THE COMPUTER STORAGE, RETRIEVAL AND SEARCHING OF GENERIC STRUCTURES IN CHEMICAL PATENTS:

THE MACHINE-READABLE REPRESENTATION OF GENERIC STRUCTURES

A Study Submitted in Fulfilment of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

by

JOHN MORDAUNT BARNARD

December 1982

Department of Information Studies

University of Sheffield

CONTENTS

CONTENTS

THE COMPUTER STORAGE, RETRIEVAL AND SEARCHING OF GENERIC STRUCTURES IN CHEMICAL PATENTS: THE MACHINE-READABLE REPRESENTATION OF GENERIC STRUCTURES

Thesis submitted for the Degree of Ph.D. by J.M. Barnard

ABSTRACT

The nature of the generic chemical structures found in patents is described, with a discussion of the types of statement commonly found in them. The available representations for such structures are reviewed, with particular note being given to the suitability of the representation for searching files of such structures. Requirements for the unambiguous representation of generic structures in an "ideal" storage and retrieval system are discussed.

The basic principles of the theory of formal languages are reviewed, with particular consideration being given to parsing methods for context-free languages. The Grammar and parsing of computer programming languages, as an example of artificial formal languages, is discussed. Applications of formal language theory to chemistry and information work are briefly reviewed.

GENSAL, a formal language for the unambiguous description of generic structures from patents, is presented. It is designed to be intelligible to a chemist or patent agent, yet sufficiently

ABSTRACT

formalised to be amenable to computer analysis. Detailed description is given of the facilities it provides for generic structure representation, and there is discussion of its limitations and the principles behind its design.

A connection-table-based internal representation for generic structures, called an ECTR (Extended Connection Table Representation) is presented. It is designed to represent generic structures unambiguously, and to be generated automatically from structures encoded in GENSAL. It is compared to other proposed representations, and its implementation using data types of the programming language Pascal described.

An interpreter program which generates an ECTR from structures encoded in a subset of the GENSAL language is presented. The principles of its operation are described.

Possible applications of GENSAL outside the area of patent documentation are discussed, and suggestions made for further work on the development of a generic structure storage and retrieval system based on GENSAL and ECTRs.

NOTE

The work described in this Thesis has been undertaken as part of a more comprehensive project on the computer storage and retrieval of generic chemical structures in patents. Whilst this has involved close liason with the other research worker on the project, S.M. Welford, the work described in this Thesis is entirely that of the author.

A number of publications have appeared describing work on the project;[174-177] the substance of Chapter 3 appeared in the second of these[176] and the substance of Chapter 4 in the third.[177]

In addition, presentations have been given at the following meetings:

1. Chemical Notation Association (UK) Seminar on "Structure Searching in the Published Literature", Daresbury, March 1980.

2. Chemical Notation Association (UK) Seminar on "The Future of Chemical Documentation", Exeter, September 1982.

ACKNOWLEDGEMENTS

I should like to thank my supervisor, Professor M.F. Lynch for his constant support and encouragement during the course of this research, and for inviting me to participate in it.

One of the satisfactions of this project has been the opportunities it has afforded to discuss my work with a wide range of individuals from different organisations, especially in the chemical and pharmaceutical industries. I should particularly like to thank the following for their many helpful comments and advice: John Silk (EUSIDIC, formerly ICI PPD), Peter Steele (Glaxo), Clive Tomlin, Richard Waterman, and David Pearson (all of ICI PPD), Frank Jackson (Pfizer), Charles Oppenheim and Peter Norton (Derwent), George Adamson (ICI Pharmaceuticals), Bill Town and Ole Norager (ISPRA), Todd Wipke (University of California), Claus Suhr (BASF) and Dr G.Pötscher (Fachinformatationszentrum Chemie, Berlin).

In addition, from my own Department I should like to thank Peter Willett, and Annette von Scholley who has still not managed to send my program down.

I should like to thank Mike Elder and Steve Hull (SERC, Daresbury) for providing the software for the Feldmann chemical structure graphics system, and for helping me to get it working.

ACKNOWLEDGEMENTS

None of the programming work in this project would have been possible without the dedication to duty of the staff of the University Computing Services Department, and I should especially thank Richard Gilbert and Chris Martin for so efficiently removing all the bugs I found in their Pascal compiler. Also, from the Computer Science Department, I would like to thank Siobhan North for setting me along the right road in the matter of syntax analysis.

Finally, and most importantly, I should like to express my gratitude and appreciation to my friend and colleague Stephen Welford, who has spent countless hours discussing this project with me, and who also put up with sharing an office with me for two and a half years.

CHAPTER 1

GENERIC STRUCTURES IN PATENTS

"Bloody instructions, which, being taught
return to plague the inventor"

Macbeth, Act I, Sc. vii

On the basis of those words, it might well be supposed that Macbeth was an information scientist in the chemical or pharmaceutical industries. Those industries are not only prolific generators of patent documents, but are also major users of patent information, and the efforts made to protect a company's invention in drafting a patent return to cause many problems for information searchers in the patent literature.

The increase in the number of chemical patent documents published in recent years has been prodigious, and the increase has continued in spite of a slight fall-off in the number of journal articles published. In 1981 more than 71 000 patents were abstracted in Chemical Abstracts, as compared with fewer than 62 000 the previous year, [1] and this continues a trend which can be traced back many decades [2] though at least part of the increase can be explained by improvements in the range of countries covered by Chemical Abstracts; its relatively poor coverage compared to other indexing systems had previously attracted criticism. [3]

A further factor in the increase in published patent documents is the change in patent legislation in a number of countries, including Britain, during the 1970's. [4, 5] This has resulted in a move from the publication of examined and accepted patents only to the publication of unexamined applications. Initially this led to the sudden publication of backlogs of applications, increasing the figures for patent documents published, but it has also led to a change in the actual substance of patent claims, especially in the chemical area, which has itself caused problems for patent documentation systems. [6]

This is because patents for chemicals and pharmaceuticals frequently do not lay claim to the single compound which the company taking out the patent intends to market, but rather lays claim to a whole class of compounds having broadly the same properties. In the initial application for a patent, a company may attempt to claim as wide a range of compounds as possible, partly to cover anything which might conceivably have the desired activity (patent application normally takes place well before testing and development of "lead compounds" has been completed), partly to intimidate rival companies who may be working in the same area, and partly to disguise the true nature of the invention. It is very possible that the initial application may have to be modified before it can be accepted and a patent granted, but under the early publication system now adopted by most countries, it is the initial application which is published first. This retains its significance after examination - and many patent applications are in fact abandoned, no examination taking place and no patent being granted - as the information contained in it may affect the validity of future patents.

The class of compounds claimed in a patent is described by means of a **generic structure** which contains both fixed and variable parts, the extent of the variation defining the size of the class of structures.

1.1. THE NATURE OF GENERIC STRUCTURES

1.1.1. Patent Claims

In 1924 an American chemist, Eugene A. Markush, applied for a patent for a class of novel pyrazolone dyes,[7] but his application was rejected on the grounds that it claimed alternatives. After making suitable changes to the wording of his application in order to leave out the word "or", it was accepted, and since then the term "Markush" has been applied to this type of generic structure. Rosa [8] has discussed the legal wrangles over this and other applications, and outlined the type of generic structure which may be claimed under the precedent set by Markush, though the rigid "Rule Against Or", [9] which never applied in other countries, has now been abolished in the United States too.

The expression "Markush Structure" is now used rather loosely to refer to a wide variety of types of generic structure, though U.S. patent attorneys use it to refer specifically to patents granted under the precedent established by Markush's pyrazolone dyes application. On account of this special legal meaning the expression has generally been avoided in the present work following advice from Silk [10] and despite its use by many other authors in the field, and the expression "generic structure" is used throughout this Thesis.

A single generic structure may cover an enormous, and in some cases infinite, number of specific compounds, [11] only a tiny fraction of which have actually been tested for the claimed activity. Beton [12] cites the example of a patent application on sulphathiazole which was rejected because, of the at least 93 million specific compounds covered, only two had been shown to have the claimed activity. In the same paper however, he refers to the original patent on the Ziegler process for ethylene polymerisation in which aluminium trialkyl is claimed as catalyst. Following grant of this patent, Ziegler found that alkyl aluminium halides and organomagnesium compounds could also be used, and was obliged to make further applications to cover them also. However, this still left him with no patent protection for the use of such catalysts in the polymerisation of other alkenes.

These examples illustrate the need to formulate a patent specification sufficiently widely to cover all the compounds with the required activity, yet sufficiently narrowly not to claim untested compounds which are actually inactive.

1.1.2. Types of Generic Structure

Valance [13] has discussed the variety of different types of statement that may be found in generic structures, with a survey of their relative frequencies. Sneed, Turnipseed and Turpin [14] have attempted a rudimentary classification of generic structures, dividing them into **determinate** and **indeterminate** structures, the former having variable substructure groups (all defined) occurring with variable frequency at fully-defined positions of attachment, and the latter comprising all other generic structures, including those involving verbal expressions, undefined substructures and undefined positions of attachment. Concentrating on **determinate** structures, they give examples of the different types of expression that may be found.

A similar classification has been given by Krishnamurthy and Lynch [15, 16] dividing generic structures into **delimited** and **undelimited** structures, though these classes are not identical with Sneed et al.'s **determinate** and **indeterminate** structures. **Delimited** structures are essentially those which cover a finite (even if very large) number of specific compounds; **undelimited** those which cover an infinite number of specific compounds.

In the present work these classifications have not been found helpful, and analysis of generic structures has been based on an approach given by Geivandov [17] which views such a structure as a (possibly vestigial) constant part to which are attached variable parts that can vary in their chemical nature, in their position of attachment to the constant part, and in their multiplicity of occurrence. This concept may be extended to encompass the idea of a "Markush within a Markush" so that each variable part can have further variable parts attached to it, continuing to any level.

On this basis, two opposite "extremes" of generic structures may be identified: that where the "variable" parts are fully defined in terms of nature, position and multiplicity, in which case the structure is a specific structure identifying a unique chemical substance, and that where the variable parts are totally undefined, in which case the structure is a substructure which may be found embedded in any of a potentially infinite variety of specific structures.

Between these extremes lie generic structures with incompletelydefined variable parts. Any variable part may still have an infinite number of different possible values, but it is none the less restricted in some way. For example, the term "alkyl" strictly covers the infinite variety of radicals containing carbon and hydrogen only, with no double or triple bonds and no rings, but it nonetheless restricts the variety of values a group defined as "alkyl" can take.

1.1.3. Generic Structures Outside Patents

Generic structures are also found outside patents. They appear in the journal literature, where a large number of related compounds have been tested for a particular property or activity, and in this case a generic structure is essentially a shorthand way of listing the compounds tested. Figure 1.1 shows an example of a generic structure from the Journal of Medicinal Chemistry.

Generic structures may be used as queries in some chemical structure search systems, with databases of specific structures. Generally, only very simple generic structures can be used, but the recently-developed COUSIN system allows more complicated queries. This is discussed more fully in Section 1.4.7.

The description of generalised chemical reactions can involve the use of generic structures for the generalised reactants and products, though no reaction indexing system has yet been developed using such reactant and product descriptions.

Hallucinogenic Amphetamines *Journal of Medicinal Chemistry, 1977, Vol. 20, No. 12* 1633

Table I. Substituted Amphetamines and Predicted Hallucinogenic Activity

No.	Ring position and group: 2	3	4	5	6	$^3\chi_p$	$^4\chi_p$	$^4\chi_{pc}^v$	Exptl[a] $\log \mu$	Calcd[a] $\log \mu$
1			OCH_3			3.348	1.034	0.469	0.59	0.55
2	OCH_3		OCH_3			4.124	1.508	0.642	0.67	0.87
3	OCH_3			OCH_3		4.124	1.683	0.638	0.87	1.06
4		OCH_3	OCH_3	OCH_3		4.808	1.830	0.739	0.37	0.55
5	OCH_3	OCH_3		OCH_3		4.830	2.083	0.765	0.63	1.01
6	OCH_3		OCH_3		OCH_3	4.853	2.031	0.798	1.03	1.12
7	OCH_3	OCH_3			OCH_3	4.933	2.045	0.810	1.14	1.06
8	OCH_3		OCH_3	OCH_3		4.892	2.058	0.785	1.26	1.00
9	OCH_3	OCH_3	OCH_3	OCH_3		5.629	2.363	0.917	0.86	0.92
10		$-OCH_2O-$				4.203	1.705	0.576	0.41	0.21
11		OCH_3	$-OCH_2O-$			4.925	2.252	0.707	0.43	0.62
12	$-OCH_2O-$		OCH_3			5.043	2.272	0.756	0.48	0.80
13	OCH_3	$-OCH_2O-$				5.027	2.197	0.753	1.00	0.71
14	OCH_3		$-OCH_2O-$			4.993	2.317	0.751	1.08	0.71
15	OCH_3	OCH_3	$-OCH_2O-$			5.746	2.749	0.885	0.75	1.09
16	OCH_3	$-OCH_2O-$		OCH_3		5.761	2.906	0.887	1.13	1.29
17	OCH_3		OC_2H_5	OCH_3		5.027	2.285	0.768	1.22	0.93
18	OCH_3		Br	OCH_3		4.574	1.762	1.157	2.71	2.92
19	OCH_3		CH_3	OCH_3		4.574	1.762	0.868	1.89	1.85
20	OCH_3		C_2H_5	OCH_3		4.892	2.058	0.910	2.01	1.70
21	OCH_3		n-C_3H_7	OCH_3		5.027	2.285	0.850	1.94	1.60
22	OCH_3		n-C_4H_9	OCH_3		5.296	2.436	0.880	1.63	1.34
23	OCH_3		n-C_5H_{11}	OCH_3		5.546	2.548	0.880	1.09	1.09

[a] Molar basis, ref 1 and 5, converted by multiplying by the ratio of molecular weights of amphetamine to mescaline.

Figure 1.1: A generic structure from the journal literature

1.2. GENERIC STRUCTURE DESCRIPTIONS IN PATENTS

The manner of description of generic structures in patents from different countries is basically equivalent, and an example of such a description from a recent British patent is shown in Figure 1.2.

1.2.1. The Constant Part

In a typical patent specification, or abstract, there is a structure diagram for the constant part, in which the attached variable parts are indicated by symbols such as R, X, R", R_2 etc. There is little or no standardisation of the symbols used, and occasionally valid atomic symbols (such as B or C) appear as structural variables, which can cause ambiguity and confusion.

The variables may be attached to the constant part at fixed or variable positions, the latter normally being indicated by the convention of a bond going into the centre of a ring, or sometimes, where the attachment is to a chain, by means of a brace over the possible atoms of attachment. The variables may have one or two connections to the constant part, with any bond orders, or infrequently three or more.

Multiplicity of occurrence of certain portions (normally structural variables) of the structure diagram is often indicated by a subscript to a symbol, or to parentheses around a multiplied portion. The subscript may be a single integer, a range of integers, or an alphabetical or other symbol that is defined elsewhere. Examples are:

$(R')_3$ $(\text{-CONH-})_p$ X^2_{1-4}

1.2.2. The Variable Parts

Following the constant part, the variables introduced in it are defined, usually by listing the alternative values for each structural variable. However, several different types of

1 GB 2 000 512 A 1

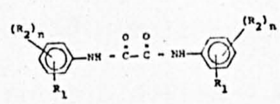
SPECIFICATION

Improvements in or relating to organic compounds

The present invention concerns industrial enamels containing u.v. absorbing compounds, which enamels afford coatings with improved resistance to weathering.

More particularly, the present invention provides an industrial enamel having incorporated therein one or more u.v. absorbing oxamides, i.e. u.v. absorbing compounds featuring the structural unit –NHCOCONH–.

The preferred u.v. absorbing oxamides in the industrial enamels of the present invention are those of formula I,



I

in which each R_1, independently, is hydrogen, halogen, hydroxyl, $C(_{1-18})$alkyl, (C_{1-18})alkoxy, phenyl or phenoxy,
each R_2, independently, is hydrogen, halogen, hydroxyl, (C_{1-8})alkyl or (C_{1-8})alkoxy,
and each n, independently, is 1, 2 or 3.

In the above definition of formula I, halogen means fluorine, chlorine or bromine. Preferably halogen is chlorine or bromine, more preferably chlorine.

When any R_1 is alkyl, this is preferably (C_{1-8})alkyl, more preferably (C_{1-4})alkyl, and most preferably ethyl.

When any R_1 is alkoxy, this is preferably (C_{1-12})alkoxy, more preferably (C_{1-8})alkoxy, even more preferably (C_{1-4})alkoxy, and most preferably $(C_2$ or $C_4)$alkoxy.

Each R_1, independently, when signifying a substituent other than hydrogen, is preferably in an ortho-position to the oxamide linking moiety –NHCOCONH–.

Of all the significances of R_1, hydrogen, hydroxyl, alkyl or alkoxy, especially R_1', as hereinafter defined, are preferred, and alkyl or alkoxy, especially (C_{1-8})alkyl or (C_{1-8})alkoxy, respectively, are more preferred. Most preferably, one of the two R_1's is alkyl and the other is alkoxy.

When any R_2 is alkyl, this is preferably (C_{1-4})alkyl, more preferably $(C_1$ or $C_4)$alkyl, and most preferably tert.-butyl.

When any R_2 is alkoxy, this is preferably (C_{1-4})alkoxy, and more preferably methoxy or ethoxy.

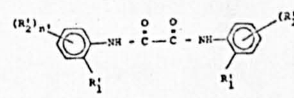
When any R_2 is hydroxyl, this is preferably in the para-position to the oxamide linking moiety –NHCOCONH–. Furthermore, if R_1 is alkoxy in an ortho-position to the oxamide linking moiety –NHCOCONH– and any R_2 on the same phenyl ring as this R_1 is alkyl, any such alkyl group R_2 is preferably in the para-position to R_1. If R_1 is alkyl in an ortho-position to the oxamide linking moiety –NHCOCONH– and any R_2 on the same phenyl ring as this R_1 is alkyl, any such alkyl group R_2 is preferably in the meta-position to R_1.

Of all the significances of R_2, hydrogen, hydroxyl, alkyl or alkoxy are preferred, hydrogen, alkyl or alkoxy, especially R_2'', or R_2''', as hereinafter defined in formula Ib, are more preferred, and hydrogen or (C_{1-4})alkyl are most preferred.

If any R_1 is phenyl or phenoxy, n on the same phenyl ring as the phenyl or phenoxy substituent is preferably 1.

In general n is preferably 1 or 2, more preferably 1.

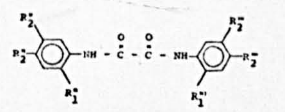
A preferred class of compounds of formula I is constituted by the compounds of formula Ia,



Ia

in which each R_1', independently, is hydrogen, hydroxyl, (C_{1-8})alkyl or (C_{1-12})alkoxy,
each R_2' independently, is hydrogen, hydroxyl, (C_{1-8})alkyl or $C_{1-8})$alkoxy,
and each n' independently, is 1 or 2.

A preferred class of compounds of formula Ia is constituted by the compounds of formula Ib,



Ib

in which R_1'' is (C_{1-8})alkoxy,
R_1''' is (C_{1-})alkyl,
each R_2'' independently, is hydrogen, (C_{1-4})alkyl or (C_{1-4})alkoxy,
and either both R_2''' 's are hydrogen, or one of the two is hydrogen and the other is (C_{1-4})alkyl.

The compounds of formula I are either known or can be produced in analogous manner to the known compounds from available starting materials.

The industrial enamels of the present invention are preferably those for enamelling vehicles. Especially suitable industrial enamels are those which contain, as a binder, combinations of oil-modified polyester resins (oil-modified alkyd resins) and melamine resins, combinations of addition cross-linking polyacrylate resins and melamine resins, combinations of saturated polyesters and melamine resins, cross-linking polyacrylate resins, two-component polyacrylate resins consisting of hydroxy-containing polyacrylate resin and aliphatic or aromatic isocyanates, thermoplastic polyacrylate resins, or two-component polyurethane resins consisting of hydroxy-containing polyester and/or polyether resins hardened with aliphatic or aromatic isocyanates. Of these, thermoplastic polyacrylate resins, combinations of addition cross-linking polyacrylate resins and butanol-etherified melamine resins, and

Figure 1.2: Part of a British Patent Specification

expression may be used for these values.

There may be simple nomenclatural terms (e.g. "methyl", "cyclohexyl", "pyridyl", "amino", etc.) that represent single chemical entities, or terms or expressions that represent a limited group of such entities (e.g. "halogen", "alkali metal", etc.).

Alternatively, there may be further structure diagrams, perhaps introducing new symbols for structural variables, or further citing structural variables that have already been introduced. Such structure diagrams will normally have an indication of which atom or atoms is/are attached back to the constant part.

There may be linear formulae, which can represent single entities (e.g. "OH", "COOH", "$COOCH_3$" etc.), or include symbols for structural or multiplicative variables, or represent classes of structural entities (e.g. "C_6H_{13}").

