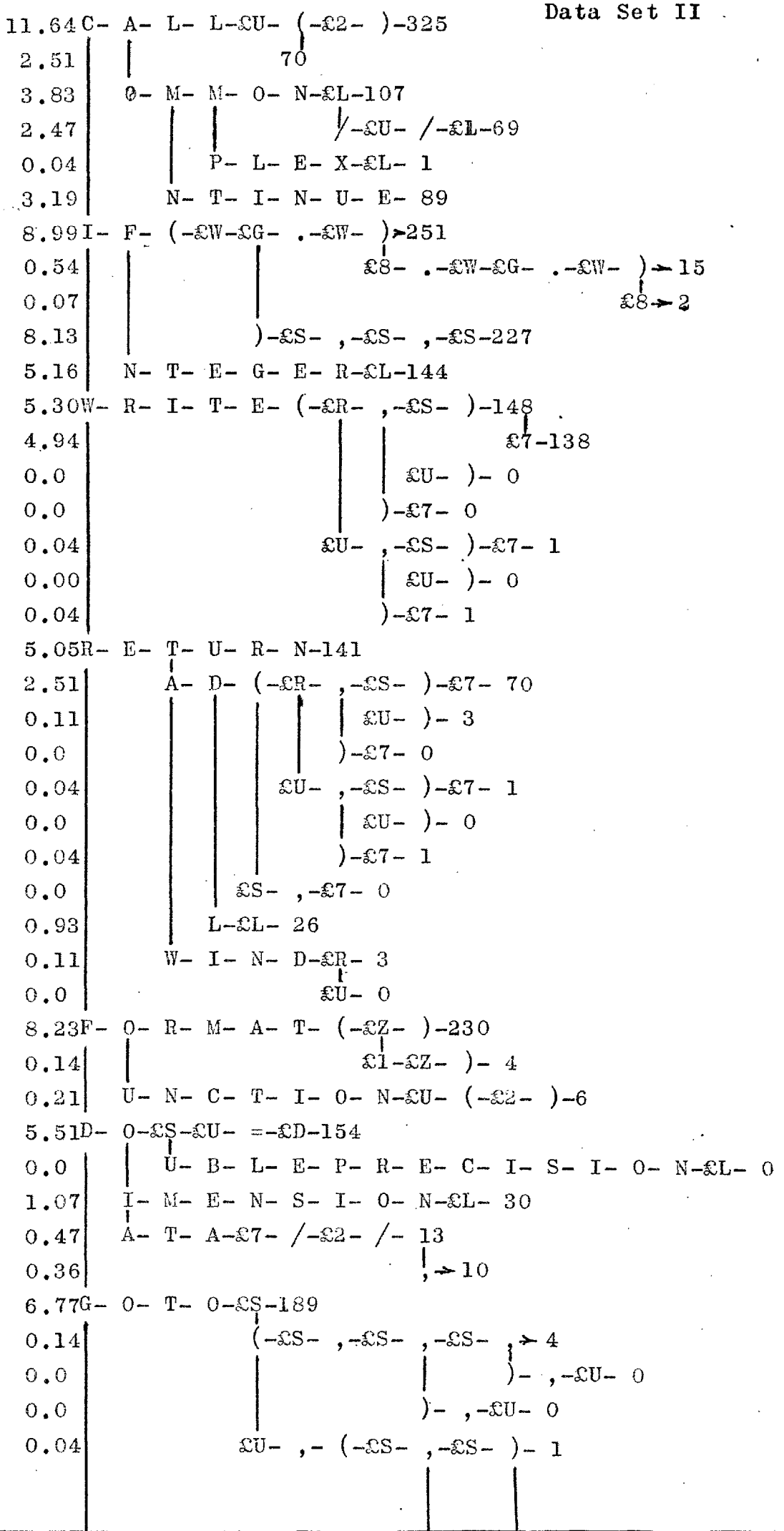


Fig. IV the main tree F - after analysis of Data Set II



```
0.0 |                                  |       ,-> 0
0.0 |                                  )- 0
5.41E- N- D-151
0.04|    |   |  F- I- L- E-£U- 1
0.0 |    |   |              £R- 0
0.04|    |   T- R- Y-£U- 1
0.04|    |                (-£2- )- 1
0.14|    Q- U- I- V- A- L- E- N- C- E- (-£U-> 4
0.0 |    X- T- E- R- N- A- L-£L- 0
2.15S- U- B- R- 0- U- T- I- N- E-£U- (-£2- )- 60
1.07|    |                                  30
1.15|    T- 0- P- 32
0.0 |           £S- 0
0.90L- 0- G- I- C- A- L-£L- 25
0.14B- L- 0- C- K- D- A- T- A- 4
0.11|    A- C- K- S- P- A- C- E-£R- 3
0.04|                          £U- 1
0.11A- S- S- I- G- N-£S- T- 0-£U- 3
0.07N- A- M- E- L- I- S- T- /-£U- /- 2
0.04P- U- N- C- H-£S- ,-£U-> 1
0.0     A- U- S- E- 0
0.0     |        £S- 0
0.0     R- I- N- T-£S- ,-£U-> 0
```

the subtree R

```
77.53 6-283
19.18 5- 70
 3.29 4- 12
 0.0  1- 0- 0
 0.0  |  1- 0
 0.0  |  2- 0
 0.0  |  3- 0
 0.0  |  4- 0
 0.0  |  5- 0
 0.0  |  0
 0.0  |  9- 0
 0.0  7- 0
 0.0  8- 0
 0.0  9- 0
 0.0  3- 0
 0.0  2- 0
```

the subtree S

```
38.71 £N-£N-£N-542
12.07       |  |  £N-169
 0.21       |  |     £N- 3
30.14       |  422
18.86    264
```

the subtree U

```
19.57 £A-£X-£X-£X-£X-1440
13.81       |  |  |  |  £X-1016
17.20       |  |  |  1266
10.99       |  |  809
11.48       |  845
26.95    1983
```

the subtree X

```
95.06 £A-16256
 4.94 £N-  845
```

the subtree A

```
10.31 I-2434
 7.95 T-1878
 7.52 N-1776
 7.38 E-1742
 6.83 A-1614
 6.11 L-1443
 6.11 R-1442
 5.67 O-1338
 5.12 S-1210
 5.09 M-1203
 3.87 P- 914
 3.82 F- 902
 3.38 D- 799
 3.10 C- 731
 2.64 B- 623
 2.53 U- 598
 2.25 J- 531
 2.17 K- 512
 1.99 X- 470
 1.54 H- 363
 1.19 G- 282
 1.17 Y- 277
 0.97 W- 229
 0.94 V- 221
 0.25 Z-  59
 0.10 Q-  24
```

the subtree K

```
67.68 £N-2342
26.11    £N-904
 4.94       £N-171
 0.72          £N-25
 0.26             £N-£N-£N-9
 0.14                   £N-5
 0.03                      £N-£N-1
 0.0                          £N-0
 0.03                      1
 0.0                    0
 0.09                3
```

the subtree L

```
27.96 £V- ,-£V- ,-£V- ,-£V- ,-£V- ,-£V- ,→137
 1.63                                   8
 9.18                            45
 8.16                        40
 6.33                    31
24.90                122
21.84            107
```

the subtree 5

```
91.36 (-£4- 74
 8.64     (-£4- ,-£4- 7
 0.0         (-£4- ,-£4- ,-£4- 0
```

etc. no occurrences of implied DO LOOPs greater than depth two.

the subtree N

```
28.89 1-2654
22.47 0-2064
13.72 2-1260
 8.83 5- 811
 7.38 3- 678
 5.16 4- 474
 4.91 6- 451
 3.52 8- 323
 2.83 7- 260
 2.29 9- 210
```

the subtree 4

```
100.0  £L- =-£D- )- 88
```

the subtree C

```
54.26 £K-529
 0.21    *-£U-2
 0.21       +-£K-2
 0.0        +-£K-0
44.41 £U-433
 0.92     +-£K-9
 0.0      +-£K-0
```

the subtree D

```
58.68 £K- ,-£K-142
 2.89          ,-£K-7
 0.0              £U-0
31.82       £U-77
 0.0            ,-£K-0
 0.0              £U-0
 4.13 £U- ,-£U-10
 0.41          ,-£U-1
 0.41             £K-1
 1.24        £K-3
 0.41           ,-£U-1
 0.0               £K-0
```

the subtree 6

```
80.99 £V-345
19.01 £5- 81
```

the subtree 8

```
52.63  A- N- D-10
47.37  0- R- 9
```

the subtree V

```
74.61 £U-2224
18.08      (-£C- )-539
 7.31           !-£C- )-218
 0.0                 !-£C- )-0
                              etc.
```

no occurrences of variables with more than two subscripts

the subtree W

```
59.22 £E-472
38.52      !-307
 2.26          £E-18
00.0             !-0
 0.0                 !-0
 0.0                    !-0
 0.0                £W-0
```

the subtree 7

```
62.55 £6-147
21.28      !-£6-50
 0.85           !-£6-  ,-£6-  ,-£6-  ,-£6-  ,-£6-  ,-£6-  ,-£6-2  !-£6
 0.85                    |       |       |       |       |       |
 0.43                    |       |       |       |       |       1
 0.43                    |       |       |       |       1
 1.28                    |       |       |       3
 2.55                    |       |       6
 3.83                    |       9
 5.96                    14
```

the subtree Q

```
33.47 £K-'H-320
 1.78    |  £Q-17
28.54    X-272
 9.97    (-£Z- )-95
 5.14    A-£K-49
 3.36    I-£K-32
 1.26    F-£K- .-£K-12
 0.31    O-£K- 3
 0.21    E-£K- .-£K- 2
 0.0     D-£K- .-£K- 0
 7.35  I-£K-70
 3.25  A-£K-31
 2.31  F-£K- .-£K-22
 1.78  E-£K- .-£K-17
 0.73  O-£K- 7
 0.52  (-£Z- )- 5
 0.6     £1-£Z- )- 0
 0.0   +-£N- P-£K- F-£K-.-£K-0
 0.0                E-£K-.-£K-0
 0.0                D-£K-.-£K-0
 0.0   --£N- P-£K- F-£K-.-£K-0
                    etc.as above
 0.0   L-£K-0
```

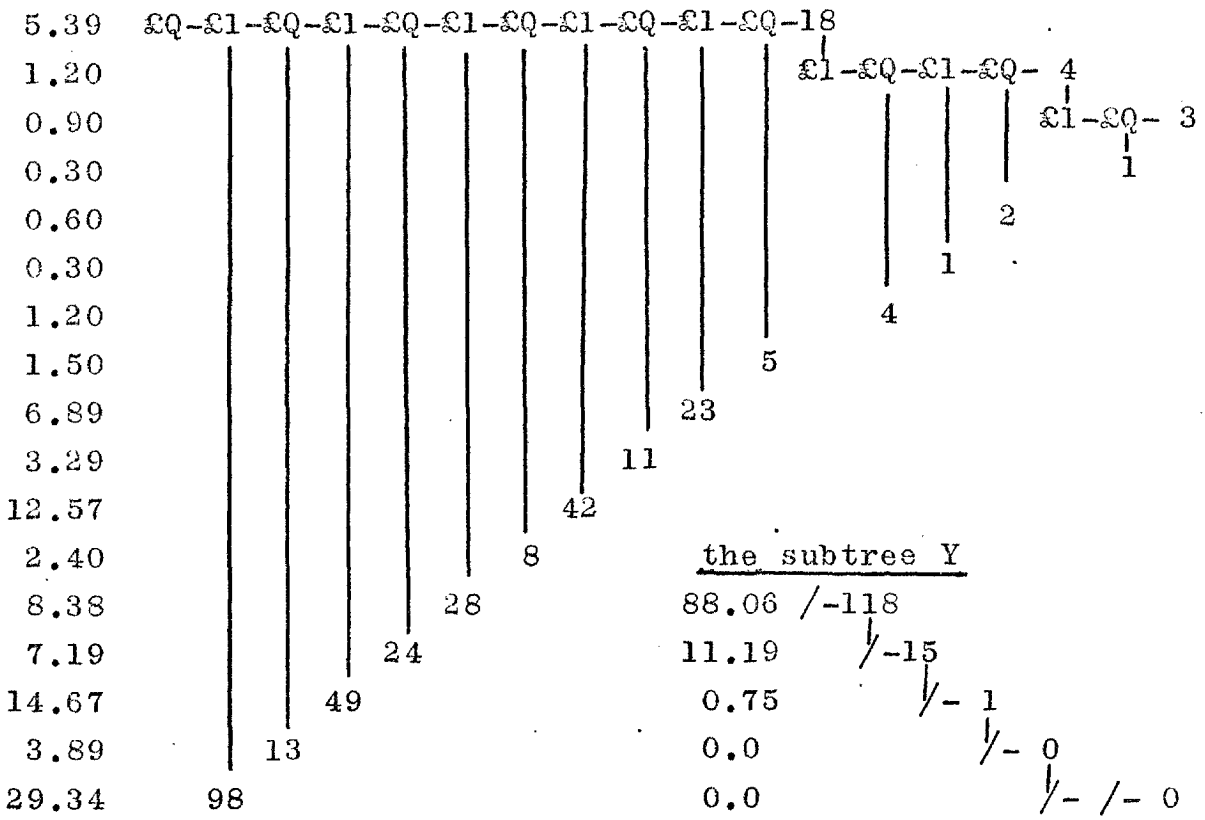
the subtree J

```
86.42 £K-1145
 7.32    !-£K-97
 0.68          E-£B-9
 0.6             0
 0.0           D-£B-0
 0.6             0
 5.36      71
 0.0         D- 0
 0.0           £B- 0
 0.0         E- 0
 0.0         E-£B- 0
 0.23  .-£K- 3
 0.0         D-£B- 0
 0.0         E-£B- 0
```

the subtree G

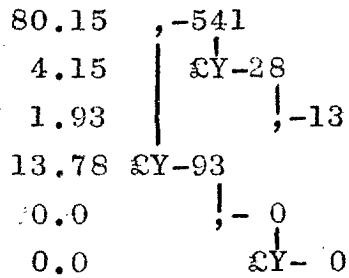
```
51.58 E- Q-147
20.00 G- T-57
 3.86    E-11
 9.82  L- T-28
 5.26    E-15
 9.47  N- E-27
```