There may be nomenclatural terms or expressions describing classes of structural entities, such as homologous series (e.g. "alkyl", "alkylcycloalkyl", "alkenyl" etc.). Frequently these are qualified by indications of the number of atoms, the degree of branching, or other factors (e.g. "straight-chain 1-6C alkyl"). Alternatively, the class described may be less well-defined (e.g. "heterocyclic ring system", "aryl" etc.).

Finally, there may be expressions describing groups in terms of their properties (e.g. "electron-withdrawing group", "photographically-useful group", "easily-hydrolysed group", "group known in the art" etc.).

In addition, all these types of expression may be further qualified by indications of position or multiplicity, or the statement that they are "substituted by" or "optionally substituted by" a further list of values. Occasionally the epithet "substituted" or "optionally substituted" may occur without any indication of the nature of the further substitution. Furthermore, certain of the alternatives listed may be indicated as preferred, possibly ranging over a hierarchy of preferability; expressions involving "preferably ... more preferably ... even more preferably ... most preferably" are not uncommon.

In some examples, two structural variables may be combined to form a ring which can be described by any of the methods given above; such combination may be a value for the two variables alternative to those given for each individually, if any. The structural entity specified as a value for the combination of the variables may consist only of the atoms added to those present in the constant part, or (more commonly) may also include those atoms of the constant part which are part of the ring formed. Occasionally two structural variables are combined to form an extra bond between the (adjacent) atoms to which they are each attached.

1.2.3. Conditional Expressions

Frequently, certain of the alternative values for structural and multiplicative variables are dependent upon the values of others, and this is indicated in patent specifications and abstracts in a variety of ways.

If there are several occurrences of a structural variable in the constant part, then the definition of it may specify that all its occurrences should have the same value, or different values etc. Alternatively, it may be specified that certain values for a variable are only possible when another variable has a particular value or values, or the possible values may be limited to a subset of the alternatives given originally when another variable has a particular value. There may be stipulations that a certain proportion of the occurrences of a variable should have a particular value etc. Sometimes these conditions and restrictions can become very complicated.

1.3. THE "MARKUSH PROBLEM"

In recent years chemical information scientists have tended to talk about the "Markush problem", and the possibilities for its solution. By this they refer to the problem of developing a computer system capable of storing and searching files of generic structures, especially those found in patents.

During the past two decades a great deal of work has been done on the developement of storage and retrieval systems for specific structures, and Warr [18] has recently reviewed the available software. Many excellent systems have appeared, for use both with a company's files of internally-developed compounds, and with "public" databases such as the Chemical Abstracts Registry file.

Amongst the former group are the CROSSBOW system (Computerised Retrieval of StructureS Based On Wiswesser) [19] in which structures are encoded in the Wiswesser Line Notation, [20] and more recently MACCS (Molecular ACCess System) [21] which has sophisticated facilities for graphical input of structure diagrams. The two main systems supporting the Chemical Abstracts Registry file are CAS ONLINE [22] which was developed by the Chemical Abstracts Service itself, and the French Systeme DARC (Description, Acquisition, Retrieval, Correlation) [23-25] which is also now available for in-house use. Although these systems support limited facilities for generic structure queries, none of them, as yet, has any facilities for generic file structures.

Jackson [26] has outlined the essential features of an "ideal" system for generic structures in patents, and achievement of these objectives could be regarded as a solution of the "Markush problem":

1. Total recall with minimum noise.
2. Include both generic structure and specific compounds.
3. Easy to use for encoding and retrieval.

4. Automatic input with error checks.

5. Available online.

6. Abstract and structure as output.

In his paper Jackson also surveys the existing systems available, and discusses the ways in which they fall short of the ideal. Existing chemical patent documentation systems have also been reviewed by a Japanese Study Team [27] and in a number of other publications. [28-30] The storage and retrieval of Markush structures was identified as a priority area for research by the British Library's Chemical Information Review Panel, which reported in 1978. [31, 32]

1.4. GENERIC STRUCTURE REPRESENTATIONS

An essential prerequisite for a satisfactory storage and retrieval system for generic structures is a satisfactory means of representing them for computer manipulation. A number of different forms of representation are used in existing systems and have been proposed for new systems, and these are discussed in this Section with some comments on the efficacy of the systems which use them.

Like those for specific structures, the forms of representation may broadly be divided into **ambiguous** and **unambiguous**; the former allow the same representation to stand for different structures, whereas in the latter each representation stands for only a single structure. All operational computer storage and retrieval systems for generic structures are based on ambiguous representations of the structure, and this is one reason for the unsatisfactory performance of existing systems.

1.4.1. Derwent Publications Ltd.

Derwent Publications Ltd. is a British company, owned by the Thompson Organisation, and it produces a variety of current awareness and retrospective search services, both for patents and in other areas, though patent documentation represents the major part of its business. The chemical area is well covered, and Derwent's services have been discussed recently by Kaback.[33-34, 6]

In general, non-chemical patents are included in the World Patent Index (WPI), and chemically-related ones in the Central Patent Index (CPI), of which three sections (Section B on pharmaceuticals ("FARMDOC"), Section C on agrochemicals ("AGDOC") and Section E on general chemistry ("CHEMDOC")) use a complex fragmentation code, the CPI code, to represent the chemical structures, generic and specific, shown in the patent in question.

Both WPI and CPI are available for searching online via the SDC Search Service, using the ORBIT software.

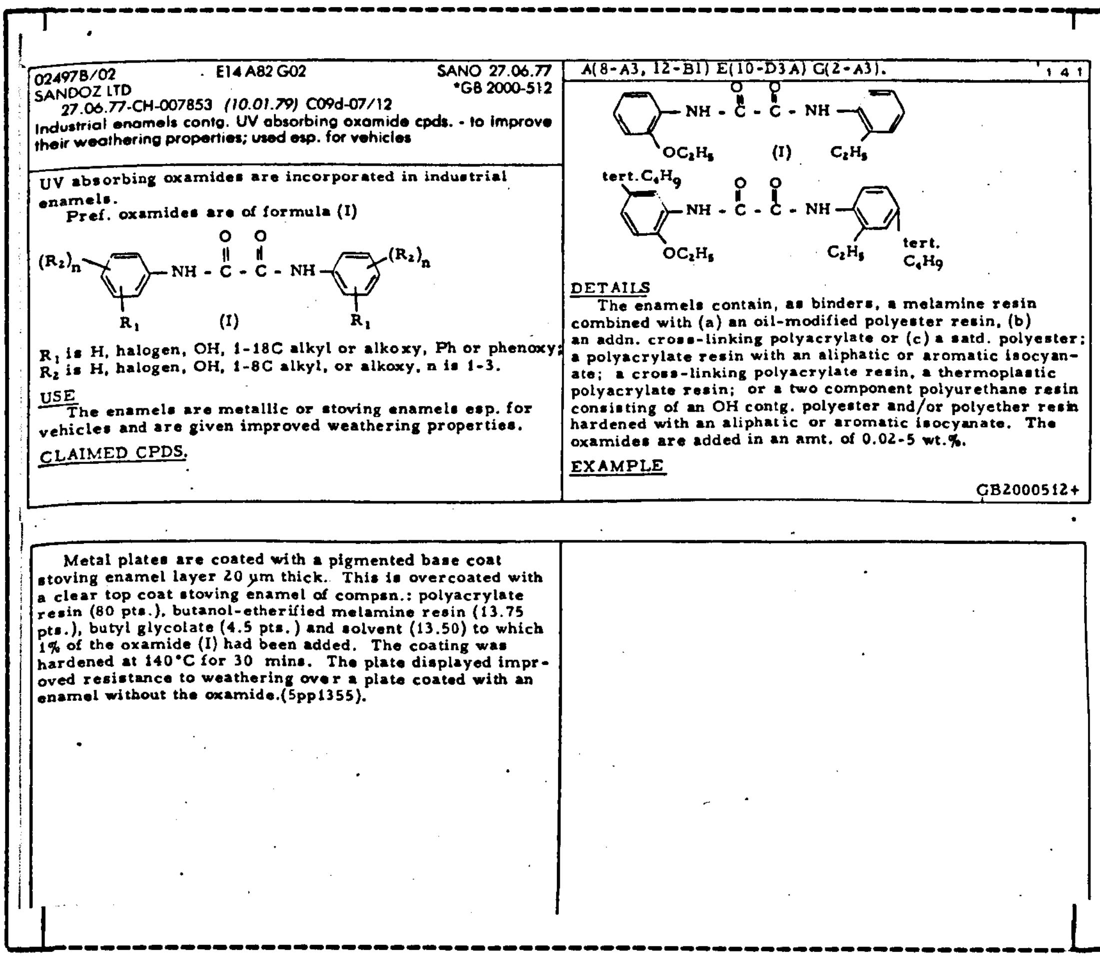
The CPI code has undergone a large number of revisions during its history, which goes back to 1963 when the FARMDOC service began. It is a manually-assigned fragment code, and was originally based on the 960 punch positions available on an 80-column punched card, the cards being sorted mechanically.[35] The code has been substantially revised over the years, and the database made available on magnetic tape as well as punched cards, and the revisions introduced in 1982 removed the restriction to punched-card format.

Each punch position, or fragment number, represents a functional group, ring system, or other feature of chemical significance, and coding is carried out manually by highly-trained and experienced encoders; there is no automatic error checking of input. The generic structure as a whole is encoded, but this involves assigning fragment numbers for all chemical features present in the generic structure, irrespective of the logical relationships between them. Thus, in effect, all the alternative specific structures covered by a generic structure are over-coded on the same representation.

Searching is carried out by combining fragment numbers with Boolean operators, and the results are characterised by high recall and low relevance, a figure of 5% for the latter being not uncommon.[26]

Whilst the improvements in the code over the past two decades have been substantial, it remains less than fully satisfactory.

Up to 1977 the Pharma Dokumentation Ring (PDR), an association of European pharmaceutical companies, found it necessary to recode the generic structures from patents in Ringcode, another fragmentation code also used for Derwent's RINGDOC and Chemical Reactions Documentation Service (CRDS) services. A semi-automatic coding system, CORA,[36] was developed for this purpose, but in 1977 the recoding was discontinued as improvements in the CPI code had meant that Ringcode no longer gave a better retrieval performance.[26]



02497B/02 E14 A82 G02 SANO 27.06.77
SANDOZ LTD *GB 2000-512
27.06.77-CH-007853 *(10.01.79)* C09d-07/12
Industrial enamels contg. UV absorbing oxamide cpds. - to improve their weathering properties; used esp. for vehicles

UV absorbing oxamides are incorporated in industrial enamels.
Pref. oxamides are of formula (I)

(I)

R_1 is H, halogen, OH, 1-18C alkyl or alkoxy, Ph or phenoxy; R_2 is H, halogen, OH, 1-8C alkyl, or alkoxy, n is 1-3.

USE
The enamels are metallic or stoving enamels esp. for vehicles and are given improved weathering properties.

CLAIMED CPDS.

A(8-A3, 12-B1) E(10-D3A) G(2-A3). 141

(I)

DETAILS
The enamels contain, as binders, a melamine resin combined with (a) an oil-modified polyester resin, (b) an addn. cross-linking polyacrylate or (c) a satd. polyester; a polyacrylate resin with an aliphatic or aromatic isocyanate; a cross-linking polyacrylate resin, a thermoplastic polyacrylate resin; or a two component polyurethane resin consisting of an OH contg. polyester and/or polyether resin hardened with an aliphatic or aromatic isocyanate. The oxamides are added in an amt. of 0.02-5 wt.%.

EXAMPLE

GB2000512+

Metal plates are coated with a pigmented base coat stoving enamel layer 20 μm thick. This is overcoated with a clear top coat stoving enamel of compsn.: polyacrylate resin (80 pts.), butanol-etherified melamine resin (13.75 pts.), butyl glycolate (4.5 pts.) and solvent (13.50) to which 1% of the oxamide (I) had been added. The coating was hardened at 140°C for 30 mins. The plate displayed improved resistance to weathering over a plate coated with an enamel without the oxamide.(5pp1355).

Figure 1.3: Derwent Basic Abstract for a patent.

In addition to their fragment-code indexing of chemical structures in patents, Derwent produce a compact and highly informative abstract of the patent, which was originally designed to appear on the back of the punched card used for coding. Where a generic structure appears in the patent, this is reproduced in the abstract, in which it is slightly reformatted to conform to Derwent's house style. Figure 1.3 shows the Derwent Basic Abstract for the British patent part of which was illustrated in Figure 1.2; other examples of Derwent abstracts appear in Figures 3.2 to 3.11 in Chapter 3.

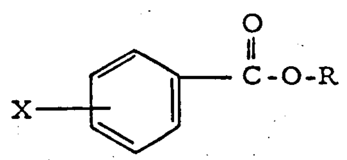
1.4.2. IFI/Plenum Data Co.

The patent documentation services provided by this American company have their origins in systems developed by a number of different organisations. The chemical coding system was developed by E.I. Du Pont de Nemours & Co. [37-39] and like Derwent's CPI code it is a manually-assigned fragment code.

Its unique aspect is that a distinction is made between fragments derived from the constant and variable parts of the structure. Figure 1.4 illustrates the assignment of such fragments for a simple generic structure, and it can be seen that those fragments deriving from either the constant or variable parts are designated **possible**, but only those deriving from the constant part are designated **must**.

In searching, the **possible** fragments are searched using positive logic, and the **must** fragments using negative logic, the latter excluding particular fragment combinations not wanted, thus improving precision.

Whilst this approach is likely to improve retrieval performance over systems such as Derwent's, which effectively use only the **possible** fragments, it does not solve the problem of indicating **possible** fragments that are mutually exclusive (e.g. halo and nitro in Figure 1.4).



(R is an alkyl group, X is a halogen or nitro group).

Possible Terms	Must Terms
ester	ester
halogen	
nitro	
carbon ring compound	carbon ring compound

Figure 1.4: Fragments in the IFI/Plenum System

In addition to the fragment descriptors, "link" and "role" indicators are used, the former linking fragments from the same structure (where there is more than one in a patent) and the latter designating the structure as reactant, product etc. Searching can be carried out using a "weighted-term" query [40] in which each query term is given a "weight", retrieved documents being those whose total score of weights exceeds a specified value.

The IFI/Plenum system, which is available online as the CLAIMS database on the Lockheed system, is restricted to United States patents, which severely limits is usefulness to patent searchers in other countries.

1.4.3. International Documentation in Chemistry

Internationale Dokumentationsgesellschaft für Chemie mbH (IDC) is a German company set up by a consortium of mainly German pharmaceutical companies, the principal members being BASF, Bayer and Hoechst. It is now part of the German National Information Centre for Chemistry.

So far as chemical structures are concerned, the core of the IDC system is the GREMAS (Genealogical REtrieval by MAgnetic tape Storage) code, originally developed at Hoechst.[41-43] This is an open-ended fragment code containing two different types of fragment, respectively called **semantic** and **syntactic** terms, and certain aspects of its design make it especially well-suited to the encoding of generic structures.[44] In fact Mullen[45] has gone as far as to claim that "the problem with Markush formulae ... [has been] solved by the GREMAS system developed by Hoechst".

The semantic terms describe the functional groups present by means of three-letter codes, in which each successive letter indicates more precisely the nature of the group. Figure 1.5 shows some examples of the letters used to represent some common functional groups.

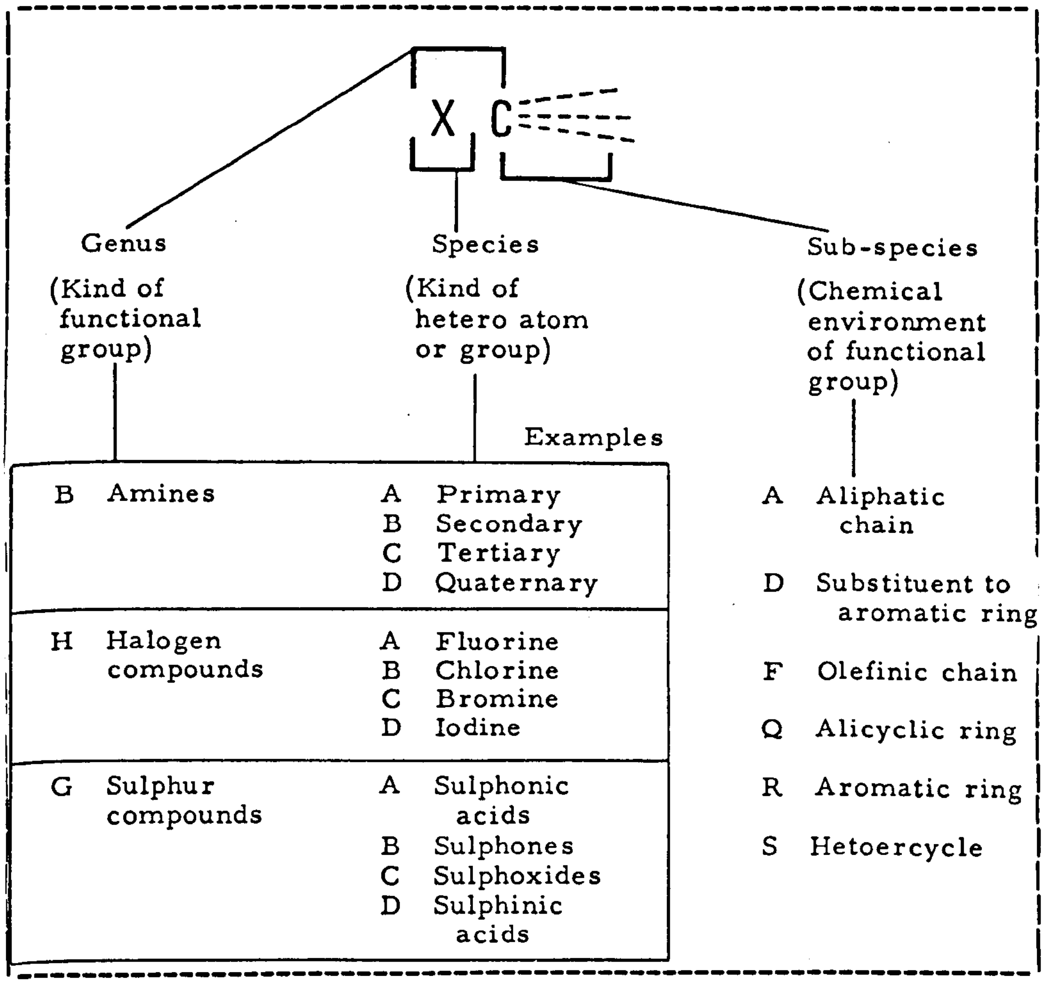


Figure 1.5: Some GREMAS semantic terms (from [27])

For generic structures, the numeral 0 can be used to give a generalised fragment, e.g. H0R represents a halogen substituent on an aromatic ring, but does not specify the particular halogen.

The **syntactic** terms in the code indicate the relationships between the semantic terms, and they normally begin with a Y. Each one represents one of four **regions** of the structure, and this is indicated by the second letter: YR... for carbon chains, YS... for alicycles, YT... for aromatic rings and YU... for heterocycles. These two letters are followed by the initial letters from the semantic terms represented in the **region** in question, with the result that these terms can be of any length. Numeric locants can also be used to indicate substitution patterns on rings.

In generic structures, where there is a list of alternatives for a structural variable, the appropriate initial letters of the semantic terms are shown all together in the syntactic term, following a slash, which indicates that only one of them may be present. An example of this appears in Figure 1.6.

Specific structures can be encoded automatically in GREMAS terms, but generic structures are encoded manually.

The use of an open-ended code of syntactic descriptors in the GREMAS system, able to handle alternatives in a generic structure, makes GREMAS far more effective for storage and retrieval of generic structures than other fragment-based systems. It has, however, severe limitations in that a maximum of nine alternatives can be catered for in each of a maximum of three structural variables; its ability to handle generic expressions such as "alkyl" is also slight.

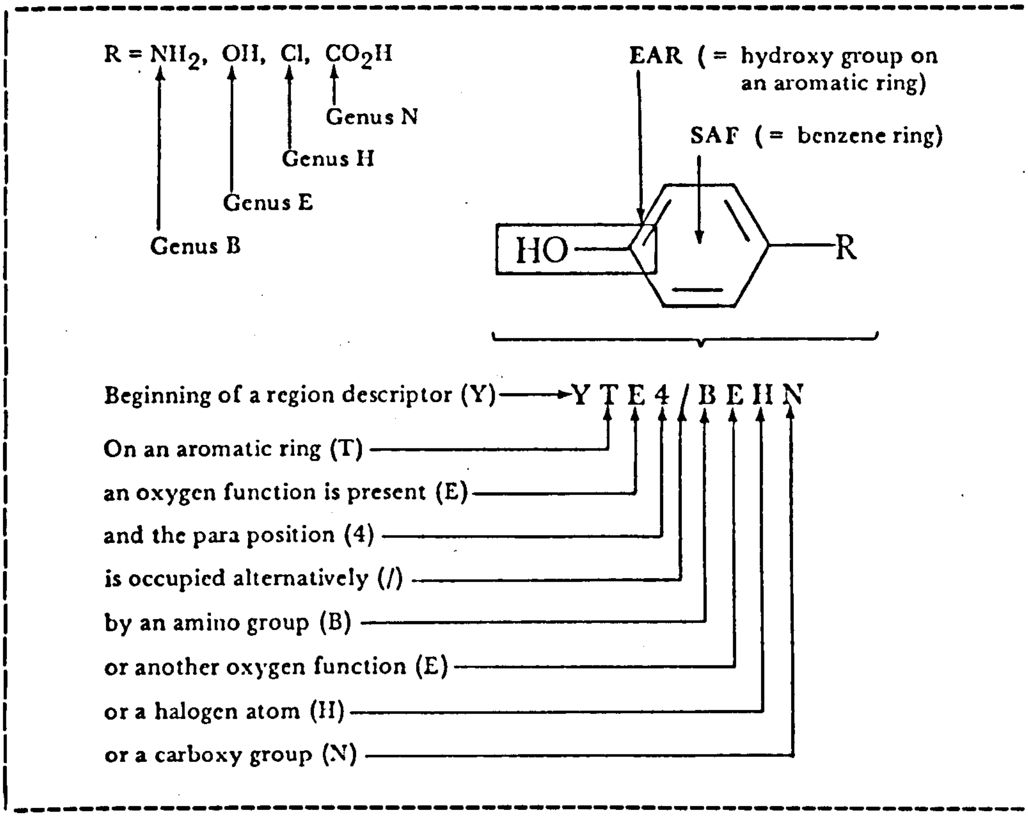


Figure 1.6: GREMAS coding for a generic structure (from [27])

Silk [29] has pointed out that the GREMAS code possesses a far more precise search capability than other fragment codes in use, even for specific structures, since it is able to deal with specified positions of substitution on rings or chains, and to distinguish between substituents on different ring systems. The inclusion of specific structures from the Chemical Abstracts Registry file in the IDC database, along with generic structures from patents, also gives it an edge of rival systems for many types of enquiry. However, Silk also notes that the system is extremely expensive, and suggests that there could be many problems in mounting it online.

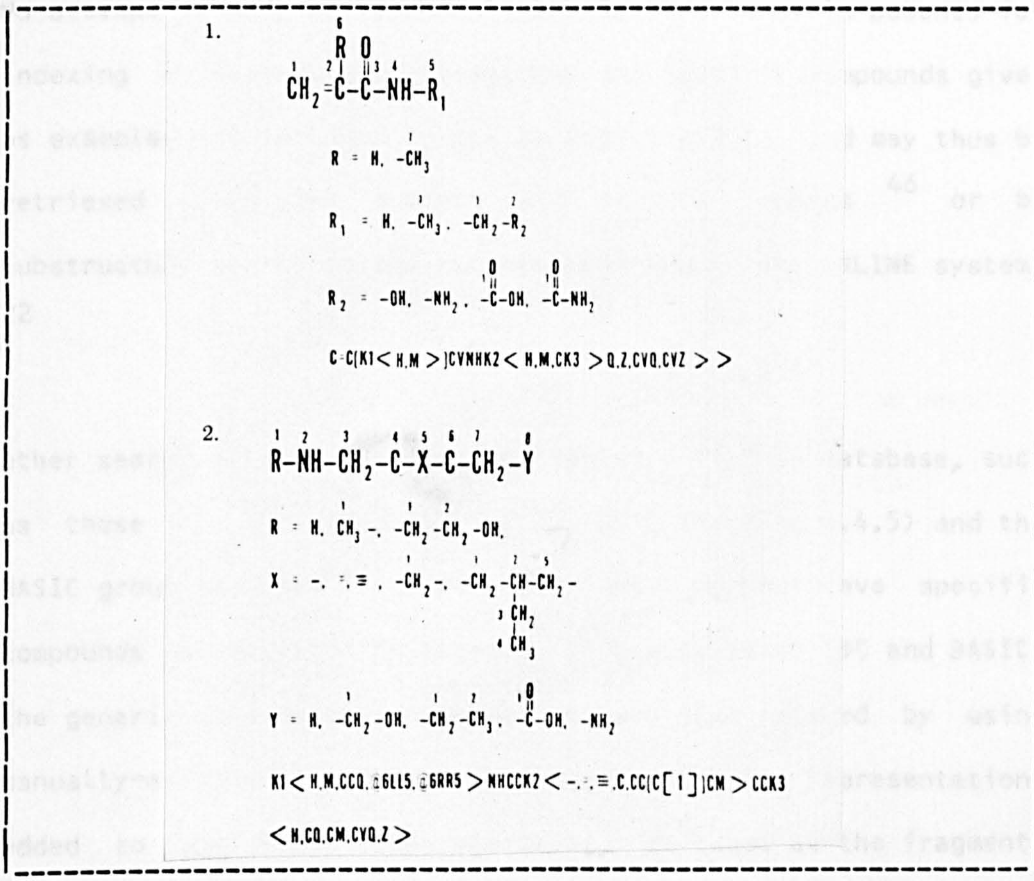


Figure 1.7: Hayward Notations for generic structures[14]

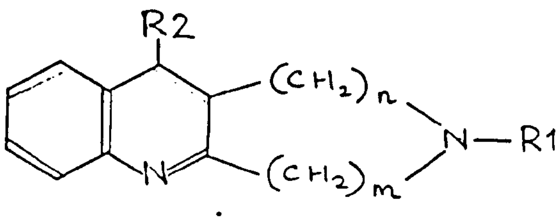
1.4.4. Chemical Abstracts Service

Abstracts of chemical patents appear in Chemical Abstracts (CA), which also includes concordances relating basic and equivalent patents from different countries. Though in the past the coverage of patents was not as comprehensive as in other services, such as Derwent's,[3] it has improved recently, and CA now abstracts more than 70 000 patents annually, of which 45% are from Japan alone[1] resulting in serious translation difficulties.

No attempt is made to represent generic structures in patents for indexing or searching purposes, but any specific compounds given as examples are included in the CA Registry file, and may thus be retrieved using the subject and formula indexes [46] or by substructure search on the recently-introduced CAS ONLINE system. [22]

Other search systems using the CA Registry file as database, such as those of IDC (Section 1.4.3), DARC (Section 1.4.5) and the BASIC group in Basel [47] are thus able to retrieve specific compounds exemplified in patents. In the cases of IDC and BASIC, the generic structures in the patents are also indexed by using manually-assigned fragment descriptors, and these representations added to the files for searching, at least at the fragment-matching level.

Whilst the effectiveness of such systems clearly depends on the relationship between the generic structure in a patent, and the specific compounds exemplified in it, [48] a group of searchers from ICI have suggested that even with this limitation, retrieval performance is at least comparable with that achieved by searching in a database such as Derwent's or IDC's where the generic structure is indexed by manually-assigned fragment codes. [49]



R1 IS H, 1-3C ALKYL (OPT. SUBST. BY CN OR PHENYL)

2-4C ALKOXY CARBONYL, 1-3C ACYL (OPT. SUBST. BY OCH3) OR PHENYL.

R2 IS OH, 1-3C ALKYL OR PHENYL (OPT. SUBST. BY HALOGEN OR CH3)

N, M. = 1, 2, or 3

N + M 4

PROPOSED WLN

T C66-N1-BN-X1-NJ I-R1-X1-R2- •R1=Q
-ALKYL (1-3C)-, R, R X J, R X1; •R2=H, -ALKYL (1-3C)-,
-ALKYL (1-3C)-XCN, -ALKYL (1-3C) - XR, VO-ALKYL (2-4C)
-ACYL (1-3C), - ACYL (1-3C) - XO1; •N1 (X1) = 5 (L),
6L, M), 7 (L, M, N).

Figure 1.8: WLN representation for a generic structure[26]

1.4.5. Système DARC

Système DARC (Description, Acquisition, Retrieval, Correlation) is a chemical substructure search system developed at the University of Paris by Dubois and others,[23-25] and it is available online through the French host system, Telesystèmes.

At present it permits substructure searching in the CA Registry file, and thus gives access to the specific compounds exemplified in patents. Bois and Chaumier [50] have compared DARC's performance to that of IDC (Section 1.4.3), and noted the latter's better coverage of patents.

Facilities for generic query structures are shortly to be implemented, though these will still only permit searching in files of specific structures. However, some comments on storage and retrieval of generic structures have appeared in publications on the DARC system [51-52] even if limited to the statement that "the treatment of Markush formulae has been studied in Paris by Professor Dubois", and it was recently claimed that a full generic structure search system would be ready in 1984. [53] No information has been forthcoming as to its capabilities or method of operation.

1.4.6. Line Notations

Up until a few years ago, line notations predominated as a means of unambiguously representing specific chemical structures for machine processing, and so it was to be expected that investigations should be made into the possibility of extending such notations to handle generic structures also.

The first attempt of this sort was by the late G.M. Dyson [54] who showed how generic groups such as "alkyl" could be encoded in a modification of his own IUPAC notation, along with lists of specified alternatives for structural variables. A problem area he identified was that of the definition of expressions like "cyclic carbon compound", and he suggested that a data bank might be maintained with standard notations for such expressions. This problem has also been encountered in the present work, and is discussed in Section 5.7.3.

In the mid to late 1960's work was carried out at the U.S. Patent Office on the encoding of generic structures in a form of Hayward Notation,[55, 14] though it was only applicable to certain types of generic structure (the so-called **determinate** structures identified by Sneed <u>et al.</u>[14] and referred to in Section 1.1.2). Figure 1.7 shows examples of the notations that resulted. Associated with this was work on search algorithms for generic structures, using a connection table representation.[56-58]

The dominant position of Wiswesser Line Notation (WLN)[20] in specific structure systems led to a number of efforts by the British software house Fraser Williams (Scientific Systems) Ltd[59] and others to adapt it for generic structures; an example of the rather unwieldy notations which resulted is shown in Figure 1.8.

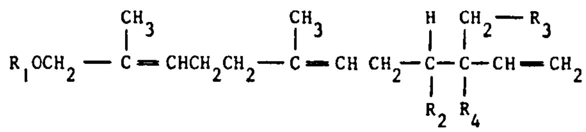
A more promising approach was suggested by Krishnamurthy and Lynch[15-16] and is based on Krishnamurthy's own "ALgorithmic WIswesser Notation" (ALWIN)[60-62] which is a modification of the original WLN and is designed to be amenable to automatic generation.

A unique feature of this approach is the use of formal grammars for the representation of members of radical classes such as "alkyl". Figure 1.9 illustrates an ALWIN-based notation for a generic structure, with the associated grammar production rules.

None of these notation-based suggestions has been implemented, despite the potential advantages (discussed in Section 1.5 below) of an unambiguous representation of the generic structure. There are a number of reasons for this. In the first place the notations that result from even quite simple generic structures are generally-speaking horrendous, and a system using them could hardly be described as "user-friendly".

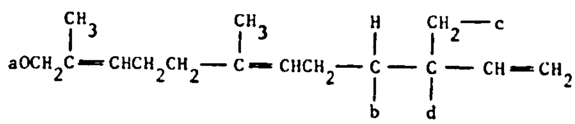
Secondly, many of the existing rules in line notations are designed to produce a canonical notation for a given specific structure. It is difficult to see what purpose would be served by a canonical (as opposed to merely unambiguous) notation for generic structures, whereas to ignore the canonicalisation rules altogether would result in widely-differing notations for quite similar structures.

Furthermore, the fact that many notations (WLN and ALWIN in particular) emphasise ring systems would lead to great difficulty in structures with optional rings, or with rings of variable size, on account of problems in assigning locants for substitution positions etc. Finally, the use of a line notation for generic structure representation might severely restrict the options available for generation of fragments for a first-level screening search; it is likely that such fragments would have to be closely related to the symbols used in the notation to represent functional groups etc.



$R_1 = CH_3$ or C_2H_5 and R_2 and R_3 are different and each is hydrogen or together with R_4 is an additional bond between the carbon atoms carrying R_2 and R_4 or R_3 and R_4 respectively.

Let $R_1 = a$, $R_2 = b$, $R_3 = c$ and $R_4 = d$.



Notation: a01TAD11TAD1Y*b*X1*cd*D1

Production rules

$a \rightarrow A \mid 1A$		$a \rightarrow A \mid 1A$
$b, d \rightarrow \phi$	or	$b \rightarrow H$
$c \rightarrow H$		$C, D \rightarrow \phi$
Y*b*X1*cd* → DTA		Y*b*X1*cd* → 1T1

Figure 1.9: ALWIN representation for a generic structure

1.4.7. The COUSIN system

Howe and Hagadone have developed an online structure storage and retrieval system at the Upjohn Company in Michigan.[63-64] Called COUSIN (CompOUnd Search INformation system), the system is particularly interesting on account of the extensive facilities it provides for generic query structures, though it uses a database of specific structures.

Generic queries may be input using a special notation, the R_k notation, which allows R groups to be introduced in the structure diagram for the constant part of the structure, and subsequently defined. Figure 1.10 illustrates a generic structure in R_k notation, and it can be seen that the system requires every possible attachment position for each R-group variable to be indicated in the diagram. In the definition of the R-group, each possible value is followed by the number of times it can occur in the specified positions.

The query validation program is able to check that there are no inconsistencies in the information given, and to calculate multiplicities where the user has simply specified "rest".

From the R_k notation input, the system is able to form a connection table based internal representation of the query, which is used in searching, though details of this have yet to be published.

COUSIN is not intended for use outside Upjohn, and the hardware configuration it runs on would make it extremely difficult to transport, but it is probably the most sophisticated computer representation for generic structures (albeit only query structures) currently in operation.

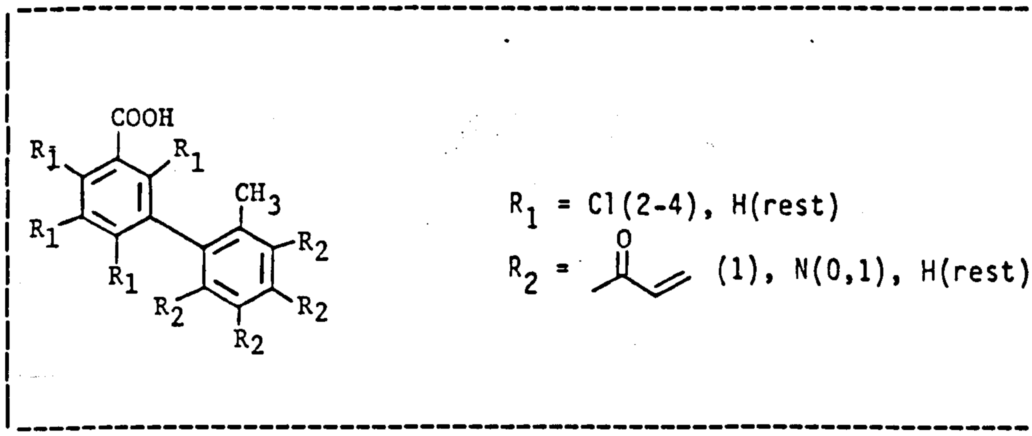


Figure 1.10: R_k notation for a generic structure[64]

1.5. REQUIREMENTS FOR A SEARCH SYSTEM REPRESENTATION

The various forms of generic structure representation described in the last section are all unsatisfactory for one reason or another, and the work described in this Thesis has had as its aim the development of a more effective representation, allowing a closer approach to Jackson's concept of an "ideal" generic structure information system (Section 1.3). This has led to the idea of a number of different representations for use at different stages of such a system.

The conventional arrangement of storage and retrieval systems for specific structures has involved a number of stages in a search. Lynch [65] has discussed the need for a first-level "screening" search to remove from consideration those structures in the database which, by virtue of their lack of some feature present in the query, cannot possibly satisfy the query. When the file to be searched has ben thus reduced, computationally more expensive procedures can be used to search those structures which remain candidates. A variety of different "screens" have been used for first-level searching, including molecular formulae and various fragment-code representations, often implemented as bit-screens.

The present work has envisaged an analogous approach to generic structure searching, and Figure 1.11 illustrates the overall process intended. An input notation, called GENSAL (GENeric Structure LAnguage), has been designed for the unambiguous description of generic structures in a form which is intelligible to a chemist or patent agent, yet sufficiently well formalised to permit automatic analysis by computer. GENSAL is intended to be the representation used for input both of file structures from patents and of query structures, and it is described in Chapter 3. It is a formal language, analogous to a computer programming language, and Chapter 2 reviews briefly the theory of such languages.

The GENSAL representation input to the computer will be used to generate an internal representation of the structure, and this is described in Chapter 4. It is based on connection tables [66] and

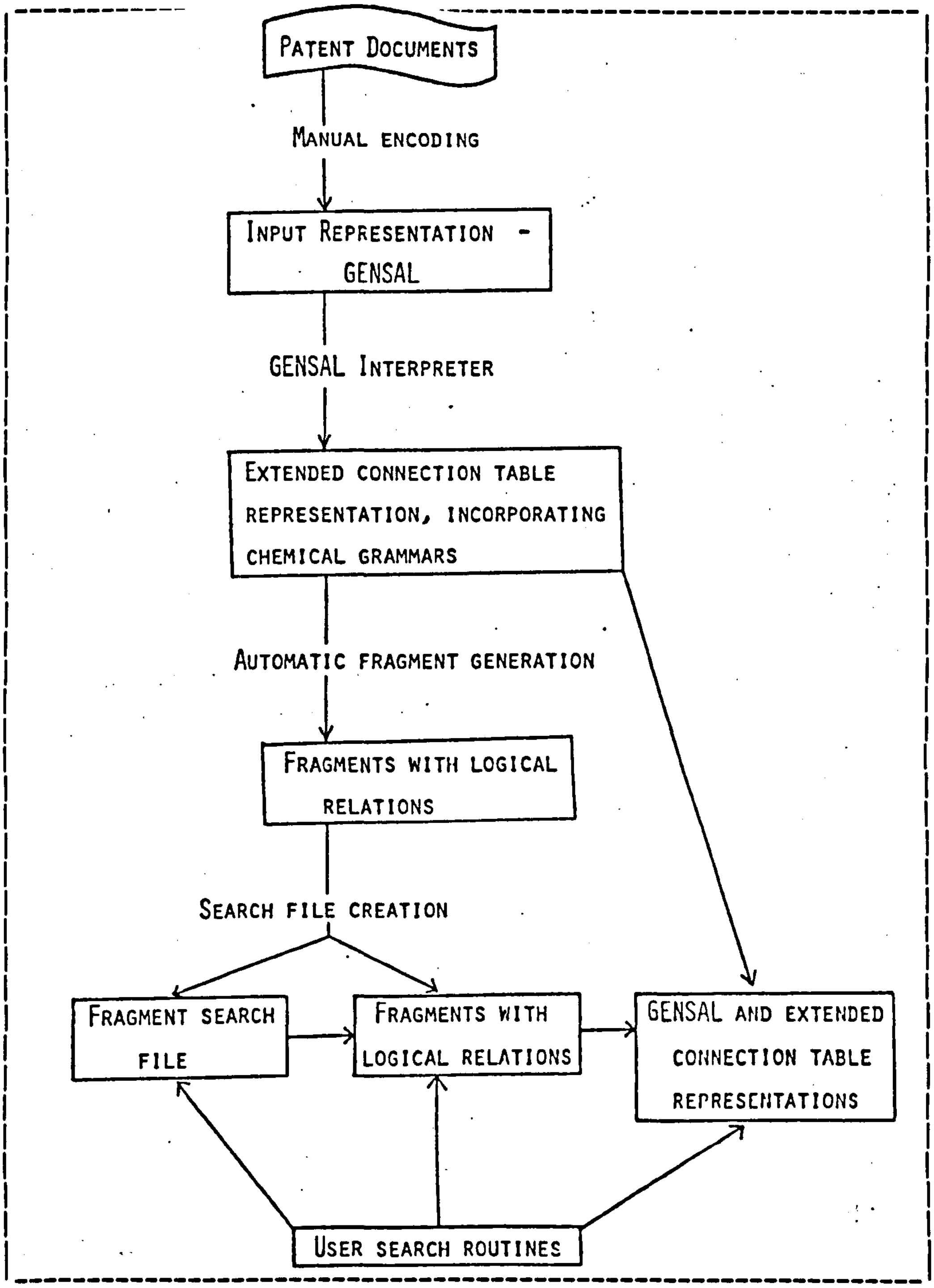


Figure 1.11: Overall process for generic structure system

like GENSAL is an unambiguous representation of the generic structure; it is intended to be transparent to the user. The interpreter program which performs the conversion from GENSAL to the internal representation is described in Chapter 5.

The internal representation is envisaged as the basis for searching. Ultimately, it should be possible to perform an atom-by-atom match between query and file structures in the internal representation, but such a match is likely to be extremely expensive computationally - much more so than in specific structure search systems, on account of the possibility of alternatives at various points.

Thus it is expected that there will probably be at least two fragment-based screening searches to reduce the file of candidate database structures. A number of different types of fragment can be generated from the internal representation, and algorithms for such fragment generation, and the use of fragments in different search representations are discussed by Welford.[67]

CHAPTER 2

FORMAL LANGUAGES

"I conceive you may use any language you choose
to indulge in without impropriety."