the subtree Z

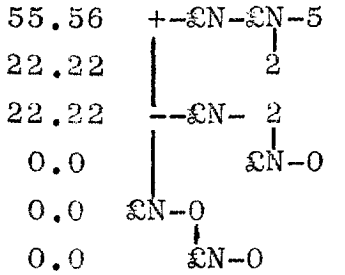


the subtree Y

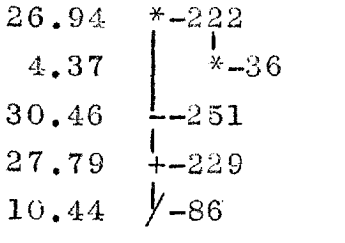
the subtree 1



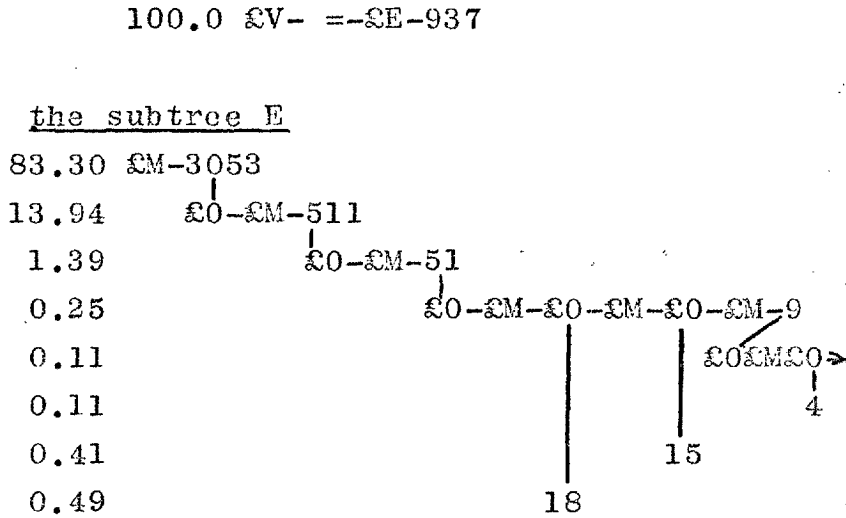
the subtree B



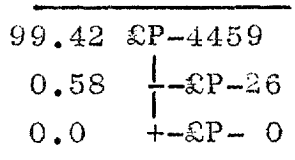
the subtree O



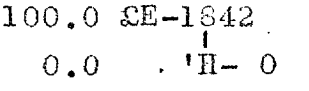
the tree for assignment statements H



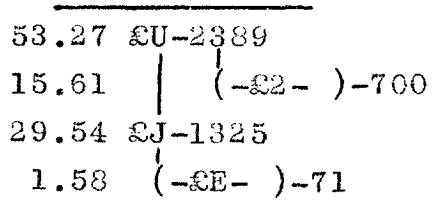
the subtree M



the subtree 3



the subtree P



the subtree 2

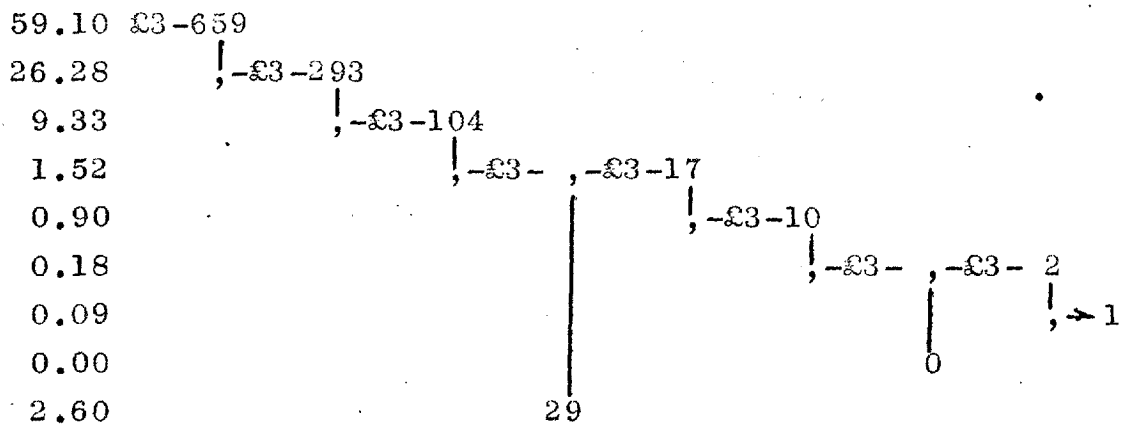
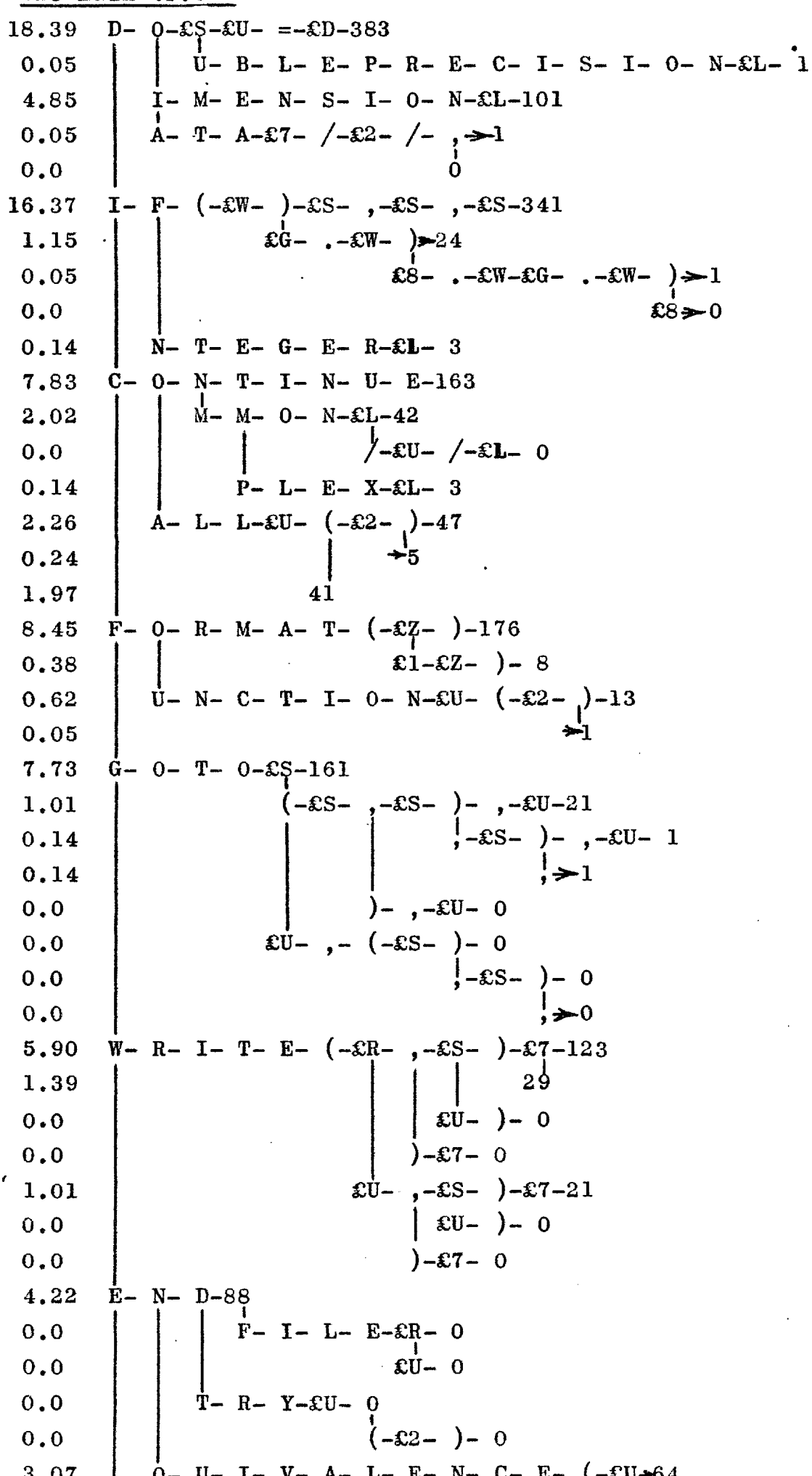


Fig. V - the total experience tree after the analysis of Data Set III, and optimisation

the main tree F



```
0.34   X- T- E- R- N- A- L-£L- 7
3.74 R- E- T- U- R- N-78
2.59       A- D- (-£R- ,-£S- )-£7-54
0.0                      £U- )- 0
0.0                     )-£7- 0
0.62                £U- ,-£S- )-£7-13
0.0                          £U- )- 0
0.0                         )-£7- 0
0.0              £S- ,-£7- 0
0.14          L-£L- 3
0.0        W- I- N- D-£R- 0
0.0                   £U- 0
1.34 S- U- B- R- O- U- T- I- N- E-£U- (-£2- )-28
0.19                                    4
1.15                                24
0.19    T- O- P- 4
0.0               £S- 0
0.05 B- L- O- C- K- D- A- T- A- 1
0.0     A- C- K- S- P- A- C- E-£U- 0
0.0                              £R- 0
0.0  N- A- M- E- L- I- S- T- /-£U- />0
0.0  L- O- G- I- C- A- L-£L- 0
0.0  P- A- U- S- E- 0
0.0                £S- 0
0.0     U- N- C- H-£S- ,-£U>0
0.0     R- I- N- T-£S- ,-£U>0
0.0  A- S- S- I- G- N-£S- T- O-£U- 0
```

the subtree R

```
48.06 3-99
18.93 2-39
14.08 1- 5-29
 1.46    4- 3
 0.0     2- 0
 0.0     3- 0
 0.0     0- 0
 0.0     1- 0
 0.0     0
 0.0     9- 0
11.17 6-23
 5.83 5-12
 0.49 7- 1
 0.0  9- 0
 0.0  4- 0
 0.0  8- 0
```

the subtree S

```
55.65 £N-£N-1039
22.82       £N-426
 0.80           £N-15
 0.0                £N- 0
20.73    387
```

the subtree U

```
 9.17 £A-£X-£X-£X-£X-1013
 1.85                   £X-204
 6.21                686
10.64             1175
23.20          2562
48.92       5401
```

the subtree X

```
86.15 £A-10374
13.85 £N-1668
```

the subtree A

```
 9.19  A-1967
 8.56  I-1833
 8.29  N-1775
 5.16  W-1105
 4.77  T-1021
 4.47  M-958
 4.46  D-956
 4.45  R-952
 4.40  S-943
 4.27  E-914
 4.17  K-892
 3.86  C-827
 3.80  J-813
 3.61  L-773
 3.60  X-772
 3.51  F-751
 3.39  B-727
 2.94  P-629
 2.10  G-449
 2.05  V-440
 2.03  O-434
 1.98  Y-424
 1.57  U-336
 1.39  Z-298
 1.30  H-279
 0.69  Q-147
```

the subtree K

```
76.95 £N-4461
17.04     £N-988
 3.85         £N-223
 0.74             £N-£N-43
 0.21                 £N-£N-£N-12
 0.03                     £N-£N- 2
 0.0                          £N- 0
 0.0                      0
 0.16                 9
 0.12                 7
 0.90             52
```

the subtree L

```
15.04 £V- ,-£V- ,-£V- ,-£V- ,-£V- ,-£V- ,-34
 0.88                                    2
 5.31                            12
 3.10                        7
23.89                    54
26.55                60
25.22            57
```

the subtree 5

```
88.14 (-£4-52
11.86     (-£4- ,-£4- 7
 0.0          (-£4- ,-£4- ,-£4- 0
                  etc. no occurrences of implied
DO loops nested to a depth of greater than two
```

the subtree N

```
28.12  1-3745
16.62  0-2214
15.87  2-2114
 9.93  3-1323
 7.94  5-1057
 6.94  4-924
 5.35  6-713
 3.18  7-424
 3.12  9-415
 2.92  8-389
```

the subtree 4

```
100.0 £L- =-£D- )-66
```

the subtree C

```
55.13 £K-1080
 0.31     *-£U- 6
 0.31          +-£K- 6
 0.10          +-£K- 2
41.45 £U-812
 2.04     +-£K-40
 0.66     +-£K-13
```

the subtree D

```
56.79 £K- ,-£U-255
 0.45          ,-£K-2
 0.0               £U-0
36.08      £K-162
 1.78          ,-£K-8
 0.0               £U-0
 3.34 £U- ,-£U-15
 0.67          ,-£U-3
 0.0               £K-0
 0.89       £K- 4
 0.0           ,-£U-0
 0.0               £K-0
```

the subtree 6

```
87.53 £V-414
12.47 £5-59
```

the subtree 8

```
100.0 0- R- 1
  0.0 A- N- D- 0
```

the subtree V

```
50.77 £U-1640
37.80      (-£C- )-1221
11.42            |-£C- )-369
 0.0             |      |-£C- )-0
                                etc.
```

no occurrence of subscripted variables with more than two subscripts

the subtree W

```
82.28 £E-325
 9.87      |-£E-39
 0.51      |    |    |-£W- 2
 0.0       |    |    |   |- 0
 0.0       |    |    0
 7.09      |   28
 0.25      |    |- 1
```

the subtree 7

```
1.42 £6- ,-£6- ,-£6- ,-£6- ,-£6- ,-£6- ,-£6- ,-£6- ,-£6- 3
0.47      |      |      |      |      |      |      |      |-£6-
0.94      |      |      |      |      |      |      2
0.47      |      |      |      |      |      1
2.83      |      |      |      |      6
3.30      |      |      |      7
7.55      |      |      16
12.74     |      27
22.64     48
47.64    101
```