W.S. Gilbert

The mathematical theory of languages has been extensively developed over the past quarter of a century, and has been the subject of several textbooks. [68-73] This Chapter gives an outline of those aspects of the theory of formal languages, and the means of parsing them, which have been built upon in the design of the GENSAL language described in Chapter 3, and in the programming of its interpreter, described in Chapter 5. It will do this with particular reference to computer programming languages, which are the most commonly encountered class of artificial formal languages, and in this context will discuss the choice of programming language for the software development described in Chapter 5. The Chapter also considers the use of artificial formal languages in information work, and in particular in chemical information.

No attempt will be made to give a comprehensive review of the subject of formal language theory, as many excellent such reviews exist, and will be referred to, and as far as possible the more mathematical aspects of the area will be avoided.

2.1. DEFINITION AND CLASSIFICATION OF FORMAL LANGUAGES

The earliest work on the mathematical theory of languages, which was done in the late 1950's, is largely due to Noam Chomsky,[74-78] who was attempting to find a means of modelling natural languages such as English. His aim was to understand the mechanism by which it is possible to comprehend sentences never heard before, and to produce completely novel, but grammatically correct, sentences.

For the purpose of his analysis Chomsky considered a language as being a set (finite or infinite) of sentences, each finite in length and constructed by concatenation out of a finite set of elements. These elements are termed the "terminal symbols" of the language, and might, in the case of English, be identified with the set of valid English words.

The grammar of a language he considered as a means for generating sentences in such a language. The grammar will generate all possible grammatically correct sentences in the language, but no others. It specifies the symbols of language, and includes a set of rules, sometimes called "productions", or "rewriting rules" which specify the replacement of one group of symbols by another during the generation of a sentence: a grammatically correct, or "well-formed" sentence in a language is one that can be generated by the grammar.

Whilst a given grammar is only able to generate sentences in a single language, several different grammars may all generate the same language - such grammars are said to be <u>equivalent</u>.

Put more mathematically, a grammar G may be represented as a "4-tuple":

$$G = (V_N, V_T, P, S)$$

V_N is the set of "non-terminal symbols" or "variables" (descriptive terms or "metasymbols" representing elements of the sentence), and V_T is the set of "terminal symbols". Both V_N and V_T are called "alphabets", and they are disjoint. Their union is symbolised V.

Strings, or "sentences", can be constructed over an alphabet, and consist of concatenated sequences of elements of the alphabet, of arbitrary length. The set of sentences over an alphabet V is symbolised V^*, and may include the null string (which is of zero length). The set of sentences over V, excluding the null string, is symbolised V^+.

P is a set of "productions" or "replacement rules", which are of the form

$$\alpha \dashrightarrow \beta$$

where α is a string in V^+ and β a string in V^*.

If a production in P can be used to rewrite a string α_1 as another string α_2 then it is said that α_1 <u>directly derives</u> α_2 in grammar G. If the application of a series of productions in P enable α_1 to be rewritten as α_m then it is said that α_1 <u>derives</u> α_m in grammar G.

The grammar G is said to generate a language L(G), which consists of the set of sentences over V_T (i.e. elements of V_T^*). However, only certain of the sentences in V_T^* are grammatically correct ("well-formed"), and these are those of them that can be <u>derived</u> from S (which is a distinguished member of V_N called the "sentence symbol" or "start symbol") in grammar G.

2.1.1. The Chomsky Hierarchy

Grammars, and hence the languages which they generate, have been classified by Chomsky [74] by imposing successively tighter restrictions on the form of the production rules in P. The above grammar, in which no restrictions are imposed is called a Type 0, or unrestricted grammar, and the languages it generates are called the recursively enumerable languages.

A Type 1 grammar is obtained if it is required that the number of symbols on the right hand side of each production should be greater than or equal to the number of symbols on the left hand side. An alternative, and equivalent restriction is that the rules in P should be of the form

$$\alpha_1 A \alpha_2 \dashrightarrow \alpha_1 \beta \alpha_2$$

where α_1 and α_2 are in V^*, β is in V^+, and A is in V_N. This form of the restriction leads to the name context sensitive for this type of grammar, as it allows A to be replaced by β when it occurs in the context of α_1 and α_2.

In Type 2 grammars, the left hand side of the production must be a single non-terminal symbol. The productions are therefore of the form

$$A \dashrightarrow \beta$$

where A is an element of V_N, and β a string in V^*. Since A may be replaced by β independently of the context in which it occurs, this type of grammar is called context free.

The most restricted type of grammar, Type 3, requires that all productions are of the form

A ---> a B

or

A ---> B

where A and B are members of V_N and a is a member of V_T. Type 3 grammars are called regular grammars.

It is clear that these increasingly severe restrictions on the form of the productions mean that the types of grammar are arranged in a hierarchy: every Type 3 grammar is also Type 2, every Type 2 grammar is Type 1, and every Type 1 grammar is Type 0. This is sometimes referred to as the Chomsky hierarchy.

Many important and interesting properties can be shown for all these grammar types,[68-70, 72, 74, 77, 79-81] but detailed coverage of them is beyond the scope of this Thesis.

2.2. PARSING OF CONTEXT-FREE LANGUAGES

The relative simplicity of context-free grammars has allowed considerable progress to be made in the automatic syntactic analysis (parsing) of sentences in the languages generated by them, whereas such analysis has proved highly intractable for context-sensitive and unrestricted grammars.

Unfortunately, despite initial hopes, context-free grammars have not proved adequate for the description of natural languages, but they have been extremely useful in the definition of artificial languages, in particular, programming languages.

```
            S
           / \
          /   \
         A     D
        / \     \
       /   \     \
      a     C
           / \
          /   \
         B     E
        /|\     \
       / | \     \
      b  c  d     e
```

Figure 2.1: Derivation tree for the sentence abcde in $L(G_1)$.

A sentence in a context-free language can be analysed in terms of a grammar which generates it using a <u>derivation tree</u> or <u>parse diagram</u> in which the root vertex of the tree is S, its leaves are all elements of V_T, and its interior vertices are all elements of

V_N. Consider the language $L(G_1)$ generated by grammar $G_1 = (V_T, V_N, P, S)$ where

V_T = { a, b, c, d, e }

V_N = { A, B, C, D, E }

P = { S --> AD

A --> aC

B --> bcd

C --> BE

D -->

E --> e }

The derivation tree for the sentence abcde is shown in Figure 2.1.

It will be seen that the symbols of the sentence can be followed around the leaves of the tree from left to right, and that each branch of the tree corresponds to an appropriate production in P.

If a sentence has more than one derivation tree, then it is <u>ambiguous</u> -- grammars which generate only sentences with unique derivation trees are unambiguous grammars, whereas other grammars are ambiguous. Since it is likely to be essential for an artificial language to have an unambiguous grammar, this point is very important.

Even when there is a unique derivation tree, it is possible to obtain it by applying the productions in different sequences. The sequence may, however, be standardised by stipulating that at each derivation the rightmost non-terminal symbol should be replaced in accordance with an appropriate production. The derivation sequence corresponding to Figure 2.1 is therefore:

S --> AD --> A --> aC --> aBE --> aBe --> abcde

2.2.1. LR Parsing

In parsing a sentence, it is necessary to start from the string of terminal symbols (i.e. the leaves of the tree), and to reconstruct the derivation tree, leading eventually back to S, its root. This is a process of successively reducing substrings of terminal and non-terminal symbols in accordance with the productions.

At each step in the parse, the derivation required is, according to the above stipulation, one in which the rightmost non-terminal symbol is replaced. This corresponds, in the parsing direction, to reducing the leftmost set of adjacent leaves of the tree (which will only all be in V_T at the start of the parse) that form a complete branch. Since the parsing thus operates from left to right along the original string of terminal symbols, it is called a left-right parse.

Knuth [82] has defined a subclass of context-free grammars, called LR(k) grammars, for which this parsing method will work, looking ahead a maximum of k symbols to identify with certainty each production in the derivation. It may also be shown that all LR(k) grammars are unambiguous, and that there is an algorithm to determine whether or not a context-free grammar is LR(k) for a given k.

The derivation tree is reconstructed from the bottom upwards, and LR(k) grammars are therefore sometimes called bottom-up grammars, and the parsing method bottom-up parsing.

2.2.2. LL Parsing

Lewis and Stearns [83] have defined another subclass of context-free grammars, the LL(k) grammars, which allow an even simpler approach to the parsing of sentences. Each production in the derivation can be identified with certainty by inspecting the sentence from its beginning (left) end to the k-th symbol beyond the beginning of the production.

In this type of parse, the derivation tree is being reconstructed from the top downwards, and hence LL(k) grammars are called top-down grammars, and the parsing method top-down parsing, or parsing by recursive descent.

At the start of a parse based on an LL(k) grammar, it is assumed that a production having S as its left-hand side is required. Which of the various productions in P having S as left-hand side is appropriate can, for an LL(k) grammar, be determined by looking at a maximum of k symbols.

Rosenkrantz and Stearns [84] have shown that it is possible to determine if a grammar is LL(k) for a given k, that all LL(k) grammars are unambiguous, and that the LL(k) grammars are a subset of the LR(k) grammars. In addition they have shown that provided there are no productions with an empty string as right-hand side, it is possible to construct for a language generated by an LL(k) grammar an equivalent LL(k) grammar in Greibach Normal Form (i.e. where the right-hand side of each production starts with a terminal symbol). [85] Furthermore, if every production with a given non-terminal as its left-hand side has a different terminal as the first symbol on its right-hand side, then the grammar is LL(1), and is a member of the class of simple deterministic grammars described by Koranjak and Hopcroft. [86] No look-ahead is required in the parsing of sentences in languages generated by such grammars, and the production involved can be determined at each step simply by examining the next symbol in the sentence. Some properties of deterministic context-free languages have been discussed by Ginsburg and Greibach. [79]

2.2.3. Top-Down vs. Bottom-Up Parsing

The properties of LL(1) and LR(1) grammars have recently been compared by Beatty [87] and Knuth [88] has compared the particular advantages of top-down and bottom up parsers. Because all LL(k) grammars are also LR(k), any language that can be parsed top-down can also be parsed bottom-up, but the reverse is not true, making bottom-up parsing more generally applicable. The chief advantage of top-down analysis is that it is known which production is being used after examining only k terminal symbols, and this enables some degree of prediction on the part of the parser as to which symbols will be encountered next. This is especially helpful for the semantic analysis of the sentence, and the design of modern programming languages has taken particular note of the advantages of LL(k) grammars with a low value for k.

More detailed discussion of parsing methods for context-free languages may be found in a number of textbooks and reviews. [71, 80, 81, 89]

2.3. PROGRAMMING LANGUAGES

The earliest developments of high-level computer programing languages took place in isolation from the work of Chomsky and others on formal languages, and a comprehensive survey of their history has been given by Sammet. [90] Fortran was the first high-level language to gain wide acceptance, and it is still the the most commonly-used language for scientific applications.

Attempts have been made to formalise the grammar of Fortran [91] but on account of the rigid field format for its statements, and the numerous minor restrictions on various constructs, these have been of limited success. The language was designed for speed of execution, rather than simplicity of syntax analysis.

The ALGOrithmic Language Algol 60 was designed by an international committee in the late 1950's and early 1960's [92, 93] and marked a turning point in programming language design. Its importance lies more in the manner of its definition, which has had a major influence on the design and definition of more recent languages including Algol 68, [94] Pascal [95] and Ada, [96] than in its actual use, which has been comparatively limited, at least so far as computer implementations are concerned, though it is the standard publication language for algorithms.

2.3.1. Syntax Specification

The original definition of Algol 60 [92] first introduced the so-called Backus-Naur metalanguage for the formal specification of its syntax. An example of a grammatical rule of Algol 60 expression in Backus-Naur Form (BNF) [97] is

<conditional statement> ::= **if** <boolean expression> **then** <statement> **else** <statement> | **if** <boolean expression> **then** <statement>

This defines the syntactic category "conditional statement" as being one of two alternatives: either the word "**if**" followed by a "boolean expression" followed by the word "**then**" followed by a "statement" followed by the word "**else**" followed by another "statement", or alternatively the word "**if**" followed by a "boolean expression", followed by the word "**then**" followed by a "statement". The syntactic categories "boolean expression" and "statement" are defined by other rules in the grammar.

The grammatical rules in a BNF grammar are expressed in a "metalanguage", which uses certain symbols that do not occur in the language being defined. The symbol ::= means "is defined to be", and | means "or". Angle brackets are used to enclose the names of syntactic categories, which thus themselves form symbols of the metalanguage. The words not so enclosed (**if**, **then**, etc.) are of course, the actual symbols of Algol 60.

In 1962 Ginsburg and Rice [98] proved that Algol-like languages defined using a BNF metalanguage are equivalent to the context-free languages (Type 2) defined by Chomsky which allowed the rigorous mathematical properties of context-free languages to be applied to Algol and other programming languages. In the BNF metalanguage, the syntactic categories can be identified with the non-terminal symbols of Chomsky Type 2 grammars and the actual symbols of the language being defined with the terminal symbols. For a programming language such as Algol 60, the start symbol is identified with the syntactic category "program". The various alternatives separated by the | symbol, correspond to the different productions having the same left-hand side.

The only difference between grammar specification using BNF or a 4-tuple is in the form of its representation. Other methods for syntax specification have also been suggested, including a tabular format [99] and the use of "syntax diagrams". [100] These latter represent the BNF rules in diagrammatic form, with separate branches for each alternative, and it is normally possible to combine several BNF rules into a single diagram. Wirth [101] has also proposed a extended BNF formalism.

2.3.2. Syntactic Analysis

The purpose of a high-level programming language is to allow a programmer to give instructions to a computer in a form which remains reasonably intelligible to himself, or to another programmer. Before the computer can actually execute the instructions, however, it must convert them into a form more closely related to its own internal architecture, and this process of conversion is called <u>compilation</u>. [80, 81] Three principal operations are involved in compilation: lexical analysis (in which the string of characters forming the program in the high level <u>source</u> language is split up to identify the separate <u>tokens</u> or terminal symbols of the language), syntax analysis (in which the grammatical relationships between the tokens are identified, in accordance with the rules of the grammar) and code generation (in which the machine level <u>object</u> language is generated).

The process of syntax analysis is based on the same principles of parsing as are described in general terms in Section 2.2 above, and is obviously much simpler for a programming language with an appropriately simple grammar. In the design of Algol 60 and the languages based on it particular attention has been paid to the need for simplicity of syntax analysis. Not only does this simplify the complexity of the program required to perform the syntax analysis, but a simple grammar also makes it much easier for the programmer to write elegant and error-free programs.

Irons [102] described a bottom-up syntax analyser for Algol 60 in 1961, but the first top-down syntax analyzer for a programming language was written for Cobol, and described by Conway [103] in 1963. Numerous textbooks and reviews consider the problems of compiler writing and syntax analysis for a variety of programming languages. [71, 80, 81, 104-110]

2.3.3. The Pascal Language

Pascal is a high level language based on Algol 60, and was designed by Nicklaus Wirth, who published the first description of it in 1970, [111] with a revised version in 1975. [95] A committee of the International Standards Organisation convened by A.M. Addyman has drafted a Standard definition for the language, which has been published for comments. [112, 113] The language has become extremely popular, particularly in academic circles, and has been the subject of many textbooks. [114-117]

Pascal was designed especially for compilation using top-down (or "recursive descent") syntax analysis, and Wirth described the first compiler in 1971.[118] The first version of this was written in Pascal itself, and manually translated into a lower level language. Each subsequent version of the compiler could then also be written in Pascal, and compiled by the previous version, a procedure known as "boot-strapping".

Pascal has been enthusiastically promoted by many authors[119-122] and possibly partly as a result of this has also attracted considerable criticism,[123-128] some of it quite vitriolic.[123-126] Other authors, whilst generally welcoming the language, have made suggestions for its enhancement,[129, 130] and Wirth himself has published his own retrospective assessment.[131]

In his paper, Wirth discusses the advantages Pascal has for the writing of <u>reliable</u> software, as the design of Pascal permits a great deal of checking on the self-consistency of the program to be done by the compiler, and thus a high proportion of program errors can be detected before execution begins. Its highly structured design also makes its suitable as a teaching language, and this has been its principal area of application to date. Conradi[124] has however pointed out some of its disadvantages as a systems programming language, particularly with its lack of flexibility in matters such as the absence of dynamic arrays (Pascal, unlike for example Algol 68, requires that array bounds be specified at compile time), though Wirth's paper points out that it is precisely these limitations on flexibility that give

Pascal its enhanced security.

Despite its acknowledged weaknesses, Welsh, Sneeringer and Hoare have expressed

> "the belief that Pascal is at the present time the best language in the public domain for the purposes of systems programming and software implementation". [126]

2.3.4. The Ada Language

Pascal was used as the basis for all the tenders to the U.S. Department of Defence for the design of a new programming language to be used for all their software development. [132] However, the selected language, Ada, [96] has led to even fiercer controversy than Pascal. [133-136] Much of the criticism has attacked the increased flexibility of Ada over Pascal, with many additional features not present in the older language, which, it critics claim, make it less secure, and programs written in it unreliable. In view of the likely military applications of Ada, echoes of this discussion have reached a public forum. [137]

2.3.5. Choice of Language for Software Development

In choosing a programming language for the practical work described in Chapter 5 of this Thesis a number of factors were considered. A modern, structured language was required, with good program readability and portability, since the work is of substantial interest to the chemical and patent documentation industries. In addition, the programs required would operate interactively, and would therefore need to be developed on the Sheffield University Prime computer system, which restricted the choice of language to those for which Prime compilers were available.

Whilst Fortran would have provided the greatest portability, it was felt that it was insufficiently well-structured, the same reservation applying to Basic. Implementations of both Algol 68 and Pascal were available, but only the latter was actively supported by Computing Services staff, and was therefore the language chosen.

Initially a compiler developed at the University of Hull was used, but it was later replaced by a much more powerful one written by staff of Sheffield University Computing Services,[138] which generates segmented object code, and allows much bigger programs and easier interface with routines in other languages, and contains facilities for separate compilation of Procedures and Functions. It was also found that the easy availability of the compiler's writers was extremely useful on encountering problems in software development; none of these advantages would have been available with Algol 68.

Nevertheless, a number of disadvantages were encountered with Pascal, of which the most serious was in the use of external files, particularly as Pascal does not permit programs to append data to files that already exist, and neither does it implement direct-access files.

2.4. FORMAL LANGUAGE SEMANTICS

Chomsky [78] has pointed out that there may be sentences in a language which, whilst being grammatically correct, make no sense. An English example he gives is the sentence

"Colourless green ideas sleep furiously."

Similar problems may be encountered in programming and other formal languages, and though methods for specifying the syntax of a language (at least for certain classes of language) are now well-established, comparatively little success has so far been achieved in formally specifying the semantics of languages.

Several approaches have been used, and have been reviewed by a number of authors. [139-141] Hoare and Wirth have attempted [142] to define the semantics of Pascal rigidly, using an axiom-based method developed by Hoare. [143]

The division between the syntax and semantics of a programming language is not a sharp one, and not all authors agree on where it lies. Essentially, syntax is concerned only with those matters that can generally be defined with reference to the sequence of symbols in sentences of the language; semantics is concerned with everything else.

Wirth [109] has pointed out that even where the syntax of formal languages is context-free, its semantics may be context-dependent. In programming languages, semantics is concerned with such matters as type compatability in expressions and assignments. For example the Pascal expression

5 + 'B'

is valid syntactically but not semantically as the **integer** constant 5 is not of the same type as the **char** constant 'B'. had this expression been the controlling expression in a **while** loop, then additionally its resultant type would have had to be **boolean**: this exemplifies the context-dependency of formal language semantics - even when the syntax is context-free - which is one of the difficulties in the way of the achievement of formal semantics.

Ultimately, it is the implementation of a programming language in a compiler that defines its semantics; in written descriptions of the language the semantics is normally defined informally. In any case, it is often useful to leave certain aspects of the semantics implementation-dependent as the most appropriate way of implementing them may depend on the machine architecture in question. The type **char** and the value of **maxint** are two aspects of Pascal deliberately left undefined for this reason.

2.5. INTERACTIVE LANGUAGES

For most programming languages, the operation of compilation requires no interaction with the programmer, and is often carried out in batch mode. Once compiled, the object program produced by the compiler can be executed repeatedly on different data, without recompilation.

In such a system the compiler reports any error (syntactic or semantic) that it encounters, and then attempts to continue to process the source program and to report any further errors. Obviously it is no longer practicable to continue to generate object code. This has the advantage that the programmer has all the errors in his program reported together, and can correct them all before attempting to recompile it, but has the disadvantage that the compiler may not successfully recover from an error it encounters, and may then report large numbers of spurious errors. During the present work, the author had over two hundred errors reported after a compilation, all of which were corrected by the addition of a single semicolon near the top of the program.

For certain applications, compilation may take place interactively. The programmer types his program into the computer line by line, with the compiler reporting each error as soon as it is encountered, and the programmer correcting it immediately. Languages compiled in this way are usually specially designed for the purpose, and have been discussed by Kupka and Wilsing. [144] The interactive compilers used for such languages are normally called "interpreters" to distinguish them from batch-mode compilers, and the special problems of writing them have been discussed by Brown. [105]

These authors point out that systems based on interactive compilation actually require three different languages - the programming language itself, a Command language which controls such matters as the saving of completed programs, execution etc., and an Edit language which allows interactive editing of the program. This latter is especially useful for correcting errors which are only detected by the compiler some time after they have occurred.

Both the Edit and Command languages are normally very simple, each "sentence" consisting only of a single terminal symbol (e.g. a Command) followed by one or two arguments such as a filename or a line number. Their syntactic analysis is trivial.

The language most commonly implemented in this fashion is Basic [145, 146] though the approach has also been applied, at least for teaching purposes, to Fortran, [147] Algol [148] and Pascal, [149, 150] in the latter cases only a subset of the language being implemented. In the case of higher-level programming languages, certain problems may be encountered with the need to recompile the entire program every time a change is made by the Editor, and this could be time-consuming for large programs. It could however be avoided by a process of incremental compilation, as discussed by Atkinson _et al.,_ [151] but for teaching purposes, when the programs are normally short, repeated recompilation is probably the better approach.

2.6. FORMAL LANGUAGES IN CHEMISTRY AND INFORMATION WORK

Formal language theory has been applied in a number of areas in chemistry and information work: at the simplest level the interactive search languages used in online bibliographic retrieval systems [152, 153] have grammars which can be described by the methods developed by Chomsky. For the most part, they are Type 3 (Regular) languages, with trivial syntax analysis.

Some more sophisticated query languages have also been developed for specific applications, such as MQL (Medical Query Language) [154] which is designed to allow input of queries to a database in something approximating to natural language.

Specialised descriptive languages have been developed for use with chemical synthesis planning programs. In these programs the computer, upon being presented with a "target" chemical structure, is able by use of a database of chemical reactions called **transforms** to suggest possible synthesis routes leading to the target.

Formal languages have been developed for the description of the **transforms**, two such being CHMTRN [155] used by the LHASA (Logic and Heuristics Applied to Synthetic Analysis) [156, 157] program, and ALCHEM (A Language for CHEMistry) [158] used by the SECS (Simulation and Evaluation of Chemical Synthesis) [159, 160] program, which is historically an offshoot of LHASA. Figure 2.2 illustrates the description of a **transform** using ALCHEM. Each **transform** contains information which enables the computer to decide whether or not it is applicable to the synthesis of a particular target molecule.

Both languages have been designed to represent the **transform** in a manner which remains reasonably intelligible to a chemist, yet is amenable to computer analysis, and "compiler" programs have been written for them. Both have a fairly strict line format, and their structure is more akin to that of Fortran than those of more modern languages such as Algol and its descendents; their grammars are not formalised by production rules or syntax diagrams.

Line notations used for the representation of chemical structures as strings of alphanumeric symbols can be regarded as formal languages, and some success has been achieved in writing a context-sensitive grammar for the Wiswesser notation. [161] Lin

```
TYPE PATTERN
; PROXIMITY GUIDED EPOXIDATION
; ALCOHOL GROUP CIS TO EPOXIDE ON RING
; REF: E. COLVIN, J CHEM SOC PERKIN I 1989 (1973)
; CHEM COMM 858 (1971), HOUSE P. 305
EPOX
O—C—C—@1<1, 3, 2>/
PRIORITY 0
CHARACTER ALTERS GROUP
;  CHECK IF STEREOCHEMISTRY IS IMPORTANT
   IF STEREOCENTER IS CARBON OFFPATH THEN ; IT IS IMPORTANT
      BEGIN IF ALCOHOL IS WITHIN GAMMA TO ATOM 2 (1) THEN
         BEGIN IF BOND 1 AND (1) ARE CIS THEN ADD 50
         ELSE KILL ;EPOXIDATION WOULD HAVE WRONG STEREOCHEM
         IF (1) IS ONRING OF SIZE 5-6 THEN ADD 50
         DONE
      IF NITRILE IS EPSILON TO ATOM 2 (2) THEN
         BEGIN IF BOND 1 AND (2) ARE TRANS THEN ADD 30
         ELSE SUBT 30 ;EPOXIDE TRANS TO NITRILE IS FAVORED
         DONE
   DONE
   CONDITIONS SLIGHTLY OXIDIZING
   DELETE ATOM 1
   MAKE BOND FROM ATOM 2 TO ATOM 3
   END

COMPLETE
```

Figure 2.2: ALCHEM description of a chemical transform (from Wipke et al.[160])

et al. have also written a compiler which performs automatic syntax [162] analysis on their Separate Feature Linear Notation (SEFLIN). [163]

Formal language theory has also been applied in chemistry outside the area of artificial language design. Fehder and Barnett [164] suggested in 1965 that the principles of syntactic analysis could be applied to the analysis of molecular formulae, providing a means for determining the validity (grammatical correctness) of a given molecular formula, and other authors have followed up this approach. [165-167] Similar applications have been made in the analysis of nomenclature. [168-170]

Rankin and Tauber [171, 172] have applied formal language theory to the full topological representation of chemical structures, developing generative grammars based on production rules for certain classes of molecule; such grammars are also discussed by Whitlock. [173]

In their second paper [171] Tauber and Rankin suggested that sets of grammar rules could be used for compact storage of groups of related structures, such as leucine esters, different rules being used for the generation of the constant and variable parts of the structure. A similar approach to the storage of generic structures was later taken by Krishnamurthy and Lynch. [15, 16]

More recent work by Welford [67, 174] has extended the range of structure types that can be generated and recognised by formal grammars, and has formed a cornerstone of the research on generic structure representation at Sheffield University [174-177] of which this Thesis describes a part.

CHAPTER 3

THE INPUT LANGUAGE

"My language is plain."

Bret Harte (1836-1902)

Chapter 1 has surveyed the various types of expression found in generic structure descriptions in patent specifications and abstracts, and has outlined the reasons for the development of a special input notation, or language, for the description of such structures which will be intelligible to a chemist, information scientist or patent agent, yet sufficiently formalised for automatic analysis by computer, using the principles discussed in Chapter 2.

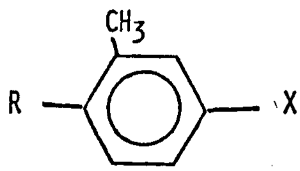
The language described in this Chapter, GENSAL, may be used to represent a generic structure unambiguously (in order that an unambiguous internal representation may be generated from it), and it has been designed to conform as far as possible to the type of description commonly found in chemical patent specifications. It is thus a formalised version of the generic structure description of patent specifications and abstracts: aspects of its formal grammar are described in Section 3.11., and as with many modern programming languages the grammar of GENSAL is expressed as a series of syntax diagrams, shown in Appendix 1. Throughout the text of this thesis, the syntactic metasymbols of GENSAL used as headings for the syntax diagrams are shown underlined.

3.1. GENERIC STRUCTURE DESCRIPTION USING GENSAL

The basic layout of generic structure descriptions in patent specifications and abstracts, as discussed in Section 1.2., is retained in GENSAL, one sentence of which describes one generic structure. Syntax Diagram 21 shows that the overall description of a structure has an introductory heading part, containing a reference number, and a **structure diagram** for the constant part of the structure which is followed by a series of statements, separated by semicolons; the sentence ends with a full stop. Figure 3.1 shows a simple generic structure and its GENSAL representation which, as can be seen, remains readily intelligible to a chemist.

The plethora of symbols used for structural and multiplicative

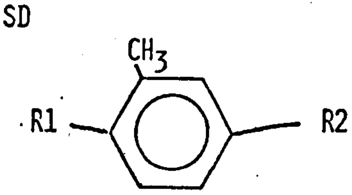
"FREE-TEXT" GENERIC STRUCTURE



WHERE R IS METHYL, ETHYL OR ISOPROPYL, AND X IS HALOGEN.

GENSAL NOTATION

INPUT 12345
SD



R1 = METHYL / ETHYL / ISOPROPYL ;
R2 = HALOGEN.

Figure 3.1

variables is reduced to two standard series: R1, R2, R3 etc. for structural variables (called substituents in GENSAL), and M1, M2, M3 etc. for multiplicative variables (called multipliers), as shown in Syntax Diagrams 3 and 4.

Variables in a GENSAL sentence must be introduced ("declared"), normally by appearing in a **structure diagram**, before being given values ("defined") in terms of chemical nature and position for substituents, and of selectors (giving integer ranges) for multipliers.

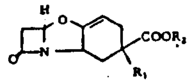
The definition of substituents and multipliers takes place in assignment statements, which contain facilities for assigning the same set of alternatives to groups of substituents or multipliers simultaneously (with both independent and non-independent selection of the alternative values) or for assigning to substituent combinations (forming an extra ring). The substituent value may be given in several different ways, and there is scope for indicating the position at which the substituent is attached, and any further substitution on it, down to any level.

Conditional definitions are indicated by IF and RESTRICT statements. The former allow the use of one of two alternative subordinate statements according to whether a condition involving substituents and multipliers already defined is TRUE or FALSE. The latter impose such conditions on the alternatives given in earlier assignment statements, allowing only those combinations of alternatives that result in the condition being TRUE.

The next nine Sections of this Chapter give a detailed description of the language, allowing a full understanding of the GENSAL notations for the actual patent examples shown with the Derwent Abstracts of the original specifications in Figures 3.2 to 3.11. A comprehensive instruction manual for GENSAL, with further examples, has been prepared by Hill.[178]

60257B/33 B02 C02 BEEC 02.02.78
BEECHAM GROUP LTD *GB 2013-676
01.02.79-GB-003610 (+004152) (15.08.79) A61k-31/42 C07d-205 C07d-263 C07d-498/04
10-Carboxy-3-oxo-6-oxa-1-aza-tri:cyclo-undec-7,8 ene derivs. - useful as beta-lactamase inhibitors

β-Lactam derivs. of formula (I) are new:



(I)

(R1 is H or 1-4C alkyl;
and R2 is H, a cation or an ester-forming gp.).

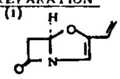
USES
(I) are β-lactamase inhibitors esp. useful for use in conjunction with penicillins and cephalosporins to give a synergistic antibacterial effect.

SPECIFICALLY CLAIMED
(I) where R1 = H and R2 = H, Li, Na, K, Ca/2, Mg/2 or NH_4.

BC(2-C1, 6-E5, 12-C9, 12-O1).4

42

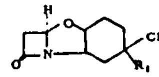
PREPARATION
(1)



$\xrightarrow[\text{with heating}]{CH_2=CR_1COOR_2}$ (I; R_2=ester gp.)

(I; R_2 = ester gp) $\xrightarrow{\text{Hydrolysis}}$ (I; R_2=H)

(2)
(II) $\xrightarrow{CH_2=CR_1CHO}$



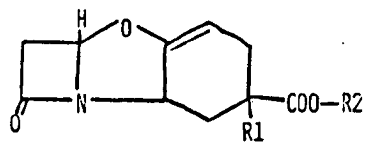
$\xrightarrow[\text{oxidn.}]{\text{mild}}$ (I)

(II) may be obtd. from clavulanic acid by adding it with N,N-dimethylformamide dimethylacetal to hydroquinone in DMF.

60257B GB2013676+

INFUT 2013676

SD



R1 = H / ALKYL <1-4> ;

R2 = H / 'CATION' / 'ESTER-FORMING GROUP' .

Figure 3.2

3.2. STRUCTURE DIAGRAM INPUT

As a whole, GENSAL is intended to be independent of any given computer system, and its high degree of readability makes it suitable as a means of describing generic structures manually, just as the programming language Algol is often used for the manual description of algorithms.

Nevertheless, certain aspects of GENSAL are intended to be implementation-dependent, and the most important of these are the **structure diagrams** which form an integral part of the language.

Any suitably-modified chemical structure graphics system might be used for their input, with a routine to convert its output into the connection table format used in the internal representation of the structure.

In the implementation described in Chapter 5 a modification of the program developed by Feldmann and others [179] and used in the Crystal Structure Search and Retrieval (CSSR) [180] and National Institutes of Health / Environmental Protection Agency (NIH/EPA) [181] substructure search systems, is being used. This is far from ideal, but has the advantages that it was provided free, and uses standard lineprinter characters in its display routines, and thus does not require any special hardware.

It is possible that an operational system might use a micro-computer as an intelligent terminal for the mainframe on which the bulk of the structure processing and searching would be carried out, and that the microcomputer would handle the chemical structure graphics locally, transmitting and receiving connection tables for each diagram.

A structure graphics system used with GENSAL requires certain features not found in all such systems. There must be a facility for defining nodes of the diagram as substituents (with the correct syntax) as well as as atoms of different types, and also a facility for applying multipliers (with the correct syntax) to nodes defined as substituents.

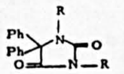
It must be possible to show that a particular node is connected back to a previously-defined part of the structure: in the modified Feldmann program used for the present work, this is achieved by attaching such an "apical" node to a dummy node, whose atomic type is given as "*".

It must be possible to show that a particular node is attached to one of the other nodes in the diagram, without specifying which. In patent specifications, and general chemists' usage, this is usually achieved by the convention of a bond drawn into the centre of a ring, but in the modified Feldmann program it is done by attaching the variably-positioned node to a dummy node of atomic type "#", which indicates that it may be attached to any other node in the diagram with sufficient spare valencies. If it is desired to restrict the available positions to, for example, those in a particular ring, this should be done by using a GENSAL <u>position set</u> (Section 3.6 below) elsewhere in the <u>structure description</u>.

Only nodes defined as <u>substituents</u> may have dummy "variable-position" nodes attached to them.

61502B/33 B03 INTE- 22.04.77
INTERX RES CORP *US 4163-058
22.04.77-US-790087 (31.07.79) A61k-31/41 C07d-233/74
5,5-Di:phenyl-hydantoin derivs. - useful as anticonvulsants, anti:epileptics and antiarrhythmics

Hydantoin derivs. of formula (I) and their acid-addn. and base salts, 1-4C alkyl halide quat. salts and N-oxides are new:



(I)

(R is H or $-CHR_1-XR_2$;
R_1 is H or 1-7C alkyl;
X is O or S;
R_2 is COR_4; and
R_4 is the acyl residue of a naturally occurring protein amino acid;
provided that one R must be other than H).

USE/ADVANTAGE

(I) are anticonvulsants, antiepileptics and antiarrhythmics. Compared with phenytoin and related cpds., (I) have greater solubility and enhanced stability without undesirable side effects. Dose is 200-800 mg. daily in man, 30-200 mg. daily in dogs etc.

B(5-B1M, 7-D9, 12-D4, 12-F1).4 110

WIDER DISCLOSURE

(I) are stated to be new when (a) R_1 is also CCl_3, CBr_3, CI_3, Ph, Me_2NCH_2, CHO, $PhOCH_2$, PhCH=CH, 2-furyl, pyridyl or R_3-phenyl; R_3 is OH, Cl, Br, I, OMe, COOMe, NO_2 or OCOMe; (b) X is also NR_1; (c) R_2 is also $PO(OH)_2$; and (d) R_4 is also $R_3C_6H_4$, pyridyl, pyridylmethyl, pyridyl N-oxides, the residue of any N-substd.-amino acid (the substituent being a protecting gp. removable by hydrogenolysis or hydrolysis), the residue of an N,N-di-(1-5C)alkyl- or 4-7C cycloalkyl-amino acid, $(CH_2)_nCOOH$, CH_2OCH_2COOH, $(CH_2)_nCOOMe$, $(CH_2)_nCOOEt$, $(CH_2)_nCONR_5R_6$, imidazolyl, 1-8C alkoxy, $PhCH_2O$, PhO or $O(CH_2)_nNR_5R_6$; where n is 1-5 and R_5 and R_6 are 1-5C alkyl or NR_5R_6 is a heterocyclic gp.

SPECIFICALLY CLAIMED

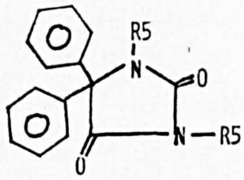
3-(N,N-Dimethylglycyloxymethyl)diphenylhydantoin (II) and its methanesulphonate and salicylate.

PREPARATION

5,5-Diphenylhydantoin (III) is either first treated with R_1CHX and then acylated with a reactive deriv. of R_4COOH, or it is converted to a Na, K, Ca or Mg deriv. and this is
61502B US4163058+

INPUT 4163058

SD



R5 = H / SD
- CH — R3 — R2
  |
  R1
;

R1 = H / ALKYL <1-7> ;

R2 = SD
—C — R4
  ||
  O
;

R3 = SD -O- / SD -S- ;

R4 = 'ACYL RESIDUE OF NATURALLY OCCURRING PROTEIN AMINO ACID' ;

RESTRICT <1> R5 <> H .

Figure 3.3

3.3. SIMPLE ASSIGNMENT STATEMENTS

Assignment statements express the definition of a substituent or multiplier. For both types of variable, the substituents or multipliers being defined are shown on the left-hand side of an assignment operator (normally "="), and the possible values on the right-hand side.

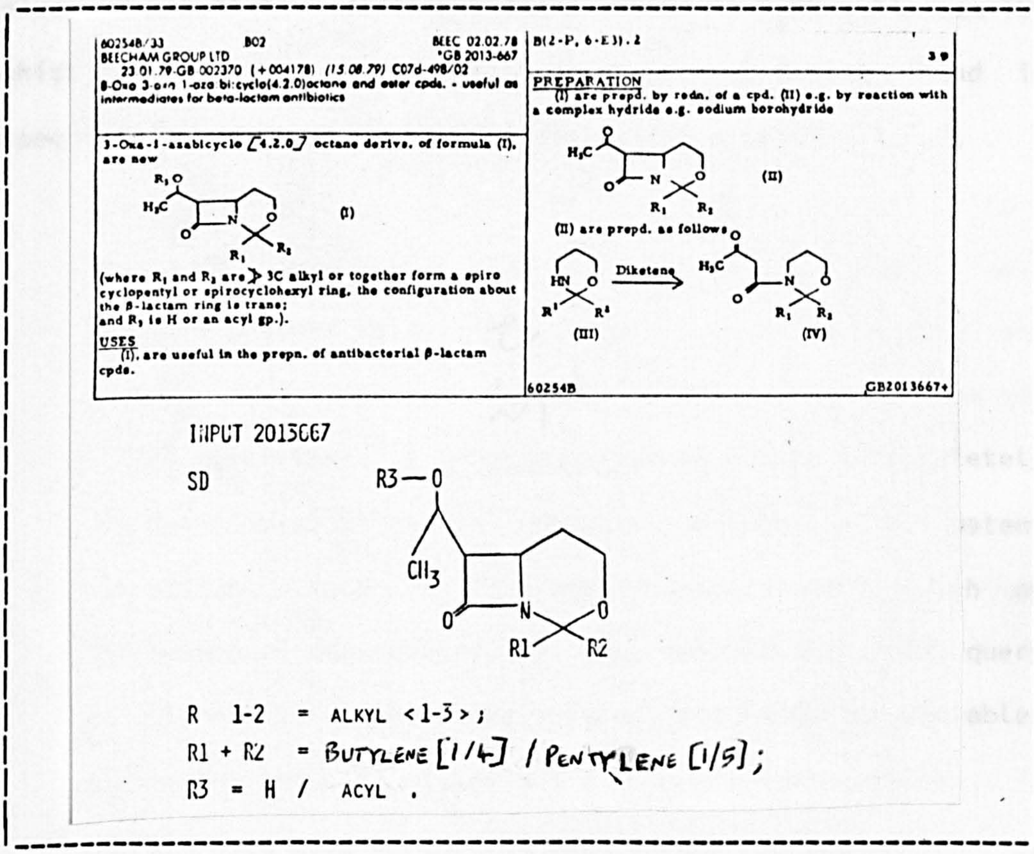


Figure 3.4

3.3.1. Substituent Assignments

The simplest form of assignment statement for substituents can be represented as follows:

```
------> substituent ----->[=]-----> substituent value ---------->
                           ^                              |
                           |---------[/]<-----------------
```

(This is a simplified version of the relevant syntax diagrams.)

It allows a single substituent on the left-hand side to be defined as having one of the values separated by the "/" delimiters on the right-hand side. Each alternative value may be given in one of five different forms, shown in Syntax Diagram 10, which correspond to the different types of expression found in specifications and abstracts and discussed in Section 1.2.2.

3.3.1.1. Unknown Value

A "?" represents a substituent whose nature is completely unknown. This situation usually occurs with patent expressions such as "optionally substituted", with no indication of the nature of the substitution. In query structures it might also be used as a value for variables indicating the unspecified parts around a substructure.

3.3.1.2. Structure Diagram

The substituent is defined by a **structure diagram**, which is input in exactly the same way as the main **structure diagram** for the constant part of the structure, and may, of course, have further substituents declared within it.