the subtree Q

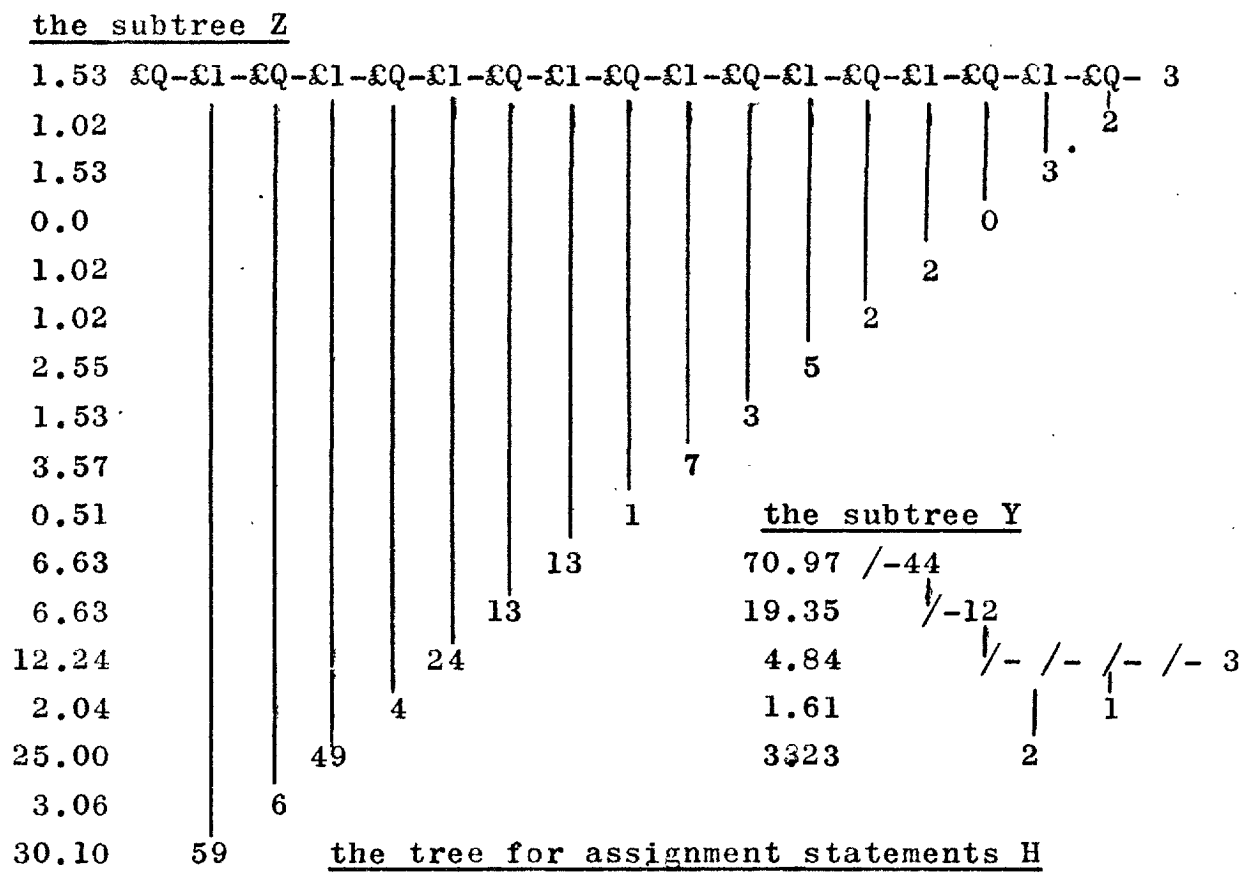
```
33.33 £K-'H-182
 6.41  |   |  £Q-35
 8.79  |   X-48
 8.06  |   F-£K- .-£K-44
 3.85  |   E-£K- .-£K-21
 3.66  |   I-£K-20
 0.92  |   A-£K- 5
 0.73  |   (-£Z- )- 4
 0.55  |   D-£K- .-£K- 3
 0.0   |   0-£K- 0
12.09  I-£K-66
12.09  F-£K- .-£K-66
 6.96  E-£K- .-£K-38
 1.28  (-£Z- )- 7
 0.18  |  £1-£Z- )- 1
 0.92  A-£K- 5
 0.18  D-£K- .-£K- 1
 0.0   0-£K- 0
 0.0   +-£N- P-£K- F-£K- .-£K-0
 0.0   |          E-£K- .-£K-0
 0.0   |          D-£K- .-£K-0
 0.0   |-£N- P-£K-etc as above
                  three items
```

the subtree J

```
72.76 £K-2041
13.40  |   |-£K-376
 1.64  |   |    E-£B-46
 0.0   |   |    |  0
 0.0   |   |    D-£B- 0
 0.0   |   |       0
 9.30  |  261
 0.18  |   D-£B- 5
 0.14  |   |    4
 0.0   |   E- 0
 0.0   |      £B- 0
 2.53  |-£K-71
 0.04  |      E-£B- 1
 0.0   |      D-£B- 0
```

the subtree G

```
30.77 G- E- 8
26.92  |  T- 7
23.08 L- T- 6
 3.85  |  E- 1
11.54 E- Q- 3
 3.85 N- E- 1
```


the subtree Z
1.53 £Q-£1-£Q-£1-£Q-£1-£Q-£1-£Q-£1-£Q-£1-£Q-£1-£Q-£1-£Q- 3
1.02 2
1.53 3
0.0 0
1.02 2
1.02 2
2.55 5
1.53 3
3.57 7
0.51 1
6.63 13
6.63 13
12.24 24
2.04 4
25.00 49
3.06 6
30.10 59
the subtree Y
70.97 /-44
19.35 /-12
4.84 /- /- /- /- 3
1.61 1
3323 2
the tree for assignment statements H
100.0 £V- =-£E-2174
the subtree 1
82.54 ,-293
0.85 £Y- 3
0.56 ,- 2
14.37 £Y-51
1.69 ,- 6
0.0 £Y- 0
the subtree E
70.59 £M-4402
19.29 £0-£M-1203
6.06 £0-£M-378
2.31 £0-£M-144
0.63 £0-£M-£0-£M-39
0.24 £0-£M-1
0.19 £0-1
0.69 43
the subtree B
50.00 ---£N-£N-26
3.85 2
28.85 +-£N-£N-15
0.0 0
9.62 £N- 5
7.69 £N- 4
the subtree M
98.79 £P-9045
1.19 --£P-109
0.02 +-£P- 2
the subtree P
40.40 £U-3699
24.74 (-£2-)-2265
30.64 £J-2805
4.23 (-£E-)-387
the subtree 0
34.38 *-1008
44.74 *-139
26.43 +-775
23.70 --695
10.74 /-315
the subtree 3
100.0 £E-3239
0.0 'H-£E- 0

the subtree 2

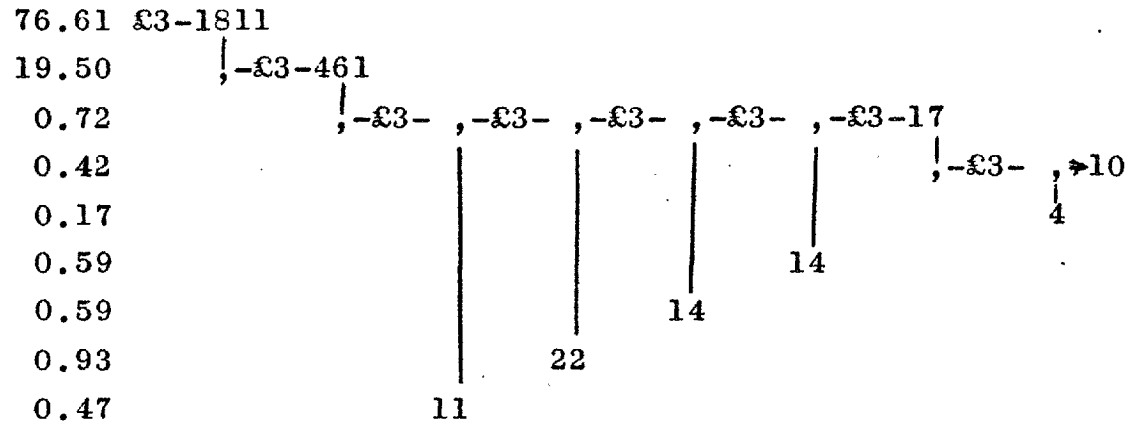
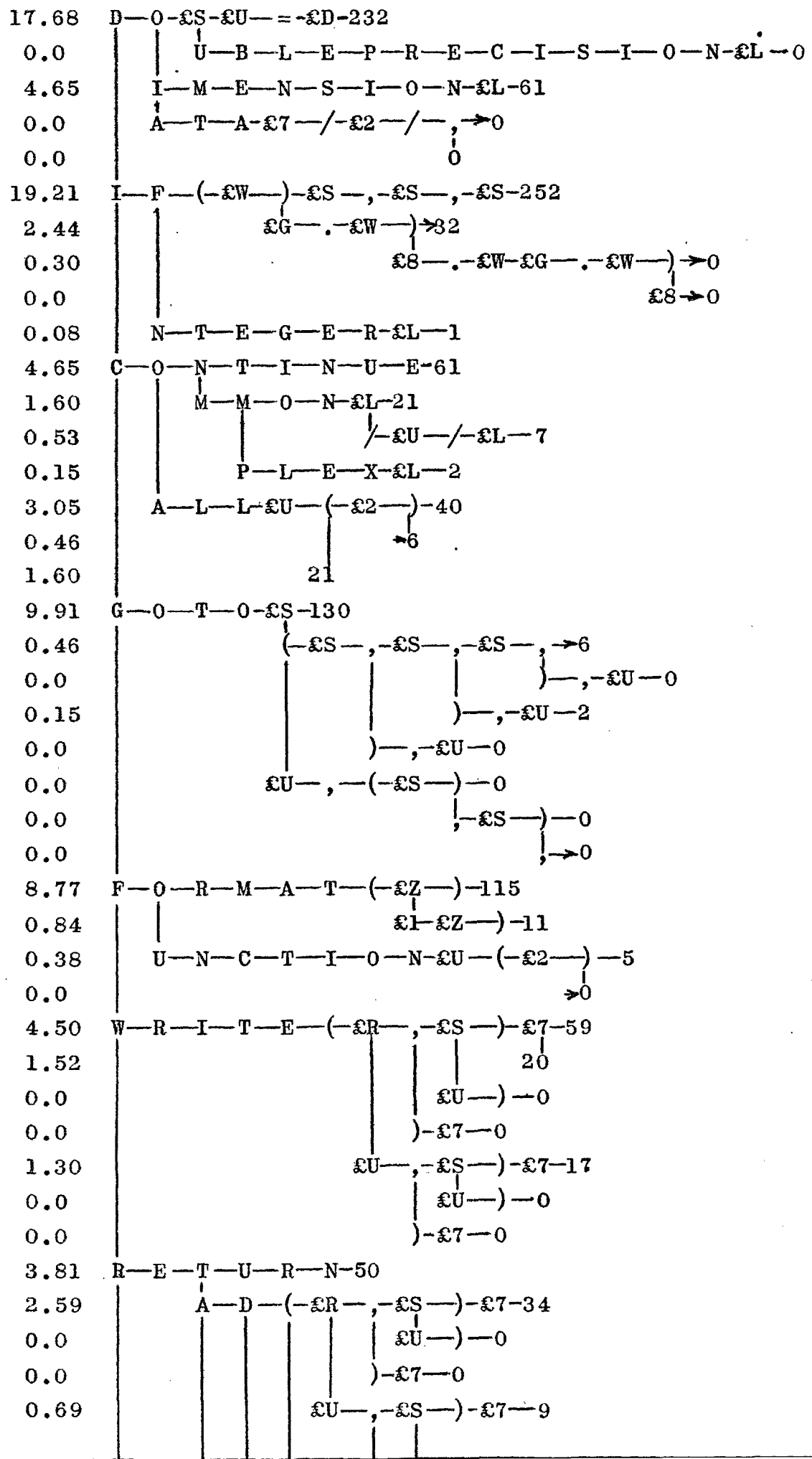
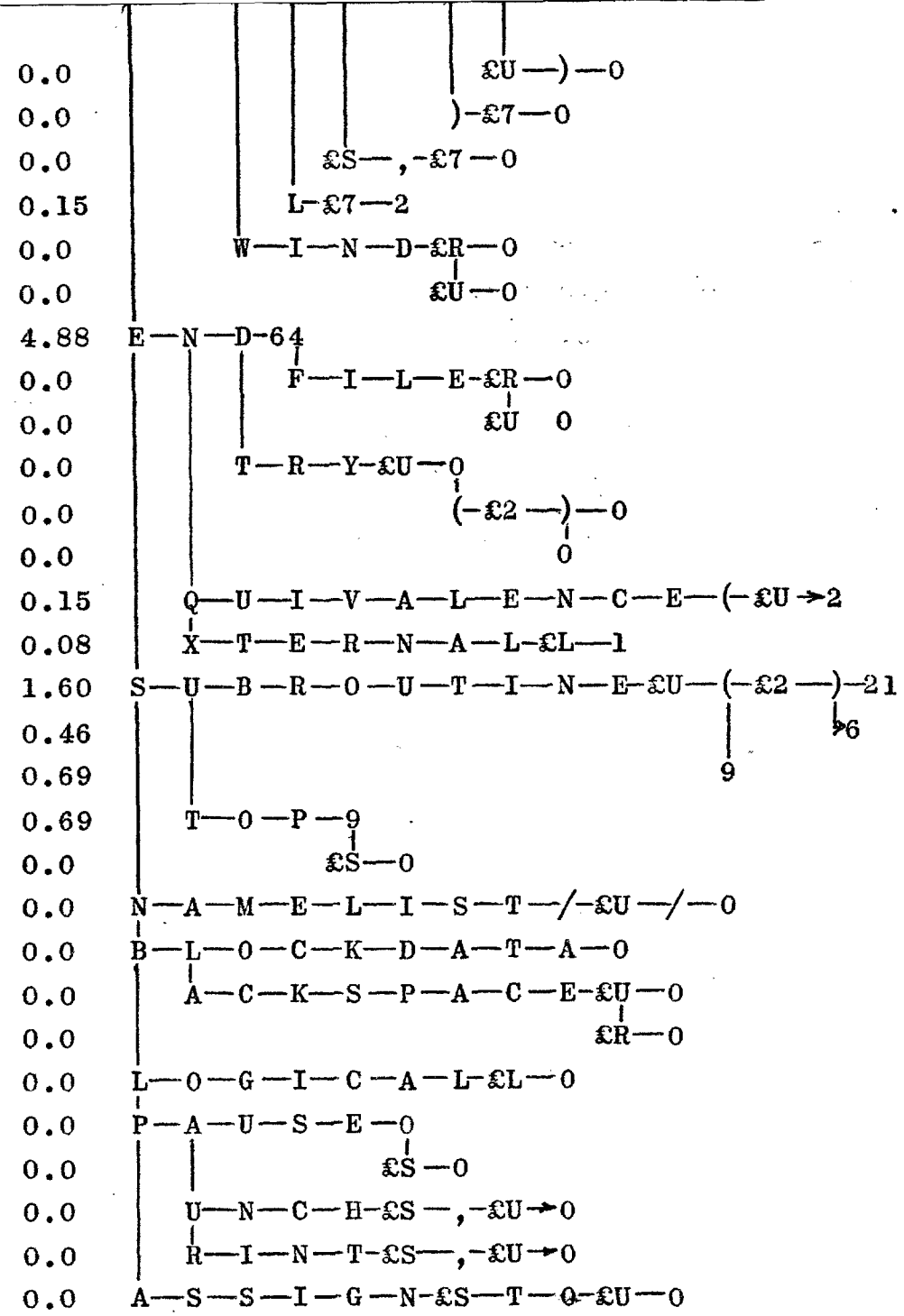


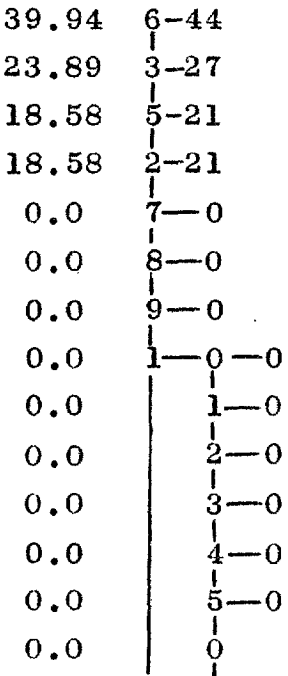
Fig. VI - the total experience tree after the analysis of Data Set IV, and optimisation

the main tree F

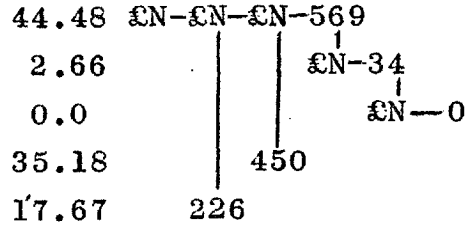




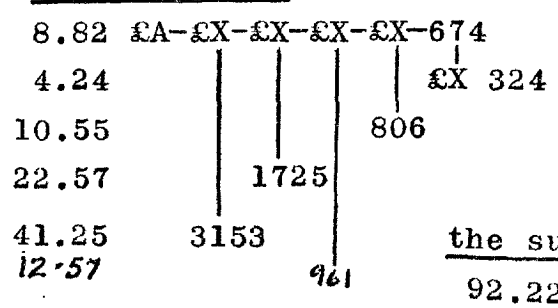
the subtree R



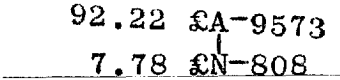
the subtree S



the subtree U



the subtree X



the subtree A

```
9.89   I-1702
8.48   A-1460
7.05   T-1213
6.59   S-1134
6.38   E-1099
5.63   M-970
5.26   N-906
4.57   R-787
4.26   K-734
3.98   B-686
3.79   L-652
3.60   P-619
3.42   O-588
3.25   C-559
3.03   X-522
2.82   D-486
2.72   H-468
2.20   Y-378
2.19   G-377
2.15   U-370
2.14   J-368
1.67   V-288
1.66   Q-286
1.49   F-257
1.03   W-178
0.75   Z-129
```

the subtree K

```
80.78 £N-2685
12.36    £N-411
 3.46       £N-116
 0.60          £N-£N-£N-£N-£N-£N-£N-20
 0.09                                 £N-3
 0.42                                    14
 0.36                                 12
 0.33                              11
 0.69                           23
 0.33                        11
 0.54                     18
```

the subtree L

```
10.56 £V-,-£V-,-£V-,-£V-,-£V-,-£V-,->15
 2.11                                  3
 4.93                               7
13.38                           19
 9.86                        14
33.80                     48
25.35           36
```

the subtree 5

```
90.70  (-£4-39
 9.30     (-£4-,-£4-4
 0.0         (-£4-,-£4-,-£4-0
                 etc. no occurrences of implie
```

DO loops nested to a depth of greater than two

the subtree N

```
25.86   1-2183
20.21   0-1706
13.00   2-1098
11.04   3-932
 7.90   4-667
 7.20   5-608
 4.19   7-354
 3.94   6-333
 3.80   9-321
 2.85   8-241
```

the subtree 4

```
100.0  £L-=-£D-)-47
```

the subtree C

```
46.23 £U-448
 7.33    +-£K-71
 0.21    --£K- 2
45.41 £K-440
 0.62     *-£U-  --£K-6
 0.21            2
 0.0             +-£K-0
```

the subtree D

```
53.41 £K- ,-£U-149
 2.51     |    ,-£K-7
 0.0      |      £U-0
36.92     £K-103
 1.43          ,-£K-4
 0.0             £U-0
 2.51 £U- ,-£U- 7
 1.79     |    ,-£K-5
 0.0      |      £U-0
 1.43     £K- 4
 0.0           ,-£U-0
 0.0             £K-0
```

the subtree 6

```
83.59 £V-219
16.41 £5-43
```

the subtree 8

```
75.00 A- N- D- 3
25.00 O- R- 1
```

the subtree V

```
62.30 £U-1403
                |
32.37      (-£C- )-729
                |
 5.33           |-£C- )-120
                |
                etc.
```

no occurrence of more than two subscripts

the subtree W

```
59.94 £E-199
              |
24.10      .-£E-80
           |  |  |
 0.30      |  |  .- 1
           |  |  |
 0.0       |  |  .- 0
           |  |  |
 0.0       |  |  £W- 0
           |  |
15.66      52
           |
 0.0       .- 0
```

the subtree 7

```
15.97 £6- ,-£6- ,-£6-19
          |     |     |
 9.24     |     |     |-£6-11
          |     |     |  |
 2.52     |     |     |  |-£6- ,-£6- 3
          |     |     |  |     |  |
 0.84     |     |     |  |     |  |-£6- ,-£6- 1
          |     |     |  |     |  |     |  |
 0.84     |     |     |  |     |  |     |  |-£6- 1
          |     |     |  |     |  |     |  |  |
 0.0      |     |     |  |     |  |     |  |  |-£6-(
          |     |     |  |     |  |     |  |
 0.0      |     |     |  |     |  |     0  |
          |     |     |  |     |  |        |
 2.52     |     |     |  |     3  |        |
          |     |     |  |        |        |
25.21     |     30    |  |        |        |
          |           |  |        |        |
42.86     51          |  |        |        |
```

the subtree J

```
66.19 £K-1067
          |
21.59     |  .-£K-348
          |  |
 0.43     |  |  E-£B- 7
          |  |  |
 0.0      |  |  0
          |  |
 0.0      |  |  D-£B- 0
          |  |  |
 0.0      |  |  0
          |  |
 8.56     |  138
          |  |
 0.0      |  D- 0
          |  |  |
 0.0      |  |  £B- 0
          |  |
 0.0      |  E- 0
          |     |
 0.0      |     £B- 0
          |
 3.23     .-£K-52
          |  |
 0.0      |  D-£B- 0
          |  |
 0.0      |  E-£B- 0
```

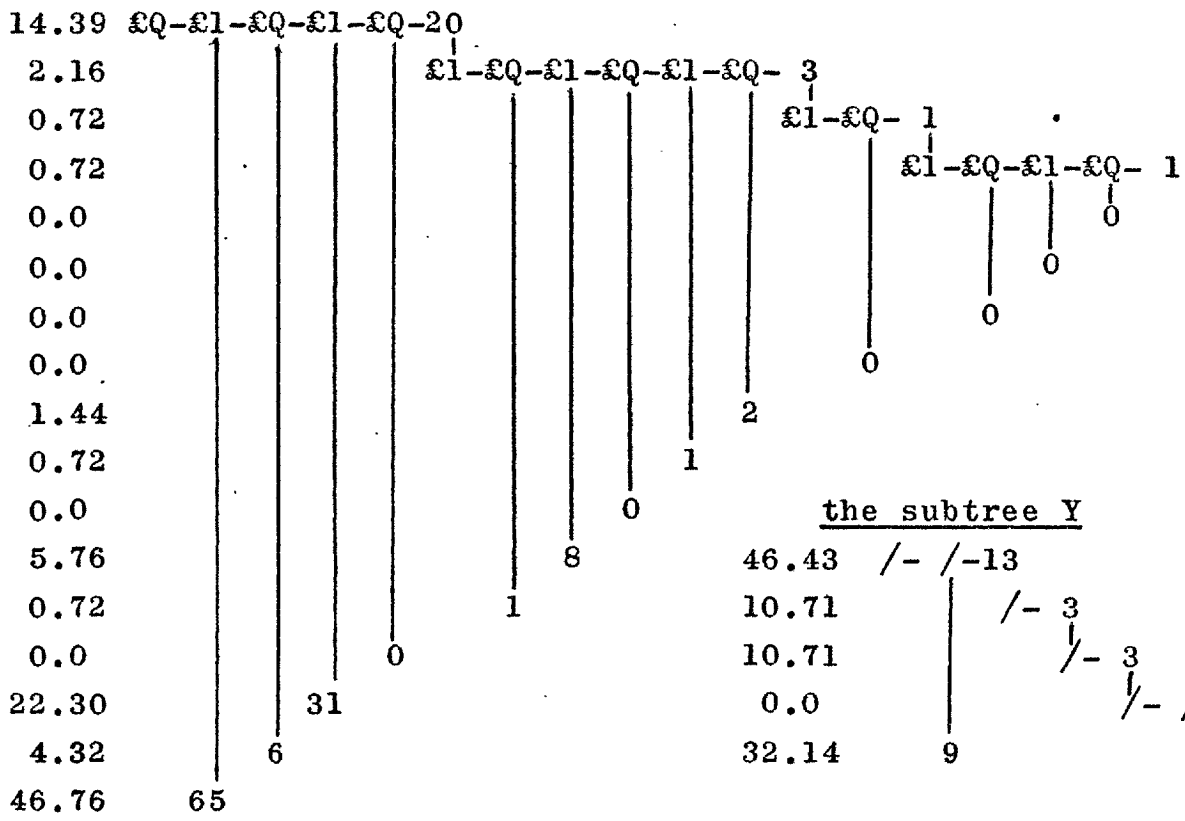
the subtree G

```
42.50  E- Q-17
       |  |
17.50  G- T- 7
       |  |
 5.00  |  E- 2
       |
15.00  L- E- 6
       |  |
 7.50  |  T- 3
       |
12.50  N- E- 5
```

the subtree Q

```
23.68 £K-'H-72
          |  |
 8.88     |  £Q-27
          |
 9.21     X-28
          |
 8.22     F-£K- .-£K-25
          |
 6.25     I-£K-19
          |
 3.62     E-£K- .-£K-11
          |
 2.96     (-£Z- )- 9
          |
 0.66     A-£K- 2
          |
 0.0      D-£K- .-£K- 0
          |
 0.0      0-£K- 0
          |
18.42  I-£K-56
       |
 9.54  F-£K- .-£K-29
       |
 5.92  E-£K- .-£K-18
       |
 1.32  A-£K- 4
       |
 1.32  (-£Z- )- 4
       |  |
 0.0   |  £1-£Z- )- 0
       |
 0.0   D-£K- .-£K- 0
       |
 0.0   0-£K- 0
       |
 0.0   +-£N- P-£K- F-£K- .-£K-0
       |           |
 0.0   |           E-£K- .-£K-0
       |           |
 0.0   |           D-£K- .-£K-0
       |
 0.0   --£N- P-£K- F-£K- .-£K-0
       |           |
 0.0   |           D-£K- .-£K-0
       |           |
 0.0   |           E-£K- .-£K-0
       |
 0.0   0-£K- 0
```