3.3.1.3. Nomenclatural Terms and Expressions

Specific nomenclatural terms represent a single chemical entity, and include terms such as "chloro", "methyl", "pyridyl" and "cyclohexyl". Simple linear formulae such as "CN", "COOH", and "NH_2" are also regarded as **specific nomenclatural terms.**

Essentially this is a shorthand method of inputting a **structure diagram:** an operational system might have sophisticated routines for nomenclature translation and linear formula analysis, though development of these has not formed part of the present work. At a simpler level, when a GENSAL sentence is being interpreted by computer, a file may be searched for a record of the structure of, e.g. phenyl, and if no entry is found a suitable message be printed at the terminal. Such a file may also be able to simplify compound terms such as "halogen" and "alkali metal". This is the approach used in the current implementation.

Homologous series terms describe classes of structural entities, and the **parameter list** which follows the term may impose restrictions on the variety of structures covered. **Parameter lists** are discussed more fully in Section 3.5.

Verbal expressions that do not correspond to any specific structure or structurally-defined class are enclosed in quotes to prevent any attempt by the computer to find a structure record.

The following are simple examples of <u>assignment statements</u>:

```
R1 = methyl / ethyl / cyclohexyl ;


R2 = 'electron-withdrawing group' ;

R3 =  SD        ___
             __/   \__ Cl
               \___/

          /  cyclohexyl ;


R4 = ?
```

and further examples may be found in the GENSAL notations for patent examples shown in Figures 3.2 and 3.3.

3.3.2. Multiplier Assignments

Simple multiplier assignments are of the form:

```
-------> multiplier -------------->[=]-------> selector ------->
```

and enable multipliers to be assigned a range of integer values.

Such a range is defined using the integer range given in Syntax Diagram 2 enclosed in angle brackets. It may consist of a single integer, or a group of single integers or "range fragments" separated by commas. Each range fragment consists of two integers separated by a hyphen, and represents all the integers from the lower to the upper inclusive. The last integer before the end of the selector (immediately before the right angle bracket) may be followed by a hyphen without a second integer, in which case all the integers from the bound upwards are included.

Thus the selector <0-6,8,12,15-19,23-25,31,43-> includes the integers 0, 1, 2, 3, 4, 5, 6, 8, 12, 15, 16, 17, 18, 19, 23, 24, 25, 31, 43, 44, 45, 46, etc., potentially up to infinity.

As can be seen from Syntax Diagram 1, negative integers are not allowed in GENSAL, and there is also a requirement that the values in an integer range (Syntax Diagram 2) must increase from left to right.

An example of a simple multiplier assignment is

    M1 = <3-5>

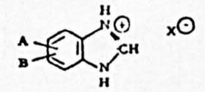
and further examples may be found in Figures 3.5 and 3.10.

3.4. MORE COMPLEX ASSIGNMENTS

The full syntax diagram for assignment statement (No. 16), and those with which it is defined, allow much more complicated assignments to be concisely represented.

03566B/02 E13 G06 R23 DUPO 23.11.77
DU PONT DE NEMOURS CO *US 4131-467
23.11.77-US-854262 (26.12.78) G03c-01/34 G03c-05/30
Di:hydroxy-benzimidazole antifoggants - are incorporated in silver halide materials and developer solns.

Dihydroxy benzimidazoles of formula (I) are incorporated in the silver halide emulsion or auxiliary layers of photographic materials or in developing solns.



In (I), A and B are OH radicals in the 4,7 or 5,6 position; and X is Br, Cl or I. (I) are added at 5×10^{-4} - 5×10^{-2} mole/mole of Ag halide.

USE

(I) improves the sensitometric properties of the emulsion by acting as an antifoggant and as a development accelerator by increasing gradient and Dmax.

DETAILS

The cpds. are H_2O and alcohol soluble and can be added to the emulsion in soln. or as a dry solid, pref. after the emulsion has been chemically sensitised.

E(6-D5) G(6-H12, 6-H13).

When used in developing solns. for silver halide emulsions pref. 10^{-4} to 10^{-3} mole of cpd. is added per litre.

EXAMPLE

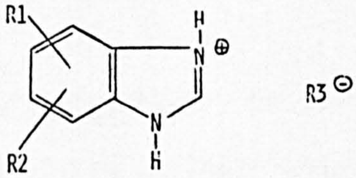
A gelatino-silver chlorobromide emulsion contg. a silver halide equiv. to 1.5 moles $AgNO_3$ was sensitized with 3.3×10^{-6} moles gold sensitizer and 1.74×10^{-5} moles sodium thiosulphate. The emulsion was digested for 60 mins. at 125°C and then 2.27×10^{-4} mole of 4,7-dihydroxybenzimidazolehydrobromide was added per mole Ag halide. After the addition of coating aids and hardeners the emulsion was coated at 35-80 mg AgH per sq. decimeter.

The sample was exposed for 4 secs. through a $\sqrt{2}$ optical wedge with radiation equiv. to 4470 metre candle-seconds and developed for 22 seconds at 38°C in a phenidomer hydroquinone developer. The material had the following properties, the values in brackets being for a material contg. the conventional antifoggant 4-hydroxy-6-methyl-1,3,3a,7-tetraazaindene. Dmin = 0.06 (0.06); Dmax = 4.22 (1.24); speed = 75 (34); average gradient = 1.83 (0.49). (8pp1355).

US4131467

INPUT 4131467

SD



R1 = [4-5] OH;
IF R1 = [4] THEN R2 = [7] OH
ELSE R2 = [6] OH ;
R3 = BR / CL / I .

Figure 3.5

3.4.1. Combined Substituents

Two substituents can be combined to form a ring:

R1 + R2 = cyclopentyl / cyclohexyl

i.e. R1 and R2 combine together, forming with atoms of the constant part of the structure, either a cyclopentyl or a cyclohexyl ring. A firm decision has not yet been made as to whether this convention is preferable to one where the substituent values describe only the atoms that are added to the structure, i.e.:

R1 + R2 = 1,3-propanediyl / 1,4-butanediyl

The former has the advantage of being more consistent with normal patent usage, but the latter is simpler to implement, and has been chosen for the present work.

Only two substituents may be combined in this way, and both must be singly connected in their independent existence.

Figure 3.4 shows an example of substituent combination in a patent; in this case the possibility of combining R1 and R2 is alternative to their being separate singly-connected radicals.

3.4.2. Group Assignment Statements

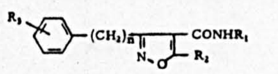
As a convenient "short-cut" several substituents or multipliers can be defined simultaneously:

R1-2 = phenyl / cyclohexyl / cyclopentyl

i.e. R1 and R2 are both defined by the three alternatives shown. (Note that in this case the three nomenclatural terms are being used to represent singly-connected radicals, substituted on the constant part of the structure, in contrast to their use above

61501B/33 B03 SANO 27.12.77
SANDOZ INC *US 4163-057
27.12.77-US-864380 *(31.07.79)* A61k-31/42 C07d-261/14
3-Phenylalkyl-isoxazole-4-carboxamide(s) - useful as minor tranquillisers, sleep inducers and muscle relaxants

Isoxazoles of formula (I) are new:


(I)

(R_1 is H or 1-4C alkyl;
R_2 is H or 1-4C n-alkyl;
R_3 is H, F, Cl, CF_3 or 1-4C alkoxy; and
n is 1-4).

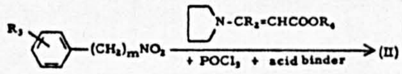
USES
(I) are minor tranquillisers, sleep inducers and muscle relaxants. Dose is 1-500 mg/kg. daily.

WIDER DISCLOSURE
(I) are also disclosed in which R_3 may also be 1-4C alkyl.

B(7-E1, 12-C8, 12-C10, 12-E2). 3
109

SPECIFICALLY CLAIMED
(I) where R_3 = H, n = 2, R_2 = Me and R_1 = H or Me.

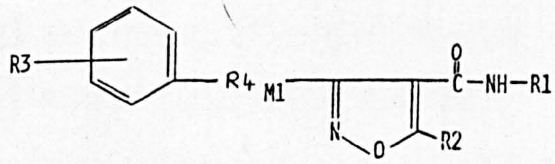
PREPARATION
A corresp. 4-isoxazolecarbonyl halide is reacted with R_1NH_2, esp. in solvent at 20-35°C for 10-16 hrs. The starting halide is obtd. by hydrolysis of a corresp. 4-isoxazolecarboxylic acid Me or Et ester (II) to the free acid, followed by treatment with halogenating agent. The starting ester may be obtd. as follows:



(m is 2-5; and R_4 is Me or Et).

EXAMPLE
Reaction of ethyl-3-pyrrolidone crotonate with 1-nitro-3-phenylpropane in presence of NEt_3 and $POCl_3$ gave Et 3-phenethyl-5-methylisoxazole-4-carboxylate, which was
61501B US4163057+

INPUT 4163057
SD



R1 = H / ALKYL <1-4> ;
R2 = H / ALKYL <1-4 > T<0> Q<0>;
R3 = H / F / CL / CF_3 / ALKOXY <1-4> ;
M1 = <1-4> ;
R4 = CH_2 .

Figure 3.6

with a substituent combination.) The integers used to list the substituents being defined are arranged in the syntax for an integer range.

3.4.2.1. Assignment Operators

The five available assignment operators shown in Syntax Diagram 16 have different meanings, which may be useful when several substituents or multipliers are being defined together in a group assignment, and their values are not necessarily independent.

= : The substituents or multipliers in the group are independently selected from the alternatives in the substituent definition or selector on the right-hand side of the assignment statement. This is the most commonly-encountered operator.

S= : All the substituents or multipliers in the group must have the same value, which is selected from the substituent definition or selector on the right hand side of the assignment statement.

D= : Each of the substituents or multipliers in the group must have a different value, all the values being selected from those on the right-hand side of the statement.

$= : Not all the substituents or multipliers in the group may be the same (which, using "=", they could be), but they need not all be different.

#= : Not all the substituents or multipliers in the group may be different (i.e. at least two must be the same), but they need not all be the same.

Examples of such simultaneous assignments are as follows:

a) R1-3 = phenyl/ cyclohexyl / cyclopentyl ;

R1, R2 and R3 can be independently either phenyl, cyclohexyl or cyclopentyl. There are thus 27 (3 x 3 x 3) possible permutations, assuming that there are no symmetry considerations involved.

b) R4-6 D= phenyl / cyclohexyl / cyclopentyl;

R4, R5 and R6 must be different, each being selected from the possibilities phenyl, cyclohexyl and cyclopentyl. There are thus 6 (3 x 2 x 1) possible permutations.

c) R7,9-10 S= phenyl / cyclohexyl /cyclopentyl;

R7, R9 and R10 must be the same. There are only three possible permutations (all phenyl, all cyclohexyl or all cyclopentyl).

d) R11,13,15 $= phenyl /cyclohexyl / cyclopentyl;

R11, R13 and R15 are not all the same, each being otherwise selected from the possibilities given. This leaves 24 possible permutations, there being three ways in which all can be the same.

e) R16-18 #= phenyl / cyclohexyl / cyclopentyl;

R16, R17 and R18 are not all different, but are otherwise selected from the available possibilities. Here there are 21 possible permutations, as there are six ways in which the three may be all different.

3.4.2.2. Selected Group Assignments

A group assignment statement can begin with a selector which allows just some of the substituents or multipliers in the group to be assigned values from the substituent definition or selector on the right-hand side of the assignment statement. e.g.:

<2-3> R1-5 = phenyl / cyclohexyl / cyclopentyl;

means that 2 or 3 of the group of substituents R1, R2, R3, R4 and R5 are independently selected from the list phenyl, cyclohexyl and cyclopentyl, the others remaining undefined at this point in the GENSAL structure description.

3.5. HOMOLOGOUS SERIES IDENTIFIERS AND GRAMMARS

Certain terms used in generic structures cover a range of specific substructures all of which are alternative to each other at that point. The most common example is the term "alkyl" which covers all rooted acyclic substructures containing carbon and hydrogen only, without any unsaturations.

Welford [67, 174] has developed chemical grammars to deal with this type of expression, and not only are they applicable to terms such as alkyl and alkenyl, which are commonly understood by chemists as "homologous series terms", but they may also be applied to many less precise terms which are none the less "structurally recognisable" -- that is, terms which encompass a range of substructures that have a particular structural feature in common, such as "aryl" and "heterocyclic". Each valid **homologous series identifier** is associated with a list of parameters to the chemical grammars, the values of each parameter being defined by means of a selector.

As can be seen from the syntax diagram for parameter (No. 7), the parameter may be indicated by a Parameter Identifier or a substituent enclosed in quotes. The standard Parameter Identifiers cover features such as atom count, branch points etc., and are shown with their meanings in Table 3.1, though it is possible that the list may be modified as the chemical grammars are further developed. Non-standard parameters, shown by substituents in quotes, cover such features as interruptions in the chain, and substitutions on it. Thus for the **homologous series identifier** "alkyl", all the parameters will be zero, except for C, T, Q and P, which can take on any (mutually consistent) values.

Syntax Diagram 10 allows a parameter list to follow a **homologous series identifier,** thus more closely defining any of the standard parameters or introducing non-standard ones. This results in

C	Carbon count
E	double bonds (alkEne)
Y	triple bonds (alkYne)
Q	Quaternary branches
T	Ternary branches
RC	number of Rings
RN	Number of Ring atoms
RS	number of Ring Substitutions
RF	number of Ring Fusions
RA	number of Aromatic Rings
Z	number of heteroatoms

Table 3.1: Parameter Identifiers and their meanings

expressions like:

alkyl C<3-8> T<1-2>

which indicates alkyl groups containing between 3 and 8 carbon atoms, with 1 or 2 ternary branching atoms. The parameters may appear in any order, or be absent altogether, in which case their default values will be the widest possible range compatible with those parameters that are present (including any implicit in the **homologous series identifier** itself). In view of the way in which the Parameter Identifier "C" (for carbon count) occurs almost every time a **homologous series identifier** is used a shorthand has been introduced whereby the C may be omitted, provided this is the first parameter in the list. e.g.

alkyl <1-4>

alkyl <3-8> T<1-2>

Further examples of **homologous series terms** and parameter lists are:

a) cycloalkyl <10-20> RC<2->

(Between 10 and 20 carbon atoms, and at least two rings.)

b) alkyl <3-12> 'R5' <0-1>

(Between 3 and 12 carbon atoms, and 0 or 1 occurrences of R5.)

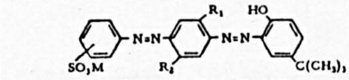
c) carbacyclic <6-10> E <1->

(Between 6 and 10 carbon atoms, and at least one double bond (number of triple bonds not specified). The term "carbacyclic" is used to indicate acyclic hydrocarbons.)

The assignment statements for R2 and R3 in Figure 3.5 include **homologous series identifiers** and parameter lists; in the case of R2 the specification of no ternary or quaternary branching atoms is equivalent to the statement in the Derwent Abstract that it is an n-alkyl group.

004848/01 E21 A60 F06 (A23) SANO 22.06.77
SANDOZ LTD *GB 2000-171
22.06.77-CH-007654 *(04.01.79)* C07c C09b-31/06
Sulphonated dis:azo dyes for polyamides, leather etc. - prepd. from sulpho-amino-azobenzene and para tert. butyl-phenol coupler

Disazo cpds. of formula (I) and their mixts. are new



(I)

R_1 and R_2 are each H or 1-4C alkyl or alkoxy. M = H or equiv. of a non-chromophoric cation (particularly Na), and the SO_3M gp. is in the m- or p-position. Pref. cpds. have R_1 and R_2 = H, CH_3, C_2H_5, CH_3O or C_2H_5O, esp. R_1 = H or CH_3 and R_2 = H, CH_3 or CH_3O with ≥1 of them H. (I) are made conventionally from the appropriate 4-amino-azobenzene sulphonic acid by diazotisation and coupling to 4-tert. butylphenol.

USE/ADVANTAGES

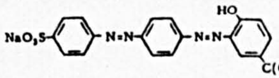
Used for dyeing or printing natural or synthetic polyamides (wool, silk or nylon), leather, base-modified polyolefins and polyurethanes, particularly for printing nylon

A(8-E3A3) E(21-B6) F(3-F2, 3-F6A, 3-F8, 3-F10, 7 :

carpets. (I) are fast to rubbing, light, wet treatments and give even dyeings on stripy nylon. When used with other dyes they are not subject to catalytic fading.

EXAMPLE

(All pts. by wt.) 27.7 pts. 4-amino-1,1'-azobenzene-4'-sulphonate were dissolved in 200 pts. water at 35-40°C (pH 9-10 with 30% aq. NaOH) then diazotised. The resulting suspension was added over 30 min. to a soln. of 15.2 pts. p-tert butyl phenol in 100 pts. acetone and 200 pts. water keeping pH at 8.5-9 with 10% NaOH. After stirring for 1 hr, the mixt. was acidified and salted out to give the dye



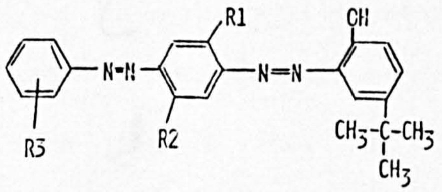
This dyed wool or synthetic polyamide in pure yellow shade with good light- and wet-fastness.(5pp1251)

3-F16A).

GB2000171

INPUT 2000171

SD



R 1-2 = H / ALKYL <1-4> / ALKOXY <1-4> ;

R3 = [3-4] SO_3 SB (H / 'EQUIV. OF A NON-CHROMOPHORIC CATION') .

Figure 3.7

3.6. POSITION SETS

Syntax Diagram 14, for <u>definition element</u> (of which <u>substituent value</u> is a simple case), allows the inclusion of some information about the position(s) of <u>substituents</u> being defined. A <u>position set</u> at the beginning of a <u>definition element</u> indicates the position(s) in the constant structure at which the <u>substituent</u>(s) currently being defined may be attached. Thus

R1 = [2,4] Cl

means that R1 is a Cl group attached in either position 2 or position 4 of the constant structure.

A position set following a substituent value indicates the position(s) in the substituent through which it may be attached to the constant structure. The example in Figure 3.12 shows that R1 is a nicotinic acid moiety attached through its 2, 4 or 6 position to the 2, 3, 4, 5 or 6 position of the toluene moiety. In both cases the numbering system refered to is the standard one for a nomenclatural term, or whatever numbering of the atoms was employed in the graphic input of the **structure diagram** in question.

The appearance of position sets after the reference to the structure to which they apply allows immediate automatic checking on the availability of the specified positions in the structure in question, which can be valuable in the machine processing of GENSAL.

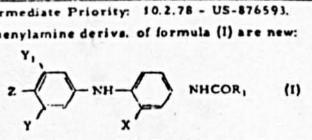
Figures 3.6 and 3.7 show patent examples involving position sets. The extent to which position sets need to be used in GENSAL notations may depend on the facilities available in the graphics system being used for indicating alternative positions of attachment (especially to rings).

For doubly-connected substituents, or combinations of singly-connected ones, it may be necessary to specify pairs of positions in the position set. In this case, the positions in each pair are separated by a "/", and the alternative pairs by commas. The order of the positions in such pairs is significant, and this may be important if the bond orders of the two connections are different, or if the substituent value is not symmetrical.

SHELL OIL CO *US 4181-519
22.01.79-US-005633 (+761515) (01.01.80) A01n-09/12 C07c-103/73
4-Acylamino di:phenylamine derivs. - useful as herbicides, esp. for post/emergent application

Intermediate Priority: 10.2.78 - US-876593.

Diphenylamine derivs. of formula (I) are new:



(I)

(X is F, Cl, Br or 1-6C alkyl, opt. substd. by ≥1 halo;
Y and Y_1 = H, F, Cl, Br or alkyl or alkoxy opt. substd. by ≥1 halo;
Z is H, F, Cl, Br, 1-6C alkyl, alkoxy, alkylthio, alkylsulphinyl or alkylsulphonyl (all opt. substd. by ≥1 halo) or is R_2R_3N;
R_2 and R_3 = H or ≯6C alkyl or cycloalkyl; R_3 may also be 1-6C alkoxy;
R_1 = 1-4C alkyl or 1-R_4-cyclopropyl;
R_4 = H, 1-4C alkyl, F, Cl or Br; provided
(a) when Z = H, X is not haloalkyl; (b) when R_1 is 1-R_4-cyclopropyl, at least 1 of Y, Y_1 and Z is other than H; (c) when R_1 is alkyl at least 2 of Y, Y_1 and Z are other than H).

USE
(I) are herbicides (some selective), esp. for post-emergent use. The usual application rate is 0.1-10 lbs./acre in standard formulations.

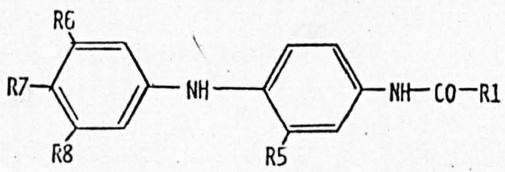
PREPARATION
By acylating the appropriate 4-aminodiphenylamine (II) with R_1COHal in presence of usual solvents and acid acceptors. (II) are made by condensing a Y, Y_1, Z-substd. anilide with a 3-X-4-chloronitrobenzene, then reducing the resulting 4-nitrodiphenylamine.