the subtree Z

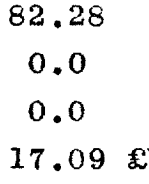


the subtree Y

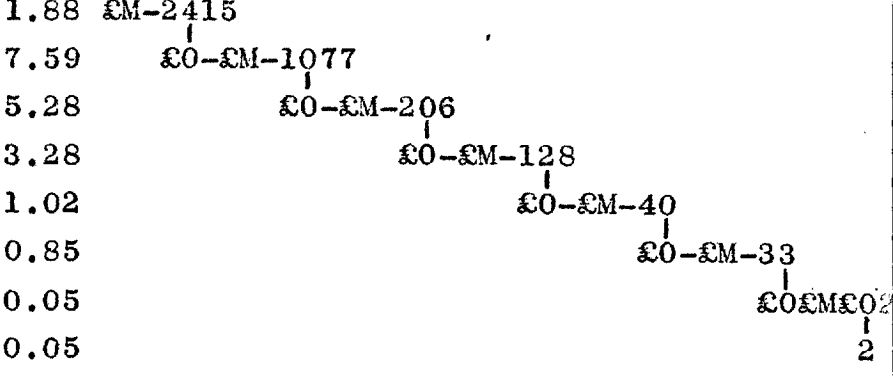
the tree for assignment statements H

100.0 £V- =-£E-1640

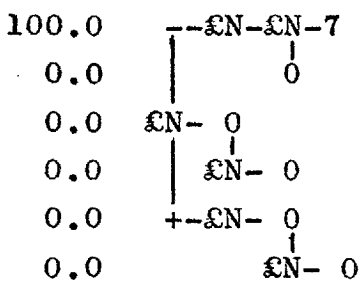
the subtree 1



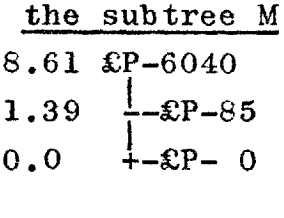
the subtree E



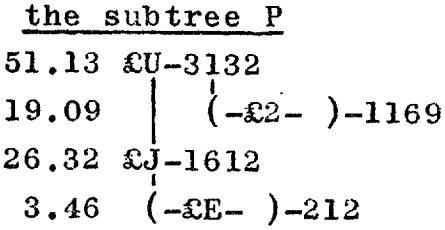
the subtree B



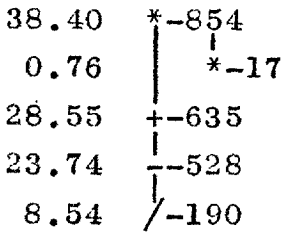
the subtree M



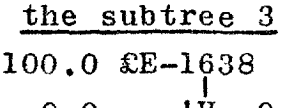
the subtree P



the subtree 0



the subtree 3



the subtree 2

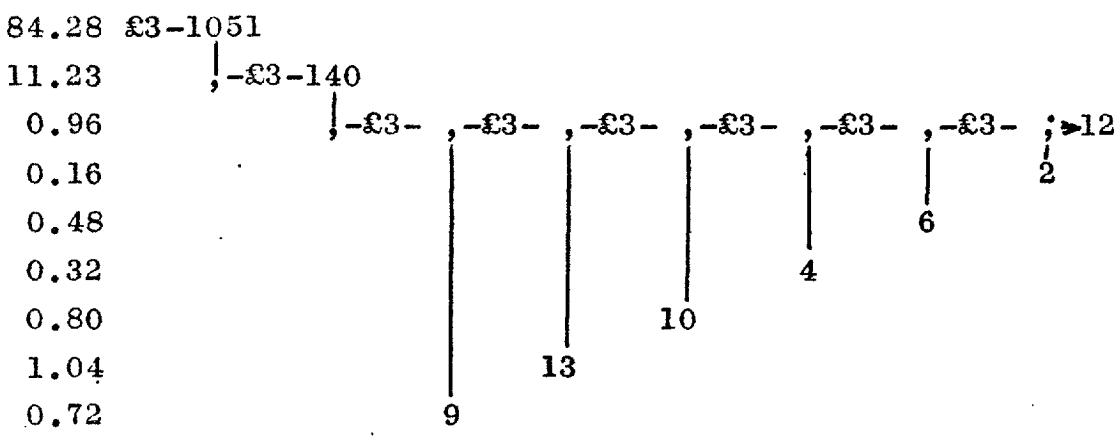
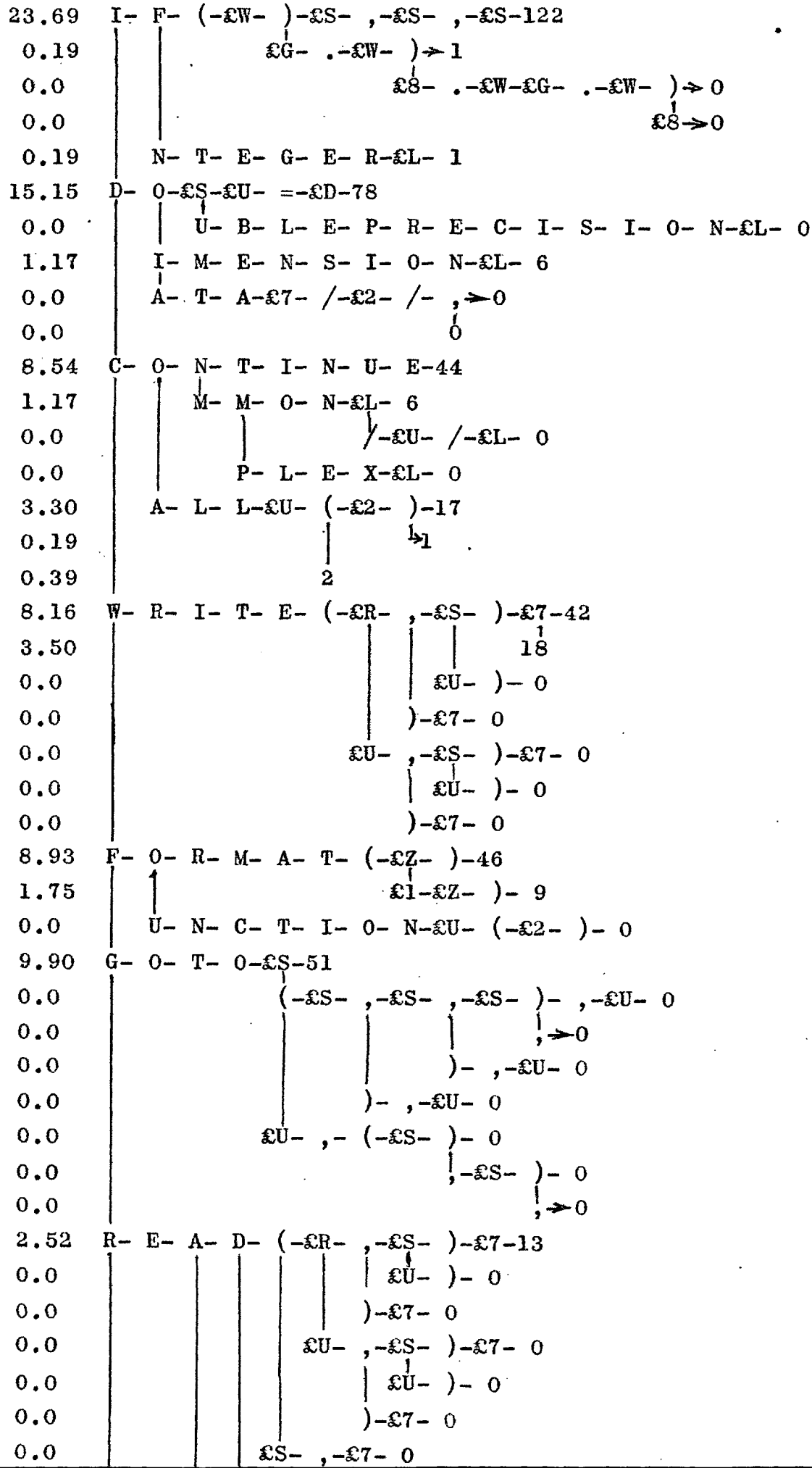


Fig. VII - the total experience tree after the analysis of Data Set V, and optimisation

the main tree F



```
0.58                L-£L- 3
1.55              T- U- R- N- 8
0.0               W- I- N- D-£R- 0
0.0                          £U- 0
3.69  E- N- D-19
0.19          F- I- L- E-£R- 1
0.0                       £U- 1
0.0           T- R- Y-£U- 0
0.0                     (-£2- )- 0
0.0       X- T- E- R- N- A- L-£L- 0
0.0       Q- U- I- V- A- L- E- N- C- E- (-£U-> 0
1.94  S- T- O- P-10
0.39          £S- 2
0.78      U- B- R- O- U- T- I- N- E-£U- (-£2- )- 4
0.19                                       -1
0.19                                    1
1.75  P- A- U- S- E-£S- 9
0.0               0
0.0       U- N- C- H-£S- ,-£U-> 0
0.0       R- I- N- T-£S- ,-£U-> 0
0.0   B- L- O- C- K- D- A- T- A- 0
0.0       A- C- K- S- P- A- C- E-£U- 0
0.0                                £R- 0
0.0   L- O- G- I- C- A- L-£L- 0
0.0   N- A- M- E- L- I- S- T- /-£U- /- 0
0.0   A- S- S- I- G- N-£S- T- O-£U- 0
```

the subtree R

```
58.11 1-43
0.0     1- 0
0.0     2- 0
0.0     3- 0
0.0     4- 0
0.0     5- 0
0.0     0- 0
0.0     9- 0
41.89 2-31
0.0   6- 0
0.0   4- 0
0.0   7- 0
0.0   8- 0
0.0   9- 0
0.0   3- 0
0.0   5- 0
```

the subtree S

```
32.99 £N-£N-£N-191
 5.53          £N-32
 0.0                £N- 0
30.05       174
31.43   182
```

the subtree U

```
 6.18 £A-£X-£X-£X-£X-148
 3.34                   £X-80
 6.56              157
11.65         279
25,27      605
46.99   1125
```

the subtree X

```
86.48 £A-2271
13.52 £N-355
```

the subtree A

```
11.45 I-534
 6.54 A-305
 6.54 N-305
 6.41 E-299
 6.22 D-290
 5.40 M-252
 5.27 S-246
 4.59 T-214
 4.57 P-213
 4.29 L-200
 3.92 J-183
 3.75 K-175
 3.58 X-167
 3.13 F-146
 2.92 R-136
 2.89 O-135
 2.89 B-135
 2.83 H-132
 2.55 C-119
 2.34 U-109
 1.89 Z-88
 1.78 Q-83
 1.52 V-71
 1.22 G-57
 1.18 Y-55
 0.34 W-16
```

the subtree K

```
84.47 £N-756
10.61    £N-95
 2.68       £N-24
 0.89          £N-£N-£N-£N-£N-£N- 8
 0.11                              £N-£N-1
 0.0                                     0
 0.11                             1
 0.11                        1
 0.11                   1
 0.67              6
 0.22         2
```

the subtree L

```
20.00 £V- ,-£V- ,-£V- ,-£V- ,-£V- ,-£V- 4
10.00                                  ,→2
 5.00                                1
 5.00                           1
 5.00                      1
35.00                 7
20.00            4
```

the subtree 5

```
100.00 (-£4- 4
          (-£4- ,-£4- 0
```

etc. no occurrence of implied DO loops nested to a depth of greater than 1

the subtree N

```
29.78 1-819
20.33 0-559
17.09 2-470
 6.98 3-192
 6.40 5-176
 4.62 4-127
 4.33 9-119
 3.71 7-102
 3.67 8-101
 3.09 6-85
```

the subtree 4

```
100.0 £L- =-£D- )-4
```

the subtree C

```
59.34 £U-162
 6.59       +-£K-18
 0.37       --£K- 1
33.70 £K-92
 0.0          *-£U- 0
 0.0                --£K-0
 0.0                +-£K-0
```

the subtree D

```
41.46 £K- ,-£U-34
 0.0      |    ,-£K-0
 0.0      |      £U-0
39.02     £K-32
 0.0           ,-£K-0
 0.0             £U-0
19.51 £U- ,-£U-16
 0.0      |    ,-£U-0
 0.0      |      £K-0
 0.0      £K- 0
 0.0           ,-£U-0
 0.0             £K-0
```

the subtree 6

```
97.58 £V-161
 2.42 £5- 4
```

the subtree 8

```
0.0 0- R- 0
0.0 A- N- D- 0
```

the subtree V

```
70.96 £U-518
20.68      (-£C- )-151
 8.36            ,-£C- )-61
 0.0                   ,-£C- )-0
                                etc.
```

no occurrence of variables with more than two subscripts.

the subtree W

```
95.97 £E-119
 3.23      .-£E- 4
 0.0             .- 0
 0.0               .- 0
 0.0                £W- 0
 0.81       1
 0.0        .- 0
```

the subtree 7

```
 3.64 £6- ,-£6- ,-£6- ,-£6- ,-£6- ,-£6- ,-£6- ,-£6- ,-£6-2
 1.82                                                ,-£6-1
 1.82                                          1
 5.45                                    3
 0.0                               0
 5.45                        3
 7.27                  4
21.82            12
21.82       12
30.91    17
```

the subtree J

```
67.06 £K-281
15.99    .-67
12.41       £K-52
 0.72          E-£B- 3
 0.0             0
 0.0           D-£B- 0
 0.0             0
 0.0         D- 0
 0.0           £B- 0
 0.0         E- 0
 0.0           £B- 0
 3.82  .-£K-16
 0.0         D-£B- 0
 0.0         E-£B- 0
```

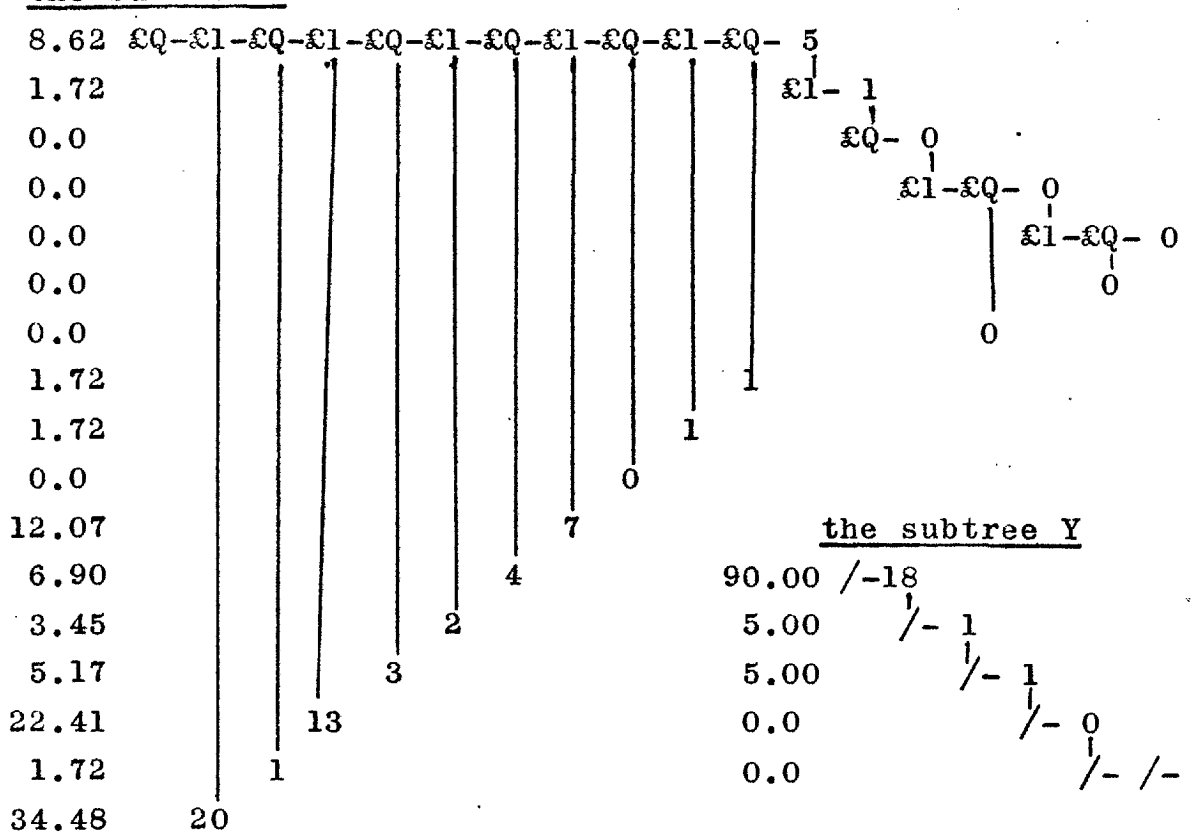
the subtree G

```
100.0 G- E- 1
  0.0   T- 0
  0.0 E- Q- 0
  0.0 L- T- 0
  0.0   E- 0
  0.0 N- E- 0
```

the subtree Q

```
19.73 £K-'H-29
 1.36      £Q- 2
17.61    X-25
 7.48      I-£K-11
 4.76        F-£K- .-£K- 7
 4.08        E-£K- .-£K- 6
 2.04        (-£Z- )- 3
 1.36        A-£K- 2
 0.0         D-£K- .-£K- 0
 0.0         0-£K- 0
23.81  I-£K-35
13.61  F-£K- .-£K-20
 4.08  E-£K- .-£K- 6
 0.68  A-£K- 1
 0.0   (-£Z- )- 0
 0.0     £1-£Z- )- 0
 0.0   D-£K- .-£K- 0
 0.0   0-£K- 0
 0.0   +-£N- P-£K- F-£K- .-£K- 0
 0.0                 E-£K- .-£K- 0
 0.0                 D-£K- .-£K- 0
 0.0   --£N- P-£K- F-£K- .-£K- 0
 0.0                 D-£K- .-£K- 0
 0.0                 E-£K- .-£K- 0
 0.0   L-£K- 0
```