EXAMPLE
17.8g. 4-amino-2,3'-bis(trifluoromethyl)-4'-chlorodiphenylamine and 5g. Et_3N were dissolved in 100 ml. THF, then treated dropwise with 6g. 1-methylcyclopropanecarbonyl chloride. The mixt. was refluxed for 1 hr., poured into ice-water and extracted with 3 x 150 ml. ether. The ex-

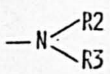
05035C US4181519+

```
INFUT 4181519

SD
```




```
R1 = ALKYL <1-4> /  CYCLOPROPYL  SB [1] R4  ;
R4 = H / ALKYL <1-4> / F / CL / BR  ;
R5 = F / CL / BR / ALKYL <1-6> SB < 0-> HALO ;
R 6,8   = H / F / CL / BR / ( ALKYL / ALKOXY )
                          SB < 0->  HALO ;
R7 = H / F / CL / BR / ( ALKYL <1-6>/ ALKOXY <1-6> /
        ALKYLTHIO <1-6> / ALKYLSULPHINYL <1-6> /
        ALKYLSULPHONYL <1-6> )  SB <0-> HALO / SD
```




```
                                          ;
R 2-3  = H / ALKYL <-6> /  CYCLOALKYL <-6>;
R3 = ALKOXY <1-6>;
IF R1 = CYCLOPROPYL SB [1] R4  THEN
        RESTRICT <1-> R 6-8  <> H ;
IF R1 = ALKYL THEN RESTRICT <2-> R 6-8   <> H ;
IF R7 = H THEN RESTRICT R5<>ALKYL <1-6>  SB <1-> HALO .
```

Figure 3.8

3.7. NESTED SUBSTITUTION

As was stated in Chapter 1, the variable parts in a generic structure may themselves be further substituted to any level. GENSAL is able to show this clearly and concisely by means of the mutually recursive Syntax Diagrams 14 and 15 in which parentheses are used in expressions involving the four substitution operators "/", "&", "SB" and "OSB" to remove any possible ambiguity.

The operators respectively represent "or", "and", "substituted by" and "optionally substituted by", and are evaluated in the order "&", followed by "SB" and "OSB" (ranking equally and evaluated from left to right), followed by "/". Expressions in parentheses are evaluated before "&".

This precedence order has been adopted because it appears to provide the most natural form for complex expressions: AND is conventionally evaluated before OR in Boolean expressions, and the intermediate positioning of SB and OSB allows several substitutions to be made on each alternative without the use of parentheses.

The following expression includes examples of the use of all four operators:

```
R1 = phenyl sb (Cl / Br / I)  & nitro  /
     (cyclohexyl / cyclopentyl)
     sb (amino / pyridyl osb methyl & methoxy) /
     naphthyl
```

The expression indicates that R1 has three possible basic alternatives:

1. A phenyl group substituted both by

   a) either Cl or Br or I

   and by b) nitro

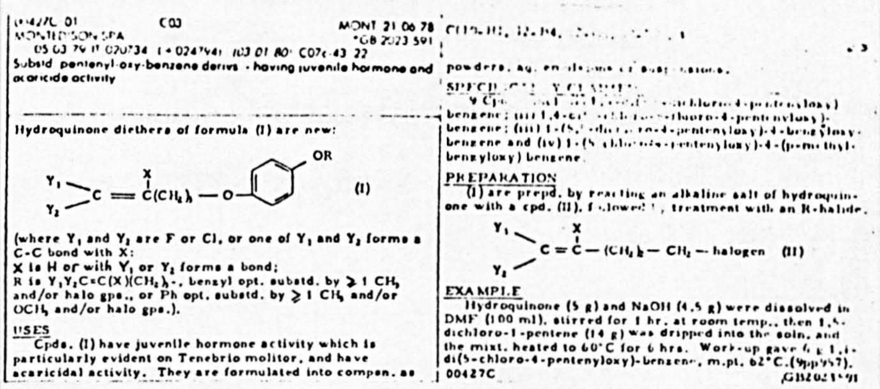
2. A cyclohexyl or cyclopentyl group substituted by either

   a) an amino group

   or by b) a pyridyl group, itself optionally further substituted both by methyl and by methoxy (i.e by both or by neither)

3. A naphthyl group, not further substituted.

The examples in Figures 3.8 and 3.9 include assignment statements involving parentheses to indicate further substitution, and that in Figure 3.9 also shows how it may be necessary to alter the way in which the generic structure is expressed in the original specification or abstract in order to encode it in GENSAL. The design of GENSAL is intended to minimise the need for such alterations, but occasionally they do become necessary.



00427C 01 C03 MONT 21 06 78
MONTEDISON SPA *GB 2023 591
05 03 79 IT 020734 (+024794) (03 01 80) C07c-43 22
Subst'd pentenyl-oxy-benzene derivs. - having juvenile hormone and acaricide activity

Hydroquinone diethers of formula (I) are new:

(I)

(where Y_1 and Y_2 are F or Cl, or one of Y_1 and Y_2 forms a C-C bond with X:
X is H or with Y_1 or Y_2 forms a bond;
R is $Y_1Y_2C=C(X)(CH_2)_3$-, benzyl opt. substd. by ≥ 1 CH_3 and/or halo gps., or Ph opt. substd. by ≥ 1 CH_3 and/or OCH_3 and/or halo gps.).

USES

Cpds. (I) have juvenile hormone activity which is particularly evident on Tenebrio molitor, and have acaricidal activity. They are formulated into compsn. as powders, aq. emulsions or suspensions.

SPECIFICALLY CLAIMED

...benzene; ... 1,4-di(5-...fluoro-4-pentenyloxy)-benzene; (iii) 1-(5,5-dichloro-4-pentenyloxy)-4-benzyloxy-benzene and (iv) 1-(5-chloro-4-pentenyloxy)-4-(p-methyl-benzyloxy) benzene.

PREPARATION

(I) are prepd. by reacting an alkaline salt of hydroquinone with a cpd. (II), followed by treatment with an R-halide.

(II)

EXAMPLE

Hydroquinone (5 g) and NaOH (4.5 g) were dissolved in DMF (100 ml), stirred for 1 hr. at room temp., then 1,5-dichloro-1-pentene (14 g) was dripped into the soln. and the mixt. heated to 60°C for 6 hrs. Work-up gave 6 g 1,4-di(5-chloro-4-pentenyloxy)-benzene, m.pt. 62°C.(9pp957).
00427C GB2023591

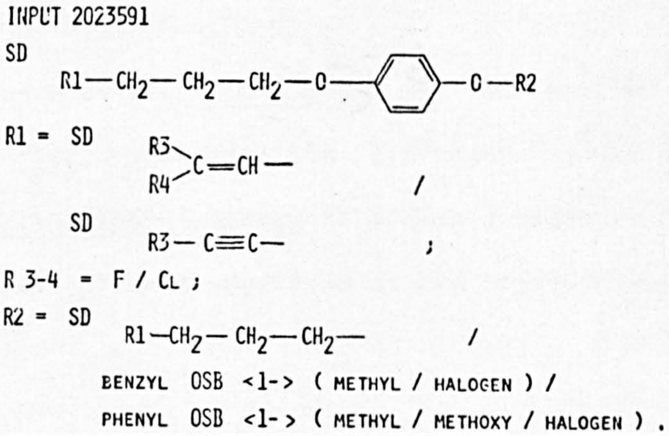


Figure 3.9

3.7.1. Selectors in Definition Expressions

A selector at the start of a definition element indicates multiple occurrences of whatever follows. Since the main part of the definition element may be a parenthesised definition expression, it is possible to have several such selectors applying to the same substituent value: in this case their effects are multiplied, with the result that the expression

```
<2-4> ( <2> methyl / ethyl )
```

implies the presence of between 4 and 8 methyl groups, or of between 2 and 4 ethyl groups.

3.7.2. Position Sets in Definition Expressions

Similarly, several position sets may be applied to a single substituent value, appearing in different recursions of the syntax for definition elements. Here each successive position set must be a subset of that specified at the previous recursion. The statement

```
R1 = [2-6] (methyl / [3,5] hydroxy)
```

is therefore valid, whereas

```
R1 = [2,4,6] (methyl / [3,5] ethyl)
```

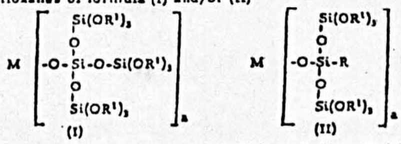
is erroneous.

3.7.3. Substituents as Substituent Values

It may be convenient, and is occasionally found in specifications and abstracts, to define one structural variable in terms of

05941B/03 E11 H08 OLIN 20.05.77
OLIN CORP *US 4132-664
20.05.77-US-798970 (02.01.79) C10m-03/46
Hydraulic and heat-transfer fluids - contg. alkoxy-siloxane(s)

The use of hydraulic and heat-transfer fluids contg. alkoxy-siloxanes of formula (I) and/or (II)



where a = 2-4; M is an opt. substd. branched or straight-chain hydrocarbon radical; and R and R^1 are each independently selected from H, alkyl, alkenyl, aryl and aralkyl, provided that at least the majority of R^1 are sterically hindered ≥3C alkyl) is claimed.

ADVANTAGES

E(5-E3) H(8-D5). 380

(I) and (II) have good hydrolytic stability, high lubricity and low viscosity indices.

DETAILS

The prepn. of cpds. (I) and (II) is described in US 3992429. The pref. cpds. are those where a = 2; M is a 4-12C hydrocarbon radical; R is H, 1-8C alkyl or alkenyl, or 6-14C aryl or aralkyl; R^1 is hindered 4-12C alkyl.

The fluids can comprise (I) and/or (II) alone or diluted, e.g., with glycols or glycol mono- or diethers. They can also contain conventional additives.

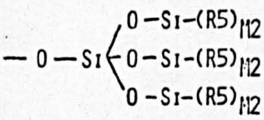
EXAMPLE

(II; a = 2, M = CH_2CH_2, R = Me, R^1 = s-Bu) has a pour point of < -40°F, a VI of 339, a 4-ball wear scar of 0.66 mm and a flash point of 420°F.(7pp367).

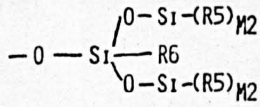
US4132664

```
INPUT 4132664

SD
          R1 —— (R2)M1
R1 = CARBACYCLIC SB ? ;
R2 S= R3 / R4 ;
R3 = SD
```



```
;
R4 = SD
```



```
;
R 5-6 = H / ALKYL / ALKENYL / ARYL / ARALKYL ;
M1 = <2-4>;
M2 = <3> ;
IF R2 = R3 THEN IF M1 =<2> THEN RESTRICT <10-> R5 = ALKYL <3->
                           ELSE IF M1 =<3>
                                THEN RESTRICT <14-> R5 = ALKYL <3->
                                ELSE RESTRICT <19-> R5 = ALKYL <3->
              ELSE IF M1 =<2> THEN RESTRICT <7-> R5 = ALKYL<3->
                           ELSE IF M1 =<3>
                                THEN RESTRICT <10-> R5 = ALKYL<3->
                                ELSE RESTRICT <13-> R5 = ALKYL<3-> .
```

Figure 3.10

another. This is usually done where there is further substitution involved, as in

R1 = R2 sb (methyl/ethyl)

or

R1 = (phenyl/naphthyl) sb R2

Syntax Diagram 14 permits a substituent in place of a substituent value, and the substituent given may or may not already have been defined; the definition eventually given is treated as a parenthesised expression.

GENSAL additionally permits a substituent to be defined in terms of itself, as in

R1 = methyl sb R1

and it can be seen that this corresponds to an infinite-length polymer.

3.7.4. Further Substitution on Parenthesised Expressions

Where one of the further substitution operators "SB" and "OSB" appears immediately after a parenthesised expression, it is understood that the substitution is made on the highest level of substitution within the parenthesised expression. For example, in the expression

(phenyl / cyclohexyl sb methyl) sb Cl

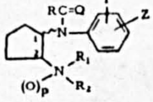
the phenyl group is substituted by chlorine, and the cyclohexyl group by both methyl and chlorine; the methyl group is not further substituted.

5?472B/?8 B05 UPJO 30.11.76
UPJOHN CO *US 4159-340
15 05 78-US-906429 (+746191) (26.06.79) C07c-103/34
Amino-cycloaliphatic amide(s) - possess CNS antidepressant properties and have a better therapeutic ratio than imipramine

Intermediate priorities: 15.3.77; 9.2.78-US-777599; 876349.

N-(2-Aminocyclopentyl)-N-acylanilides of formula (I) and their acid addition salts, are new

(I)



(the wavy line indicates cis or trans configuration of the substits. in the 1 or 2 posns. of the cyclopentyl ring;

p is 0 or 1;
Q is O or S;
R is 1-3C alkyl, vinyl, 3-6C cycloalkyl, ethoxy or methoxymethyl;
R_1 is H or 1-3C alkyl;
R_2 is $CH_2C_6H_5$, $(CH_2)_2C_6H_5$, 3-6C (allylic) alkenyl;
Y and Z are each H, F, Cl, Br, CF_3, 1 or 2C alkyl, 1 or 2C alkoxy and when Y is CF_3, Z is H, when Y is 1 or 2C alkoxy and Z is H, the 1 or 2C alkoxy is in the 3-posn., when Y and Z are both halo or 1 or 2C alkoxy, they are present in the 3- and 4-, or 3- and 5-posns).

B(10-A3, 10-B2F, 12-C6).3 106

USE

(I) possess potent CNS antidepressant properties and have a better therapeutic ratio than imipramine and long-acting activity which allows longer durations between administrations, e.g. once a day. Dose is 1-100 mg/day.

SPECIFICALLY CLAIMED

3,4-Dichloro-N-/2-(N-allyl-N-methylamino)cyclopentyl/-propionanilide, 3,4-dichloro-N-/2-(N-β-phenethyl-N-methylamino)cyclopentyl/propionanilide; 3,4-dichloro-N-/2-(N-benzyl-N-methylamino)cyclopentyl/propionanilide; and their acid addn. salts.

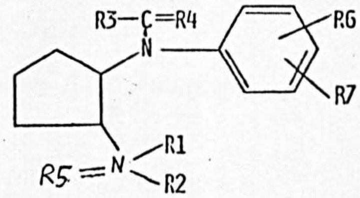
PREPARATION

Cpds. (I; Q is O and p is 0) are prepd. by heating a cpd. of formula (II) and an anhydride of the appropriate carboxylic acid to form the N-acylated prod.

52472B US4159340+

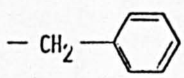
INPUT 4159340

SD



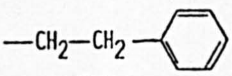
R1 = H / ALKYL <1-3>;

R2 = SD



/

SD



/

ALKENYL <3-6>;

R3 = ALKYL <1-3> / VINYL / CYCLOALKYL <3-6> /
ETHOXY / METHOXYMETHYL ;

R4 = O / S ;

R5 = <0-1> O ;

R 6-7 = H / F / CL / BR / CF3 / ALKYL <1-2> / ALKOXY <1-2> ;
IF R6 = CF3 THEN R7=H;
IF R6 = ALKOXY <1-2> AND R7 = H THEN
RESTRICT R6 = [3] ;
IF<2>R 6-7 = HALO / ALKOXY <1-2>
THEN BEGIN
RESTRICT R6 = [3] ;
RESTRICT R7 = [4,5]
END.

Figure 3.11

Had the expression been written without parentheses, it would have indicated an unsubstituted phenyl group, or a cyclohexyl group substituted by a methyl group, itself further substituted by chlorine.

This "highest-level" convention defines the level of substitution on substituents used as substituent values. In the statements

```
R1 = R2 sb Cl ;
R2 = phenyl sb methyl
```

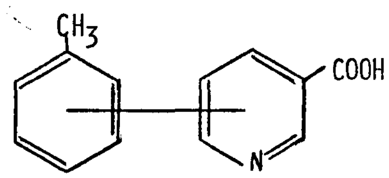
it is understood that the chlorine is a substituent on the phenyl group.

3.8. SPECIAL RESTRICTIONS IN GENERIC STRUCTURES

GENSAL provides two types of statement which allow special restrictions to be placed on the variety of specific structures covered by a generic structure : "IF" statements and "RESTRICT" statements.

Both employ the syntactic construct, condition, the former using the result of the condition (TRUE or FALSE) to determine which of two alternative statements should be used, and the latter to impose limitations on the definitions already made.

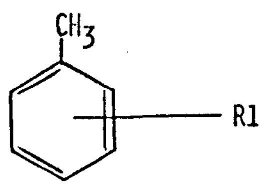
"FREE-TEXT" GENERIC STRUCTURE:



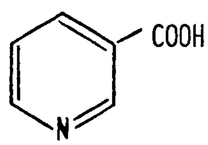
GENSAL NOTATION:

INPUT 6789

SD



R1 = [<2-6>] SD



[<2,4,6>] .

Figure 3.12

3.8.1. Conditions

Complex conditions can be formed, using the Boolean operators

AND, OR and NOT, as shown in Syntax Diagram 19. In executing such conditions the unary operator NOT is evaluated first, followed by AND, and finally OR; conditions in parentheses are evaluated first of all.

Ultimately, all conditions are composed of simple conditions of the form shown in Syntax Diagram 18. All simple conditions have two sides, separated by a relational operator ("=" or "<>", meaning "is" or "is not" respectively), and there are basically three types, which will be discussed separately. Each of them describes a particular arrangement of possible values for the variables in a generic structure, and for this reason only substituents and multipliers that have already been defined may appear in conditions.

3.8.2. Definition Relations

In "definition relations" the right-hand side is a substituent definition, of exactly the same form as is used in assignment statements, though here it may be abbreviated to a "stand-alone" position set, where the chemical nature of the substituent is not relevant. The left-hand side consists simply of a substituent or substituent combination, as in:

a) IF R1 = [4] methyl THEN...

(If R1 is a methyl group in the 4 position then...)

b) IF R2 + R3 = [2/3] THEN...

(If the attachments of the structure formed by the combination of R2 and R3 are at positions 2 and 3 then...)

c) IF R4 = alkyl<1-6> SB (Cl / Br / I) THEN...

(If R4 is an alkyl group of between one and six carbon atoms substituted by Cl, Br or I then...)

3.8.3. Integer Relations

Integer relations have a selector, identifying an integer range on the right-hand side, and the left-hand side can consist of various integer terms such as multipliers and substituents with parameters combined by arithmetic operators.

d)IF R1 C = <1-2> THEN...

(If the carbon count of the **homologous series identifier** defining R1 is in the range 1 to 2 then...)

e) IF R2 E = <2-> THEN...

(If there are two or more double bonds in the **homologous series identifier** defining R2 then...)

f) IF M1 = <2-3> THEN...

(If M1 is either 2 or 3 then...)

g) IF M1 + M2 + R1 C = <4> THEN...

(If the sum of M1 and M2 and the carbon count of R1 is 4 then...)

h) IF R1 C + R2 C = <12-> THEN...

(If the sum of the carbon counts of R1 and R2 is greater than or equal to 12 then...)

i) IF R1 + R2 C = <0-6> THEN...

(If the carbon count of the combined substituent formed by R1 and R2 is less than or equal to 6 then...)

The syntactic and semantic differences between the "+" symbols in examples (h) and (i) above are important. In the former it is an arithmetic operator combining separate integer values, whereas in the latter it combines the two substituents in a substituent combination.

3.8.4. Group Relations

"Group relations" begin with a selector which operates on the remainder of the left-hand side of the condition. If this is a substituent group, then the right-hand side will be a substituent definition, or stand-alone position set, as in the definition relations described above:

j) IF <2-> R1-5 <> H THEN...

(If two or more of the substituents R1, R2, R3, R4 and R5 are not hydrogen, then...)

On the other hand, if the left-hand side is a multiplier group or substituent group and parameter, then the right-hand side will be an integer range:

k) IF <1-3> M1-5 = <4> THEN...

(If 1, 2 or 3 or the multipliers M1, M2, M3, M4, and M5 is equal to 4 then...)

l) IF <1> R1 + R2 , R3 + R4 C = <3> THEN...

(If the carbon count of the group formed by either R1 and R2 or by R3 and R4 (i.e. if the carbon count of 1 of the two substituent combinations) is 3, then...)

3.8.5. IF Statements

In an IF statement, there are two subordinate statements after the condition, one following the delimiter THEN, and the other the delimiter ELSE (though this latter may be omitted). The statement following the THEN is used in those arrangements of the variables that make the condition TRUE, and the following the ELSE (if present) in those that make the condition FALSE.

The statements in the THEN and ELSE parts may be assignment statements, RESTRICT statements (described below), nested IF statements, "empty statements", or groups of statements enclosed within BEGIN and END delimiters (a "compound statement"). Examples of IF statements are:

```
IF R1 = methyl THEN R4 = methyl;

IF R1 = H THEN R2 = H
          ELSE R2 = halogen;

IF R1 = halogen
   THEN IF R1 = [2-3]
           THEN RESTRICT R2 = H;
           ELSE
   ELSE BEGIN
          RESTRICT R2 <> H;
          RESTRICT M1 = <3>
         END
```

There is no semicolon between the statement following the THEN and the ELSE, though the individual statements in a compound statement are separated by semicolons.