the subtree Z

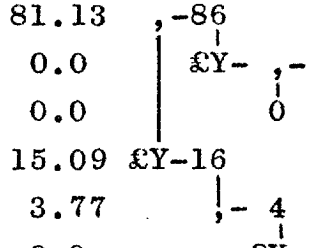


the subtree Y

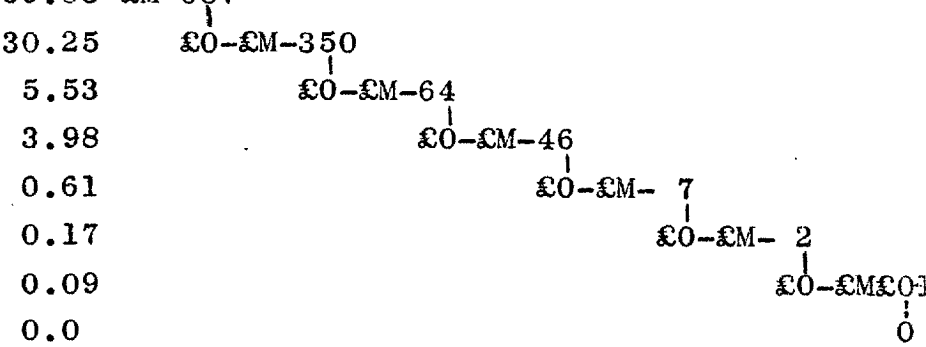
the tree for assignment statements H

100.0 £V- =-£E-503

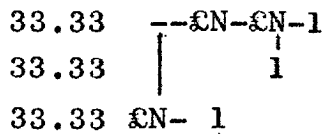
the subtree 1



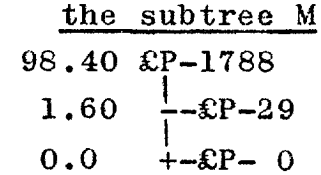
the subtree E



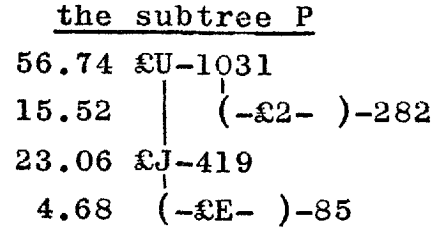
the subtree B



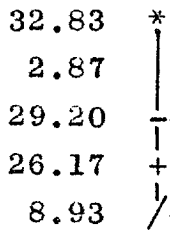
the subtree M



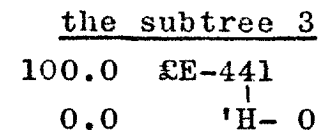
the subtree P



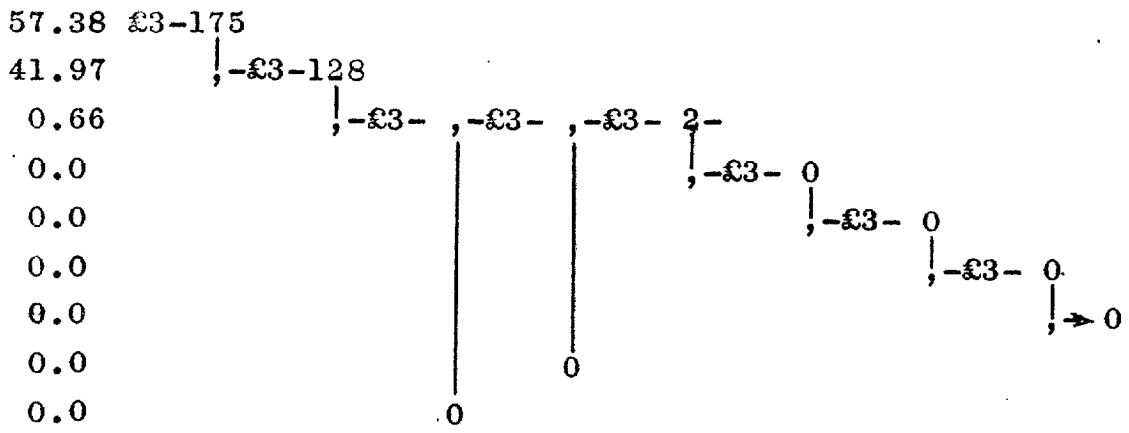
the subtree 0



the subtree 3



the subtree 2



APPENDIX IX

Fig. Ia - Data Set I.

Errors in the Imperial College typical student jobs,1969

10,690 cards or 8,164 Fortran statements,using System I

and 'random' strategy tree

NOGIRLS=0 time trapped

FORMAT(1XI10) in7.7 seconds FORMAT(19I10)

FORMAT(I1,4X,I5,5X,F5.3,5X,F5.2,5XI1) time trapped

FORMAT(1X,I6,7H STONES,2XI2,7H POUNDS)

in 15.8 secs. FORMAT(1X,I6,7H STONES,29I2,etc.

a longer version of above time trapped

IF(LASTCD=1)GO TO 101 time trapped

FORMAT(1X,3F10.0) not zero,time trapped

WRITE(0,2001) L3,N3 matched in 7.72 secs,syntactically correc

READ(5,1000)ISA_ISB_ISC IN 10.2 secs. READ(5,1000)ISAISB,SC

above 3 more times,total of 6 more commas missing

WRITE(6,1010)ISA_AGEA_HIGHA_IWAYA in 19.6 secs.

WRITE(6,1010)ISAAG,AHIGHA,WAYA

above 2 more times,total of 6 more commas missing

ALB=(WA - IWAYA)14 failed in 13.3 seconds

WRITE(10,3000) AVAGE matched in 9.1 secs.,syntactically correct

WRITE(11,3010)AVHIGH " " 11.1 " " "

above with more incorrect unit numbers,4 more times

WRITE(6,13)AVWEIGH 11.4 secs. WRITE(6,13)AVWEI,H

FORMAT(1X,FK.2) time trapped

QX(I,J_=RX(I,J) " "

READ(5,55)(IMAX,JMAX) " "

Fig. Ib - Data Set I

Error summary - Imperial College typical student jobs,1969

26 statements of 8,164 were found to be incorrect, that is 0.32%

1 character wrong 10 times
1 character inserted 8 times
1 character missing 7 times
backtrack 1 character and 1 character missing 7 times
backtrack 2 characters and 1 character missing 5 times
backtrack 3 characters and 1 character missing 5 times
3 characters missing, once
backtrack 10 characters and 1 character inserted, once

Average 1.7 errors per statement, although several were single compound errors which we have divided into two in the manner which our strategy tree would perform the correction.

7 of the statements were syntactically correct, but from the job context were obviously in error and became apparent in our experience tree by the appearence of very low frequency counts.

10 of the remaining syntactically incorrect statements were recovered - that is 53%, none 'correctly'

Fig. IIa – Data Set II

Errors in Imperial College current PUFFT batches,-novice programmers

61 jobs comprising 6,326 cards or 3958 Fortran statements,

EDASH=_1.+(2.*E-E**3/4.)*COS(M)*E**4/6.) time trapped

READ(5,10,END=1,ERR=2) time trapped

similar to above again

FORMAT(X,A4,X,A6,etc six integers missing,time trapped

FORMAT(1H1 27X13A6/) in 8.6 seconds FORMAT(1H127X,3A6/)

FORMAT(43XA5,7A6) 14.7 seconds FORMAT(43X,5X,7A6)

FORMAT(3(10A6/)_10A6) in 10.4 secs. FORMAT(3(10A6/),0A6)

READ(5,2)(CARD(POINT),POINT=1,80_ time trapped

long FORMAT with end ")" missing time trapped

FORMAT with a Hollerith spec. too short,an "E" left out in 20 secs, modified to E specification

FORMAT(11X_3HAT 05,2H, 012) in 14.6 secs.

FORMAT(11X,3HAT 05,2H, 012)

FORMAT(I5,X,I3) in 7.6 secs. FORMAT(I5,/,I3)

FORMAT with three "," missing following X specifications

14 more FORMAT statements with single X specifications and no "," following,all time trapped

WRITE(6,2000)NUMBER, _((PIC(L,M),M=1,20),L=1,50) time trapped

FORMAT(1H*I12,1H,I11,X101A1) in 22 secs. modified to

FORMAT(1H*I12,1H,I11,1X,01A1)

FORMAT(16H-FILE LENGTH = I2/(5X_20A1)) in 12.7 seconds

FORMAT(16H-FILE LENGTH = I2/(5X,0A1))

FORMAT(X22A6) in 8.2 seconds FORMAT(1X,2A6)

FORMAT(2H**,X,43HZER etc. in 12.8 secs FORMAT(2H**,/,43HZER etc

DO 20 K=_,NA time trapped

NOLIST in 9.7 seconds became ENDFILE ST

another FORMAT with single X specification time trapped

IFINISH=0 in 6.1 seconds modified to IFINIS=0

FORMAT(1H0,5X,F4.2,19X,E20.8/ time trapped

Fig. IIb - Data Set II

Error Summary - Imperial College novice programmers,1972

If we discount the 25 FORMAT statements containing missing integers and commas,before and after the specification X, respectively,as the explanation that they represent the programmers deliberate exploitation of a compiler relaxation seems the most likely.Then 12 statements were found to contain syntactic mistakes,that is 0.3% of statements or 1 incorrect statement per five jobs.

1 character omitted	6 times
1 character inserted	twice
1 character wrong	once

plus:one statement which appears to have been an incorrect job control card,

two statement that were entirely wrong,

and one job in which a section of ten to twenty cards were read backwards or upside-down.

Four of the 12 incorrect statements were recovered,and many of the 25 that we have discounted from the error summary.Even the job control statement NOLIST was interpreted as the Fortran statement ENDFILE ST,which demonstrates the persistence of the system.

Fig. IIIa - Data Set III

Errors in the Oxford University benchmark - typical student jobs.

19 jobs,comprising 5008 cards or 4410 Fortran statemerts

FORMAT(I10,I3,10I4,2I2,21XI2) 22.25 ...2I2,21X,2)

FORMAT(Hollerith TE10.3,3H + ,E10.3 .. time trapped 20.5

I=(0.0,1.0) 8.5 I=(0.0/1.0) complex const not on tree

above again

FORMAT(1H CO=,F5.3 .. time trapped 20.5

DO14I=1,M dealt with as arith by mistake 5.5 DO14I=1+M

above again five times

FORMAT(1H ,I3,1PE13.3) 14.5 ...I3,11E13.3 1P not on tree

above again

DO Loop problem again

60RM1T (I3 3F9.5) failed lack of space 9.0

4=450 system error

96(1.LT.0)KPLUS=2 8.4 IF(1.LT.0)KPLUS=2

WRITE(3,540)((CLUST(I) ...OMEGA(I)),I=1,N) lack of space

GO TO (11,12) M 6.0 GOTO(11,12),M

four more times

FORMAT(Hollerith too long time trapped

above again 3 times

DO Loop problem again

WRITE(60,100)(K(J),J=1,M) 11.8 WRITE(6,100) etc not on tree R

Fig. IIIb - Data Set III

Error summary - Oxford University benchmark, typical student jobs.

15 statements of 4323 were found to be incorrect, that is 0.35 %, but nearly one incorrect statement per job.

1 character wrong 9 times

1 character omitted 7 times

1 character inserted twice

2 characters wrong once

Average of 1.27 isolated errors per statement

6 statements were successfully recovered - all 'correctly'

e.g.

96(I.LT.0) KPLUS=2 in 8.4 seconds IF(I.LT.0)KPLUS=2

GO TO (11,12) M in 6.0 seconds GOTO(11,12),M

In addition, 39 more statements were interpreted by the system as incorrect because the necessary features had been omitted from the syntax trees. Although we cannot include these statements within the error statistics they do never the less illustrate system's recovery ability as 13 were recovered.

e.g.

I=(0.0,1.0) in 8.5 seconds I=(0.0/1.0)

FORMAT(1H ,I3,1PE13.3) 14.5 seconds FORMAT(1H ,I3,11E13.3)

WRITE(60,100)(K(J),J=1,M) 11.8 secs. WRITE(6,100)(etc.

and the DO statement DO 14 I=1,M was treated as an arithmetic statement due to a fault in the algorithm,

in 5.5 seconds DO14I=1+M

Fig. IVa - Data Set IV

Errors in CERN jobs;

19 jobs, comprising 3374 cards or 3020 Fortran statements, 2% were not successfully analysed due mainly to omissions in the tree items and also the unsatisfactory performance of the trees dealing with logical operators.

LIST(LP) systems card that was not removed

WRITE(NOUT,50) necessary item not on tree

above 16 more times

FORMAT(/) and FORMAT(//) not on tree

14 assignment statements to 16 decimal places

FORMAT(8HOR.M.S. 1PE15.7//) modified in 9.8 seconds

FORMAT(8HOR.M.S. 11E15.7//)

above modified again, and another failed lack of time

6 statements such as IF(A -1.E-20) etc. interpreted as incorrect

6 arithmetic expressions too long for satisfactory analysis

1 implied DO LOOP variable list too long

1 FORMAT statement with too many elements, 13

READ(5,21) variable list missing, time trapped

above again

FORMAT(1X6F5.2 etc, five times with total of 22 commas missing!

4 DO LOOP statements treated as arithmetic and modified

Six of the above 'errors' were modified to a syntactically compatible form, that is 10%

Fig. IVb - Data Set IV

Error summary - CERN jobs

7 statements appeared to be incorrect, which is 0.23%, of the 3020 statements, or approximately one error per three jobs.

24 times one character missing

The occurrence of one comma missing after the FORMAT specification nX (e.g. 2X), 22 times in only five statements leads us to the possibility that this was not an 'error'; but rather the programmers exploitation of a non-standard feature of the compiler used.

All of the above 7 statements were time trapped after 20 seconds owing to the complexity of the initial statements and the modification required.

Fig. Va - Data Set V

Errors in the ICL programs - commercial programs

1126 cards, comprising 1055 Fortran statements or 12 jobs.

CALL THRUSTF(VO,ZO) modified in 9.4 seccs CALL THRUST(VO,ZO)

above again three more times

SUBROUTINE THRUSTF(VO,ZO) time trapped

LINECOUNT=50 in 20.6 seconds LINECO=NT/50

IF statement with above variable failed

LINECOUNT=LINECOUNT-2 in 15 secs. LINECO=NTLIN+COUNT-2

DO 17 I=1,NODES+1 in 16.1 DO17I=1,NODES1

above again

CALL USERSUB in 17.5 seconds CALL USERSU(UIB)

again with SUBROUTINE in 16.7 seconds

DO 8 IYPLUS1=1,MAX time trapped

IY=IYPLUS1-1 in 8.9 seconds IY=IYPLUS/-1

WRITE(1,21), IR,IS,IRSQ,KLESS1,KSQ,LCD,IY failed

HIST(21),HIST(34),HIST(67)=1 failed

MTEST,IP(1)=5 in 8.2 seconds MTEST=IP(1)/5

KEND,N=1 and another similar, time trapped

DO 9 K=N+2,NQ in 12.9 seconds DO9K=N2,NQ

DO 16 J=1,NQ-K in 6.2 seconds DO16J=1,NQIK

U(N+J)=U(K+J) in 8.0 seconds U(N+1)=U(K+J)

as above with L instead of U

two more DO statements with arithmetic expression in bounds modified

U(NQ+2-J)=U(NQ-J+1) in 13.3 seconds U(NQ+2,J)=U(NQ-J+1)

above three more times, one modified and two time trapped

Fig. Vb - Data Set V

Error summary - ICL jobs

Again we encounter the problem of deciding if certain statements are incorrect or exploitation of non-standard features.The ICL programs contain variables with greater than six characters,multiple assignments and DO statements with arithmetic expressions as the bounds.The interpretation that these statements are grossly incorrect seems to be the less likely alternative.

Thus we are left with only one incorrect statement.The error was,1 character inserted.

Nevertheless the recovery ability of our system is demonstrated by the fact that 20 of the 29 syntactically incompatible statements were recovered,that is nearly 70%. Four of the recoveries were the 'best' modifications (these were the truncation of seven letter routine names to six letters),virtually all the remaining corrections required the insertion of at least one extra statement.

For example:

```
DO 9 K=N+2,NQ   should become      N2=N+2
                                   DO 9 K=N2,NQ
```

APPENDIX IX — Fig. VI - specimen listing of system output

EXECUTION

tree F

value of frequency from associated terminal node

pattern composed of immediate symbols and numbers of parameters to be output

subtree reference nodes identified by associated parameter number as a subscript

```
0   COMMOONL1tCOMMON1
0   COMMOON/U1/L2tCOMMON/1/2
0   COMPOLEXL1tCOMPLEX1
0   CONOTINUEtCONTINUE
0   CAILLU1(2)tCALL1(2)
0   CAILLU1(2tCALL1(2
0   CAILLU1tCALL1
0   IF(W1G2,W3)→4tIF(1?,3)4
0   IF(WG,W8,WG,W7)→3tIF(12,34,56,7)8
0   IF(WG.W8.WG.W8→3tIF(12.34.56.789
0   IF(W1)S2,S3,S4tIF(1)2,3,4
0   INOTEGERL1tINTEGER1
0   DOS1U2=D3tDO12=3
0   DOUOBLEPRECISIONL1tDOUBLEPRECISION1
0   DI1MENSIONL1tDIMENSION1
0   DAOTA71/2-/,→3tDATA1/2/,3
0   DAOTA71/22/tDATA1/2/
0   DAOTA71/22tDATA1/2
0   DAOTA7tDATA1
0   WORITE(R1,S2)73tWRITE(1,2)3
0   WORITE(R1,S2)tWRITE(1,2)
0   WORITE(R1,U2)tWRITE(1,2)
0   WORITE(R1)72tWRITE(1)2
0   WORITE(U1,S2)73tWRITE(1,2)3
0   WORITE(U1,U2)tWRITE(1,2)
0   WORITE(U1)72tWRITE(1)2
0   FOORMAT(71)tFORMAT(1)
0   FOORMAT(1,72)tFORMAT(12)
0   FUINCTIONU1(2)tFUNCTION1(2)
0   FUINCTIONU1(22tFUNCTION1(2
0   GIOTOS1tGOTO1
0   GIOTO(S1,S2,S3),U4tGOTO(1,2,3),4
0   GIOTO(S1,S2,S3,→4tGOTO(1,2,3,4
0   GIOTO(S1,S2),U3tGOTO(1,2),3
0   GIOTO(S1),U2tGOTO(1),2
0   GIOTOU1o,(S2)tGOTO1,(2)
0   GIOTOU1o,(S2,S3)tGOTO1,(2,3)
0   GIOTOU1o,(S2,S3,→4tGOTO1,(2,3,4
0   RETOURNtRETURN
0   READ(R1,S2)73tREAD(1,2)3
0   READ(R1,U2)tREAD(1,2)
0   READ(R1)72tREAD(1)2
0   READ(U1,S2)73tREAD(1,2)3
0   READ(U1,U2)tREAD(1,2)
0   READ(U1)72tREAD(1)2
0   READS1,72tREAD1,2
0   REALL1tREAL1
0   REWOINOR1tREWIND1
0   REWOINDU1tREWIND1
0   ENDtEND
```

a jump node and associated jump path

in the interests of clarity the jump path is omitted from succeeding instances of the same node

```
0    ENDFOILER,tENDFILE1
0    ENDFOILEU,tENDFILE1
0    ENTORYU,tENTRY1
0    ENTORYU,(2,)tENTRY1(2)
0    ENTORYU,(2,tENTRY1(2
0    EXOTERNALL,tEXTERNAL1
0    EQOUIVALENCE(U,=2,tEQUIVALENCE(12
0    STOOPtSTOP
0    STOOPS,tSTOP1
0    SUOBROUTINEU,(2,)tSUBROUTINE1(2)
0    SUOBROUTINEU,(2,tSUBROUTINE1(2
0    SUOBROUTINEU,tSUBROUTINE1
0    NOAMELIST/U,/=2,tNAMELIST/1,/2
0    BLOCKDATAtBLOCKDATA
0    BAOCKSPACEU,tBACKSPACE1
0    BAOCKSPACER,tBACKSPACE1
0    LOOGICALL,tLOGICAL1
0    PAOUSEtPAUSE
0    PAOUSES,tPAUSE1
0    PUONCHS,,U,=3,tPUNCH1,23
0    PROINTS,,U,=3,tPRINT1,23
0    AOSSIGNS,OTOU,tASSIGN1TO2
```

subtree R

```
0    6
0    5
0    10
0    11
0    12
0    13
0    14
0    15
0    1
0    19
0    4
0    7
0    8
0    9
0    3
0    2
```

terminal nodes omitted from all subtree items

subtree S

```
0    NNN
0    NNNN
0    NNNNN
0    NN
0    N
```

subtree U

```
0    AXXXX
0    AXXXXX
0    AXXX
0    AXX
0    AX
0    A
```

subtree A

```
0    I
0    A
0    N
```

```
0    D
0    C
0    S
0    R
0    T
0    P
0    L
0    F
0    B
0    M
0    G
0    X
0    K
0    D
0    F
0    Y
0    W
0    H
0    J
0    U
0    V
0    Z
0    Q

X
0    A
0    N

N
0    1
0    0
0    2
0    3
0    5
0    4
0    8
0    9
0    6
0    7
```

tree T — The Strategy Tree

```
0   BACKSPACE   0 CHARACTERS,AND SEARCH ASSUMING   0 CHARACTERS OMITTED WHILST FREQUENCY NOT LESS THAN      1
0   BACKSPACE   0 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS WRONG WHILST FREQUENCY NOT LESS THAN      1
0   BACKSPACE   0 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS INSERTED WHILST FREQUENCY NOT LESS THAN  100
0   BACKSPACE   1 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS OMITTED WHILST FREQUENCY NOT LESS THAN  125
0   BACKSPACE   0 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS OMITTED WHILST FREQUENCY NOT LESS THAN  143
0   BACKSPACE   2 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS OMITTED WHILST FREQUENCY NOT LESS THAN  165
0   BACKSPACE   3 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS OMITTED WHILST FREQUENCY NOT LESS THAN  200
0   BACKSPACE   0 CHARACTERS,AND SEARCH ASSUMING   3 CHARACTERS OMITTED WHILST FREQUENCY NOT LESS THAN  200
0   BACKSPACE   0 CHARACTERS,AND SEARCH ASSUMING   2 CHARACTERS INSERTED WHILST FREQUENCY NOT LESS THAN 1000
0   BACKSPACE   0 CHARACTERS,AND SEARCH ASSUMING   2 CHARACTERS WRONG WHILST FREQUENCY NOT LESS THAN      1
0   BACKSPACE   0 CHARACTERS,AND SEARCH ASSUMING   2 CHARACTERS OMITTED WHILST FREQUENCY NOT LESS THAN    0
0   BACKSPACE   1 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS WRONG WHILST FREQUENCY NOT LESS THAN      0
0   BACKSPACE   1 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS INSERTED WHILST FREQUENCY NOT LESS THAN   0
0   BACKSPACE   2 CHARACTERS,AND SEARCH ASSUMING   2 CHARACTERS WRONG WHILST FREQUENCY NOT LESS THAN      0
```

```
K
0    N
0    NN
0    NNN
```

```
                0       NNNNNNNNNNN
                0       NNNNNNNNNN
                0       NNNNNNNNN
                0       NNNNNNNN
                0       NNNNNNN
                0       NNNNNN
                0       NNNNN
                0       NNNN

        V
                0       U
                0       U(C)
                0       U(C,C)
                0       U(C,C,C,C)
                0       U(C,C,C,C,C)
                0       U(C,C,C,C,C,C)
                0       U(C,C,C,C,C,C,C)
                0       U(C,C,C)

        L
                0       V
                0       V,V
                0       V,V,V
                0       V,V,V,V,V,V,V
                0       V,V,V,V,V,V
                0       V,V,V,V,V
                0       V,V,V,V

        C
                0       U
                0       U+K
                0       U-K
                0       K
                0       K*U
                0       K*U-K
                0       K*U+K

        H       - The main tree for assignment statements
                0       V=E#1=2

        8
                0       OR
                0       AND

        D
                0       K,U
                0       K,U,K
                0       K,U,U
                0       K,K
                0       K,K,K
                0       K,K,U
                0       U,U
                0       U,U,U
                0       U,U,K
                0       U,K
                0       U,K,U
                0       U,K,K

        .4
                0       L=D)
```

```
5
        0       (4
        0       ((4,4
        0       (((4,4,4
        0       ((((4,4,4,4

6
        0       V
        0       5

7
        0       6,6,6
        0       6,6,6,6
        0       6,6,6,6,6,6,6
        0       6,6,6,6,6,6,6,6
        0       6,6,6,6,6,6,6,6,6
        0       6,6,6,6,6,6,6,6,6,6
        0       6,6,6,6,6,6
        0       6,6,6,6,6
        0       6,6
        0       6

W
        0       E.
        0       E..
        0       E.E
        0       E.E.
        0       E.E..
        0       E.E.W
        0       E

B
        0       -NN
        0       -N
        0       N
        0       NN
        0       +N
        0       +NN

Q
        0       KX
        0       KH
        0       KHQ
        0       KFK.K
        0       KIK
        0       K(Z)
        0       KEK.K
        0       KAK
        0       KDK.K
        0       KOK
        0       IK
        0       FK.K
        0       EK.K
        0       AK
        0       (Z)
        0       (1Z)
        0       DK.K
        0       DK
        0       +NPKFK.K
        0       +NPKEK.K
```

```
                0     +NPKDK.K
                0     -NPKFK.K
                0     -NPKDK.K
                0     -NPKEK.K
                0     LK

        Z
                0     Q1Q1Q1Q1Q1Q
                0     Q1Q1Q1Q1Q1Q1Q
                0     Q1Q1Q1Q1Q1Q1Q1Q
                0     Q1Q1Q1Q1Q1Q1Q1Q1Q
                0     Q1Q1Q1Q1Q1Q1Q1Q1
                0     Q1Q1Q1Q1Q1Q1Q1
                0     Q1Q1Q1Q1Q1Q1
                0     Q1Q1Q1Q1Q1
                0     Q1Q1Q1Q1Q
                0     Q1Q1Q1Q1
                0     Q1Q1Q1Q
                0     Q1Q1Q1
                0     Q1Q1Q
                0     Q1Q1
                0     Q1Q
                0     Q1
                0     Q

        Y
                0     /
                0     //
                0     ///
                0     ////
                0     //////

        1
                0     ,
                0     ,Y,
                0     ,Y
                0     Y
                0     Y,
                0     Y,Y

        G
                0     EQ
                0     GT
                0     GE
                0     LT
                0     LE
                0     NE

        M
                0     P
                0     -P
                0     +P

        O
                0     +
                0     *
                0     **
                0     -
                0     /
```

J

0 K
0 K.K
0 K.KEB
0 K.KE
0 K.KDB
0 K.KD
0 K.
0 K.D
0 K.DB
0 K.E
0 K.EB
0 .K
0 .KDB
0 .KEB

P

0 U
0 U(2)
0 J
0 (E)

2

0 3
0 3,3
0 3,3,3,3
0 3,3,3,3,3,3
0 3,3,3,3,3,3,3
0 3,3,3,3,3,3,3,3
0 3,3,3,3,3,3,3,3,3
0 3,3,3,3,3
0 3,3,3

3

0 F
0 EH

E

0 M
0 MOM
0 MOMOM
0 MOMOMOM
0 MOMOMOMOM
0 MOMOMOMOMOM
0 MOMOMOMOMOMOM
0 MOMOMOMOMOMOMOM
0 MOMOMOMOMOMOM

Input Statement

```
DIMENTION K(22)

IFINISH=0

M=3

READ(5,21)

FORMAT(1X,FK.2)
```

Statement Output

```
DIMENSIONK(22)      trivial error recovered

****** MODIFICATION ******
IFINIS=0            more-than-trivial error recovered

M=3                 a correct statement

****** TIME TRAP ****** SYNTAX ERROR
READ(5,21)          a failure to recover an error

****** MODIFICATION ******
21  FORMAT(1X,F1.2)
```

```
EDASH=I.*COS(M)*E)

WRITE(6,20

FORMAT(4H NAME 12X/X)

WRITE(0,22) EDASH

FORMAT(1X,F10.0/)

READ(5,23) K

FORMIT(X 22A6)

M=IFINISH/2

DO 99 =1,M

K(J+1-2)=K(J)

CONTINIE

READ(5,24)KPLUS WAYB

FORMAT(I3,X,F9.5)

96(M .LT. 2I KPLUS=2

K(1)=(KPLUS+3) 14

WRITE(6,21),K(1)

WRITE(60,25)(J,K(J),J=1,M)

FORMAT(16H=FILE LENGTH =  I2/(5X 19A1))

GO TO (11,12) M

RETUURN

STOP
```

```
****** TIME TRAP ****** SYNTAX ERROR
EDASH=I.*COS(M)*E)

****** MODIFICATION ******
WRITE(6,20)         a 'correct' recovery

****** MODIFICATION ******
20  FORMAT(4H NAM)2X/1X)   wrong Hollerith count value
                           recovered.
WRITE(0,22)EDASH

****** MODIFICATION ******
22  FORMAT(1X,F10.1/)      'shift-key' error recovered

READ(5,23)K

****** MODIFICATION ******
23  FORMAT(1X,2A6)

****** MODIFICATION ******
M=IFINIS/2          seven letter variable name
                    truncated to six
****** TIME TRAP ****** SYNTAX ERROR
DO 99 =I,M

****** MODIFICATION ******
K(J+1,2)=K(J)       invalid subscript expression recovered.

99  CONTINUE        a trivial error recovery

****** MODIFICATION ******
READ(5,24)KPLUSH,YB     missing comma in variable list
                        recovered

****** MODIFICATION ******
24  FORMAT(I3,/,F9.5)

****** MODIFICATION ******
IF(M.LT.2)KPLUS=2       a 'correct' recovery

****** MODIFICATION ******
K(1)=(KPLUS+3)/4

****** MODIFICATION ******
WRITE(5,21)K(1)     inserted comma recovered

****** MODIFICATION ******  the unit device number altered
WRITE(6,25)(J,K(J),J=1,M)  to the most likely possibility the
                           standard output device number '6'
****** MODIFICATION ******
25  FORMAT(16H=FILE LENGTH =  I2/(5X,9A1))

****** MODIFICATION ******
GOTO(11,12),M           a 'correct' recovery

11  RETURN      a trivial error recovery

12  STOP
```

```
          END OF PUFFT BATCH
TIME FOR PROCESSING LAST          26 STATEMENTS WAS          195
PROCESSING TIME WAS                    195 SECONDS .
CARDS ANALYSED          27
SYSTEMS CARDS            1        3.70 PER CENT OF TOTAL
DATA CARDS               0        0.00
FORTRAN                 26       96.30

CONTINUATION CARDS       0

FORTRAN STATEMENTS      26

COMMENT CARDS            0

PROGRAM STATEMENTS      26

ARITHMETIC               6       23.08 PER CENT
   0 STATEMENTS IN EXCESS OF 199 CHARACTERS AND THUS NOTANALYS
        TREE        F                                        -YSED
           0     COMMONL:COMMON1
           0     COMMON/U/L:COMMON/1/2
           0     COMPLEXL:COMPLEX1
           1     CONTINUE:CONTINUE
           0     CALLU(2):CALL1(2)
           0     CALLU(2:CALL1(2
           0     CALLU:CALL1
           1     IF(WG.W):IF(12.3)4
           0     IF(WG.W8.WG.W):IF(12.34.56.7)8
           0     IF(WG.W8.WG.W8:IF(12.34.56.789
           0     IF(W)S,S,S:IF(1)2,3,4
           0     INTEGERL:INTEGER1
           0     DOSU=D:DO12=3
           0     DOUBLEPRECISIONL:DOUBLEPRECISION1
           1     DIMENSIONL:DIMENSION1
           0     DATA7/2/,:DATA1/2/,3
           0     DATA7/2/:DATA1/2/
           0     DATA7/2:DATA1/2
           0     DATA7:DATA1
           2     WRITE(R,S)7:WRITE(1,2)3
           1     WRITE(R,S):WRITE(1,2)
           0     WRITE(R,U):WRITE(1,2)
           0     WRITE(R)7:WRITE(1)2
           1     WRITE(U,S)7:WRITE(1,2)3
           0     WRITE(U,U):WRITE(1,2)
           0     WRITE(U)7:WRITE(1)2
           6     FORMAT(Z):FORMAT(1)
           0     FORMAT(1Z):FORMAT(12)
           0     FUNCTIONU(2):FUNCTION1(2)
           0     FUNCTIONU(2:FUNCTION1(2
           0     GLOTOS:GOTO1
           0     GLOTO(S,S,S),U:GOTO(1,2,3),4
           0     GLOTO(S,S,S,:GOTO(1,2,3,4
           1     GLOTO(S,S),U:GOTO(1,2),3
           0     GLOTO(S),U:GOTO(1),2
           0     GLOTOU),(S):GOTO1,(2)
           0     GLOTOU),(S,S):GOTO1,(2,3)
           0     GLOTOU),(S,S,:GOTO1,(2,3,4
           1     RETURNU:RETURN
           2     READ(R,S)7:READ(1,2)3
```

```
 0     READ(R,U)0READ(1,2)
 0     READ(R)70READ(1)2
 0     READ(U,S)70READ(1,2)3
 0     READ(U,U)0READ(1,2)
 0     READ(U)70READ(1)2
 0     READS,70READ1,2
 0     REALL0REAL1
 0     REW0INDR0REWIND1
 0     REW0INDU0REWIND1
 0     END0END
 0     ENDFILER0ENDFILE1
 0     ENDFILEU0ENDFILE1
 0     ENTRYU0ENTRY1
 0     ENTRYU(2)0ENTRY1(2)
 0     ENTRYU(20ENTRY1(2
 0     EXTERNALL0EXTERNAL1
 0     EQUIVALENCE(U00EQUIVALENCE(12
 1     STOP0STOP
 0     STOP50STOP1
 0     SUBROUTINEU(2)0SUBROUTINE1(2)
 0     SUBROUTINEU(20SUBROUTINE1(2
 0     SU0BROUTINEU0SUBROUTINE1
 0     NOAMELIST/U/00NAMELIST/1/2
 0     BL1OCKDATA0BLOCKDATA
 0     BA0CKSPACEU0BACKSPACE1
 0     BA0CKSPACER0BACKSPACE1
 0     L00GICALLOLOGICAL1
 0     PA0USE0PAUSE
 0     PA0USE50PAUSE1
 0     PUNCHS,U00PUNCH1,23
 0     PRINTS,U00PRINT1,23
 0     A0SSIGN50TOU0ASSIGN1TO2
TREE        R
 3     6
 2     5
 0     10
 0     11
 0     12
 0     13
 0     14
 0     15
 0     1
 0     19
 0     4
 0     7
 0     8
 0     9
 0     3
 0     2
TREE        S
 0     NNN
 0     NNNN
 0     NNNNN
 8     NN
 0     N
TREE        U
 2     AXXXX
 3     AXXXXX
 0     AXXX
 0     AXX
```

```
  2     AX
 17     A
TREE          A
  7     I
  1     A
  2     N
  1     O
  0     C
  5     S
  0     R
  0     T
  2     P
  2     L
  1     E
  1     B
  5     M
  0     G
  0     X
  9     K
  1     D
  2     F
  1     Y
  1     W
  1     H
  5     J
  2     U
  0     V
  0     Z
  0     Q
TREE          X
 25     A
  0     N
TREE          N
 18     1
  3     0
 17     2
  4     3
  3     5
  3     4
  0     8
  2     9
  2     6
  0     7
TREE          T
```

The Strategy Tree

number of times strategy successfully used.

```
235   BACKSPACE     0 CHARACTERS,AND SEARCH ASSUMING   0 CHARACTERS OMITTED WHILST FREQUENCY NOT LESS THAN      0
 10   BACKSPACE     0 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS WRONG WHILST FREQUENCY NOT LESS THAN    109
  3   BACKSPACE     0 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS INSERTED WHILST FREQUENCY NOT LESS THAN  125
  1   BACKSPACE     1 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS OMITTED WHILST FREQUENCY NOT LESS THAN  143
  2   BACKSPACE     0 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS OMITTED WHILST FREQUENCY NOT LESS THAN  166
  0   BACKSPACE     2 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS OMITTED WHILST FREQUENCY NOT LESS THAN  210
  0   BACKSPACE     3 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS OMITTED WHILST FREQUENCY NOT LESS THAN  232
  0   BACKSPACE     0 CHARACTERS,AND SEARCH ASSUMING   3 CHARACTERS OMITTED WHILST FREQUENCY NOT LESS THAN 1000
  0   BACKSPACE     0 CHARACTERS,AND SEARCH ASSUMING   2 CHARACTERS INSERTED WHILST FREQUENCY NOT LESS THAN    0
  1   BACKSPACE     0 CHARACTERS,AND SEARCH ASSUMING   2 CHARACTERS WRONG WHILST FREQUENCY NOT LESS THAN      0
  0   BACKSPACE     0 CHARACTERS,AND SEARCH ASSUMING   2 CHARACTERS OMITTED WHILST FREQUENCY NOT LESS THAN    0
  0   BACKSPACE     1 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS WRONG WHILST FREQUENCY NOT LESS THAN      0
  0   BACKSPACE     1 CHARACTERS,AND SEARCH ASSUMING   1 CHARACTERS INSERTED WHILST FREQUENCY NOT LESS THAN    0
  0   BACKSPACE     2 CHARACTERS,AND SEARCH ASSUMING   2 CHARACTERS WRONG WHILST FREQUENCY NOT LESS THAN      0
TREE          K
 28   N
```

```
 4      NN
 0      NNN
 0      NNNNNNNNNNN
 0      NNNNNNNNNN
 0      NNNNNNNNN
 0      NNNNNNNN
 0      NNNNNNN
 0      NNNNNN
 0      NNNNN
TREE          V
 9      U
 4      U(C)
 1      U(C,C)
 0      U(C,C,C,C)
 0      U(C,C,C,C,C)
 0      U(C,C,C,C,C,C)
 0      U(C,C,C,C,C,C,C)
 0      U(C,C,C)
TREE          L
 1      V
 0      V,V
 1      V,V,V
 0      V,V,V,V,V,V,0
 0      V,V,V,V,V,V
 0      V,V,V,V,V
 0      V,V,V,V
TREE          C
 1      U
 1      U+K
 0      U-K
 4      K
 0      K*U
 0      K*U-K
 0      K*U+K
TREE          H
 5      V=E01=2
TREE          8
 0      OR
 0      AND
TREE          D
 1      K,U
 0      K,U,K
 0      K,U,U
 0      K,K
 0      K,K,K
 0      K,K,U
 0      U,U
 0      U,U,U
 0      U,U,K
 0      U,K
 0      U,K,U
 0      U,K,K
TREE          4
 1      L=D)
TREE          5
 1      (4
 0      ((4,4
 0      (((4,4,4
 0      ((((4,4,4,4
```

```
TREE        6
 5     V
 1     5
TREE        7
 0     6,6,6
 0     6,6,6,6
 0     6,6,6,6,6,6,6
 0     6,6,6,6,6,6,6,6
 0     6,6,6,6,6,6,6,6,6
 0     6,6,6,6,6,6,6,6,6,6
 0     6,6,6,6,6,6
 0     6,6,6,6,6
 1     6,6
 4     6
TREE        W
 1     E.
 0     E..
 0     E.E
 0     E.E.
 0     E.E..
 0     E.E.W
 1     E
TREE        B
 0     -NN
 0     -N
 0     N
 0     NN
 0     +N
 0     +NN
TREE        Q
 6     KX
 0     KH
 2     KHQ
 0     KFK.K
 0     KIK
 0     K(Z)
 0     KEK.K
 2     KAK
 0     KDK.K
 0     KOK
 2     IK
 3     FK.K
 0     EK.K
 0     AK
 1     (Z)
 0     (1Z)
 0     DK.K
 0     OK
 0     +NPKFK.K
 0     +NPKEK.K
 0     +NPKDK.K
 0     -NPKFK.K
 0     -NPKDK.K
 0     -NPKEK.K
 0     LK
TREE        Z
 0     Q1Q1Q1Q1Q1Q
 0     Q1Q1Q1Q1Q1Q1Q
 0     Q1Q1Q1Q1Q1Q1Q1Q
 0     Q1Q1Q1Q1Q1Q1Q1Q1Q
```

```
    0    Q1Q1Q1Q1Q1Q1Q1Q1
    0    Q1Q1Q1Q1Q1Q1Q1
    0    Q1Q1Q1Q1Q1Q1
    0    Q1Q1Q1Q1Q1
    0    Q1Q1Q1Q1Q
    0    Q1Q1Q1Q1
    0    Q1Q1Q1Q
    0    Q1Q1Q1
    0    Q1Q1Q
    1    Q1Q1
    6    Q1Q
    0    Q1
    0    Q
  TREE       Y
    4    /
    0    //
    0    ///
    0    ////
    0    //////
  TREE       1
    4    ,
    1    ,Y,
    0    ,Y
    3    Y
    0    Y,
    0    Y,Y
  TREE       G
    0    EQ
    0    GT
    0    GE
    1    LT
    0    LE
    0    NE
  TREE       M
   12    P
    0    -P
    0    +P
  TREE       O
    1    +
    0    *
    0    **
    0    -
    2    /
  TREE       J
    6    K
    0    K.K
    0    K.KEB
    0    K.KE
    0    K.KDB
    0    K.KD
    0    K.
    0    K.D
    0    K.DB
    0    K.E
    0    K.EB
    0    .K
    0    .KDB
    0    .KEB
  TREE       P
    4    U
```

```
 1     U(2)
 6     J
 1     (E)
TREE        2
 1     3
 0     3,3
 0     3,3,3,3
 0     3,3,3,3,3,3
 0     3,3,3,3,3,3,3
 0     3,3,3,3,3,3,3,3
 0     3,3,3,3,3,3,3,3,0
 0     3,3,3,3,3
 0     3,3,3
TREE        3
 1     E
 0     EH
TREE        E
 6     M
 3     MOM
 0     MOMOM
 0     MOMOMOM
 0     MOMOMOMOM
 0     MOMOMOMOMOM
 0     MOMOMOMOMOMOM
 0     MOMOMOMOMOMOM
 MINIMUM NUMBER OF AVAILABLE CELLS WAS                    4
TIME IN TRUNDL WAS                  194 SECONDS
NUMBER OF STATEMENTS PROCESSED                  26 NUMBER OF FAILURES                  3
KUMTIM=        194 KUMTRN=          26 KUMSUC=          23
NUMBER OF STATEMENTS OF MORE THAN 199 CHARS.          0
NUMBER OF STATEMENTS TIME TRAPPED AFTER   20 SECONDS WAS          3
NUMBER OF STATEMENTS MODIFIED          16
AVERAGE NUMBER OF CHARACTERS PROCESSED PER SECOND IS   1.9794, AVERAGE PERCENTAGE OF FAILURES IS    11.538
AVERAGE STATEMENT LENGTH   14.769 BEFORE PROCESSING, AND   14.192 AFTER PROCESSING
NUMBER OF CARDS PROCESSED          29

NUMBER OF CELLS                1772
```