In nested IF statements, each ELSE is paired with the most recent unpaired THEN: this can cause problems if there are nested IF statements without an ELSE part. In the last example above, an empty statement is used to provide an ELSE to pair with the second THEN, so that the effect is as intended. This point is further discussed in Section 3.11.1 below.

Clearly, certain IF statements would make semantic nonsense. e.g.:

```
IF R1 = methyl THEN R1 = ethyl
```

Such statements are illegal: if the <u>condition</u> involves a given <u>substituent</u> or <u>multiplier</u> then the <u>statements</u> in the THEN and ELSE parts may not involve that <u>substituent</u> or <u>multiplier</u>. The exception to this rule is that if the <u>condition</u> is concerned with the chemical nature of a <u>substituent</u>, then the <u>statements</u> may be concerned with its position, and vice versa. Thus the following statements are legal:

```
IF R1 = methyl THEN RESTRICT R1 = [2];

IF R2 = [4] THEN R2 = halogen
```

Figures 3.5, 3.8, 3.10 and 3.11 show examples of the use of IF statements from actual patent examples.

3.8.6. RESTRICT Statements

RESTRICT statements are used directly to reduce the possible arrangements of values for the variables in a generic structure. Only those arrangements which allow the condition to be TRUE are possible.

The form of the condition is exactly as in the IF statement, and thus RESTRICT statements appear as in the following examples:

a) RESTRICT R1 = H

H must have been given as a possible value for R1 in its original definition, and this statement eliminates all the other possibilities.

b) RESTRICT M1 + M2 <> <6>

It does not matter what the original definitions of M1 and M2 were; the RESTRICT removes those combinations of possibilities where their sum is 6.

c) RESTRICT R1 C = <2-3>

The carbon count of the **homologous series term** defining R1 is limited to 2 or 3 (which must be a subset of the values given in the original definition).

Examples from actual patents in which RESTRICT statements are used are shown in Figures 3.3, 3.8, 3.10 and 3.11.

3.9. SCOPE OF DEFINITIONS

There is no restriction on the number of different assignment statements each substituent or multiplier appears in, and all the different definitions given are alternative to each other.

When an assignment statement appears in the THEN or ELSE part of an IF statement, the alternatives given there for a substituent or multiplier are added to those given elsewhere (if any) when the condition has the appropriate value. If it is desired to limit the alternatives already given, then a RESTRICT statement should be used.

3.10. LIMITATIONS OF GENSAL

GENSAL has been designed to conform as closely as possible to the forms of expression commonly encountered in patent specifications and abstracts, whilst retaining a sufficient formalism for automatic processing to be possible. However it is not completely comprehensive, and at least in its present form, it is not applicable to certain types of expression found in patents.

In the majority of cases, the expression in question can be reformulated in such a way as to permit encoding in GENSAL: the replacement of the different symbols used for structural and multiplicative variables by standard GENSAL substituents and multipliers is a trivial example of this. Other such limitations of GENSAL are the restrictions that structural variables may be at most doubly connected (Figure 3.11 shows how a small amount of respecifying of structural variables can circumvent this), and that only structural variables may have multipliers applied to them (again in Figure 3.11, this restriction is avoided by making what in the abstract is a multiplied structural constant, a structural variable with a selector applied to its single alternative value.

Certain other expressions found in patent specifications and abstracts cannot be represented in GENSAL at all, however, and several of the examples in the Figures show this.

The Derwent Abstract for the generic structure in Figure 3.7 indicates that certain of the alternatives are "preferred". At present, the only way around this problem is to construct a GENSAL notation for the structure in which only the preferred alternatives are shown; this could be stored alongside the more general notation. A fairly simple extension to GENSAL might allow the sequence of alternatives in a substituent definition expression to be interrupted by the delimiter PREFERABLY, those alternatives following it being the preferred ones. However, this could cause complications in a search system for generic structures encoded in GENSAL.

In the structure shown in Figure 3.10 it is not possible to show the limitations on R5 adequately using GENSAL. The requirement that it be "sterically hindered" cannot be shown at all (unless it be by indicating some branch points in the parameters), and that the majority of the occurrences of R5 must be alkyl<3-> can only be shown by exhaustively enumerating all the possible combinations of R2 and M1 in separate IF statements. This is reasonably satisfactory here, but would not be were there a much larger number of possibilities.

Figure 3.11 illustrates the lack of facilities to show stereochemistry, which is largely a consequence of the absence of stereochemical indicators in the two-dimensional structure representation used by the Feldmann graphics system. If GENSAL were to be used with a graphics system incorporating stereochemistry the syntax of GENSAL could be modified to include stereochemical descriptors in definition elements, treating them in a similar way to position sets.

Whilst GENSAL is not comprehensive, experience in encoding generic structures from patents suggests that in its present form it is capable of representing adequately the vast majority. However, the possibility is discussed in Chapter 6 that some modifications and extensions may need to be made to it.

3.11. THE DESIGN OF GENSAL

3.11.1. Formal Grammar

The initial attempts at the design of GENSAL were based on analogy with Pascal, rather than on a rigorous approach using the formal grammar theory described in Chapter 2. No attempt has been made at formal proof of particular properties of the Grammar, but a number of such properties can be identified by intuitive inspection of the syntax diagrams shown in Appendix 1, or the equivalent Backus-Naur Form production rules shown in Appendix 2.

The Grammar of GENSAL is context-free, the production rules conforming to the requirements of Chomsky Type 2 Grammars (Section 2.1.1). It is unambiguous, and is a member of the class of LL(k) Grammars defined by Lewis and Stearns [83] (Section 2.2.2), which means that it can be parsed "top-down" as well as "bottom-up" (Section 2.2.3).

The syntax for IF statements in GENSAL is similar to that of Pascal and Algol 60, and as was pointed out in Section 3.8.5 can lead to difficulties with nested IF statements where not all have an ELSE part. This is one aspect of the design of Pascal which has been criticised. [128] Algol 68 and certain other languages have explicit terminators for IF statements ("FI" in Algol 68) which help to avoid this problem; this is an aspect of GENSAL syntax which could perhaps usefully be modified.

3.11.2. Non-Determinacy

In most cases, inspection of the next symbol of a sentence in GENSAL is adequate to decide which production is being used: were this always the case the Grammar would be LL(1), and would be deterministic according to the definition of Koranjak and Hopcroft.[86] [The Grammar of Pascal is of this type.] However there are three places in the Grammar of GENSAL where lookahead is required for parsing.

In integer ranges (Syntax Diagram 2) there are three possible productions starting with integer:

<integer>

<integer> - <integer>

and <integer> -

and which of these is being used cannot be decided until up to two further symbols have been examined.

In position sets (Syntax Diagram 6) it is not possible to decide whether or not a position combination is being read until a plus sign is or is not encountered after the first integer.

In substituent groups (Syntax Diagram 12), both productions start with an R delimiter, followed by an integer, and only when the symbol after the integer is examined is it possible to decide which is being used.

In simple conditions (Syntax Diagram 18) the situation is more complex. The possibility of a stand-alone position set instead of a full substituent definition expression on the right hand side of group definition relations and definition relations (Sections 3.8.2 and 3.8.4) means that which of the two is being used may not be decidable until the symbol following a position set is inspected, and as the position set may be of arbitrary length, no limit can be set on the amount of lookahead required.

Wirth [109] has suggested that where there are only a few examples of non-determinacy in the Grammar of a language, these should be handled on an ad hoc basis in the writing of a parser, and this approach has been adopted in the programming of the GENSAL interpreter, described in Chapter 5, where no particular difficulties were encountered with integer ranges and substituent groups.

The programming of the analysis of conditions has not formed part of the present work, and so the problem of arbitrary lookahead has not been considered in detail. However, there appears to be no need to know which path is being followed when a position set is encountered at the start of the right-hand side of a simple condition, and lookahead is therefore unnecessary. There is no reason to suppose that the semantic analysis of simple conditions would require this knowledge at the outset, and so it has been considered that the syntax for simple conditions is satisfactory in its present form. In fact, though strictly speaking lookahead is required in the analysis of integer ranges, the interpreter program that has been written for GENSAL does not actually look ahead at all in performing their analysis, since it records the value of the first integer encountered, and later decides what to do with it.

3.11.3. Security vs. Flexibility

The relationship between the security and the flexibility of formal languages was discussed with reference to the programming languages Ada and Pascal in Sections 2.3.3 and 2.3.4. It was pointed out that the more redundant information is included in sentences in a language, the greater are the possibilities for the checking of self-consistency etc. The requirement in Pascal that all variables be declared with an indication of their type before they are used is an example of this; the type could in many cases be perfectly well deduced from the type of expression used in assignments to the variable in question.

GENSAL has however been designed to conform as closely as possible to the types of expression commonly found in patent specifications, and flexibility rather than security has been the principal aim. This is not to say that interpreter and compiler programs for GENSAL are likely to allow large numbers of errors to pass through undetected, but it makes the task of detecting and reporting such errors much more difficult.

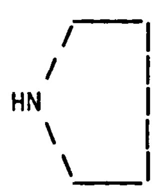
A more secure language for the description of generic structures might require all structural variables to be listed at the outset, with information on the number of connections and the bond orders for each. This information could then be used to check every occurrence of each variable. However this enhanced security would be at the cost of the natural form of expression currently found in GENSAL.

In GENSAL as it stands, information on the connectivity and bond orders of <u>substituents</u> frequently does not become available until well after the variable in question has been introduced, and in some circumstances may not become available at all, leaving the interpreter program to make assumptions about it. For example the GENSAL definition expression

phenyl sb R1

says nothing about the way R1 is connected to the phenyl group, or about its bond orders. When R1 is defined, it is possible that the information is still not given:

R1 = SD



In this case the program should assume that the connection is single, and that the bond order is single also, but a subsequent <u>position set</u> following the **structure diagram** in the manner described in Section 3.6 might give <u>position combinations</u> indicating that the connection was actually double.

Chapter 5 describes the approaches that have been used to detect incompatibilities in the information provided in a GENSAL sentence in the writing of an interpreter for GENSAL. In many cases the incompatibility can only be detected some time after an error has occurred, making recovery from the error very difficult if not impossible.

It is believed that the emphasis on flexibility rather than on security in GENSAL is justified, since most errors can be detected eventually, and it is GENSAL's flexibility, readability and similarity to the language of patent specifications and abstracts, in relation to alternative coding methods for generic structures, that is likely to be a major factor in determining its acceptability to the chemical and patent documentation industries.

CHAPTER 4

THE INTERNAL REPRESENTATION

"Look beneath the surface; let not the several
quality of a thing nor its worth escape thee"

Marcus Aurelius Antoninus (121-180)

The last Chapter gave a description of the formal language GENSAL, which has been designed to encode generic chemical structures from patents in a form which can be processed by computer, yet which remains readily intelligible to a chemist or patent agent. The formalism of its Grammar makes it comparable to a high-level programming language, and thus the program which analyzes it can be thought of as equivalent to a compiler. To extend this analogy further, the internal representation of a generic structure which this program produces can be thought of as being equivalent to the object code produced by a programming language compiler, though unlike the object code for a programming language, the internal representation is a machine-level data structure, rather than a set of machine-level instructions. Furthermore, as the analysis program is expected to operate interactively, it is better described as an **interpreter** than as a compiler.

In a generic structure information system, this internal representation can be used to generate fragments for use in searching, or directly for atom-by-atom tracing in the final stage of a search. In order to enable it to perform these functions satisfactorily, and yet remain in a form which can easily be generated from GENSAL input, a number of features have been incorporated into its design, and these will be described in this chapter.

4.1. REQUIREMENTS FOR THE REPRESENTATION

Chapter 1 discussed the need for a full and unambiguous description of the generic structure, from which fragment screen descriptors of various types could be generated algorithmically, and the reasons for the selection of connection tables as the appropriate basis for this representation.

The purpose of the representation described here is not to store explicitly all the possible specific structures covered by a given generic structure, but rather to contain sufficient information for exhaustive generation of all the specific structures to be possible, even though in most cases such an operation would be pointless, as well as computationally unfeasible where the number of specific structures covered is large, or even infinite.

Since the representation is to be built up from a generic structure input to the computer in GENSAL, the conversion problems are greatly simplified if certain features of the representation mirror features of GENSAL. In particular, as the syntax for the definition of substituents in GENSAL is essentially recursive, the structure of the internal representation should be recursive also.

GENSAL employs Geivandov's concept [17] of a generic structure as a (possibly vestigial) constant part, to which are attached variable parts which can vary in their chemical nature, position of attachment and multiplicity of occurrence, and which may themselves be further substituted by other constant and variable parts, down to any level. At each level, certain of the values for the variable parts may be alternative or additional to each other in complex nested Boolean relationships. This suggests two principal components for the internal representation, one containing information about the chemical nature of the constant and variable parts and the other information about the way in which they are connected together in terms of positions and multiplicity, and the Boolean relationships between them.

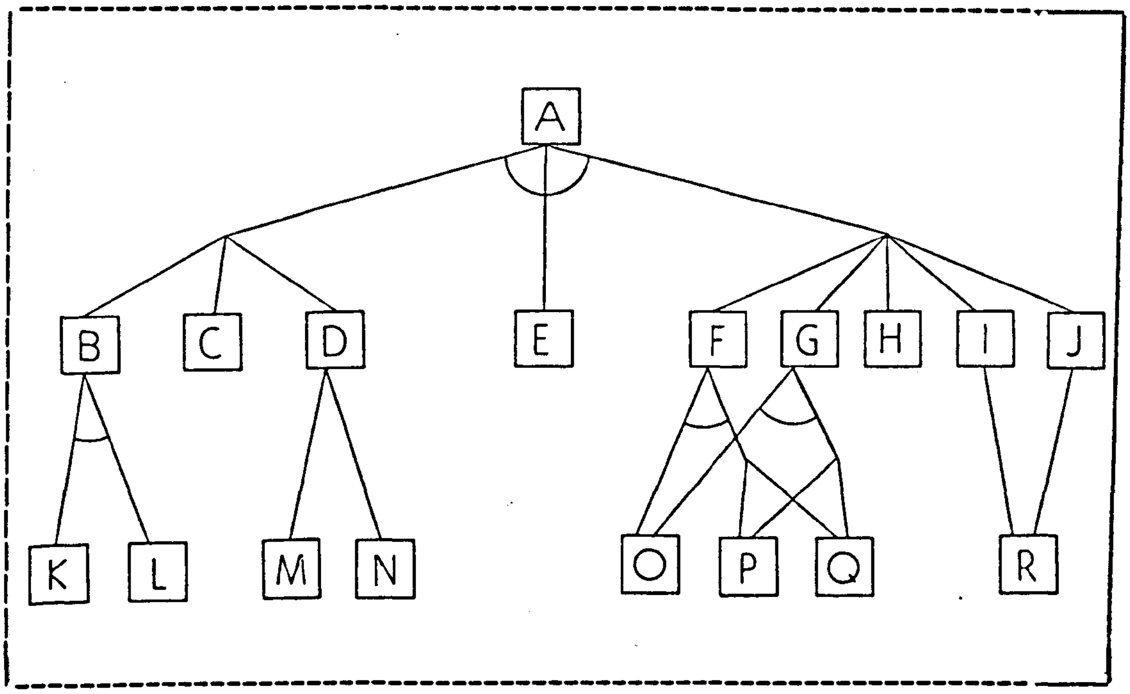


Figure 4.1: A diagrammatic representation of the basic structure of the ECTR showing the **child gates.** Each box represents a **partial structure,** and the lines represent **child gates.** Each hierarchical level of substitution is shown as a separate row of **partial structures.**

Lines meeting at a point connect together **partial structures** which are alternative to each other (OR relationship), and lines meeting at a point that are linked together by an arc connect **partial structures** which are additional to each other (AND relationship).

The GENSAL statements corresponding to this ECTR are shown in Figure 4.2.

The successive levels of further substitution imply a hierarchical relationship between the different parts of the structure, though the exact nature of the hierarchy depends on the way in which the GENSAL description of the structure was constructed, which is to a certain extent arbitrary. Where, as is illustrated in Section 3.7.4., a GENSAL substituent is defined in terms of itself, the hierarchy "loops back" to a higher level and there is no lowest level of substitution; the structure in question is a polymer.

This approach to the storage of polymers has certain conceptual similarities to that developed by the Du Pont company in the 1960's.[182] In that system, each monomer unit is shown in a connection table connected to a dummy central atom, and path tracing procedures are able to pass through this central atom and back into the monomer unit(s).

```
1 :   INPUT 12345
2 :   SD
         R3
          \
           \
            A--R1
           /
          /
         R2

3 :   R1 = B sb K andby L   /  C  /  D sb ( M / N ) ;
4 :   R2 = E ;
5 :   R3 = ( F / G )  sb  O  andby  ( P / Q )  /
6 :           H  / ( I / J ) sb R .
```

Figure 4.2: GENSAL statements corresponding to the ECTR's shown in Figures 4.1 and 4.4

Together the two components of the internal representation can be considered as forming a topological graph - the chemical nature of the various parts of the generic structure being represented in the nodes of the graph, and the information about their connections and relationships in its edges. Since information on the chemical nature of each part is predominantly based on conventional connection tables, the whole is a sort of superconnection table, or connection table of connection tables, and is called an **Extended Connection Table Representation** (ECTR). Within the ECTR each node is called a **partial structure** (PS), and each edge a **gate.** The **gates** are divided into **child gates** and **parent gates,** according to which direction in the hierarchy they point; the graph is thus a directed one. The overall layout of the ECTR for a generic structure, showing the PSs and the **child gates,** is shown in Figure 4.1.

The entire ECTR is held in the main computer memory during its generation because, as further parts of the structure are defined during the course of the GENSAL sentence, it is frequently necessary to refer back to previously-defined parts. Similarly, as fragments are generated, or an atom-by-atom search performed, it is necessary to trace from one PS to another.

4.2. THE PARTIAL STRUCTURE RECORD

Syntax Diagram 10 shows five different paths for a substituent value in GENSAL, and these were discussed in Section 3.3.1. From them it is possible to identify four fundamentally different types of **partial structure,** each requiring a different form of representation in the ECTR.

4.2.1. Specific Partial Structures

These correspond to a single fully-defined structural entity, and are the only type of PS that may be represented by a connection table. They appear in GENSAL substituent values as **structure diagrams** (Section 3.3.1.2), or as **specific nomenclatural terms** (Section 3.3.1.3) which the GENSAL interpreter program translates into connection tables via a dictionary of standard nomenclatural terms.

4.2.2. Generic Partial Structures

These appear in GENSAL substituent values as **homologous series terms** (Section 3.5), with associated **parameter lists.** They are shown in PS records as expanded **parameter lists,** including those parameters implied by the **homologous series term** itself, as well as those given explicitly. For example the term "alkenyl" implies at least one double bond, that would be indicated by the parameter E<1-> in the dictionary. This type of PS is designed to be handled for fragment generation and searching using the chemical grammars developed by Welford.[67, 174]

4.2.3. Unknown Partial Structures

These appear in GENSAL substituent values as a "?" (Section 3.3.1.1.). Clearly, no further information can be stored about their chemical nature, and search algorithms should allow them to be matched against any structural entity.

4.2.4. Other Partial Structures

These cannot be directly associated with any particular structural characteristics, and include expressions such as "electron withdrawing group" or "easily hydrolysed group". They are shown in PS records as a character string, taken from the **other term** in the GENSAL substituent value, and could be used for some sort of text-based searching. Nomenclatural terms not found in the dictionary file used by the GENSAL interpreter program can also be stored in this form.

Table 4.1 summarises the information given in a **partial structure.**

Child Gate			
Parent Gate			
Specific	Generic	Unknown	Other
Connection Table	Parameter List	-	Character String

Table 4.1: Partial Structure Record

4.3. CONNECTION TABLE FORMAT

The connection table used to represent **Specific PSs** is a simple redundant one, each row representing one node, which may be either an atom (in which case the atom type is recorded as a two-letter symbol) or a GENSAL substituent (in which case its name - the "R1", "R2" etc of GENSAL - is recorded, along with the values it can take, in the same format as a **child gate**). The record structure for a connection table row is shown in Table 4.2.

<table>
<tr><td rowspan="2">Atom Type</td><td>Substituent Name</td></tr>
<tr><td>Substituent Values (Child Gate)</td></tr>
<tr><td colspan="2">Charge</td></tr>
<tr><td colspan="2">Number of Hydrogens</td></tr>
<tr><td colspan="2">Six Congeners</td></tr>
</table>

Table 4.2: Connection Table Row

Normally, substituents attached to a **Specific PS** are not explicitly included in the connection table as information about the atoms to which they are connected is stored in the **child gates**. It is only when there is a chain (cyclic or acyclic) of such substituents connected together, as shown below, that it is necessary in order to indicate the order in which they are connected to each other.

```
      C
     / \
    /   \
   C     C----R1
   |     |    |
   |     |    |
   C     C    R2
    \   / \   /
     \ /   \ /
      C     R3
```

The number of attached hydrogen atoms is recorded for each row in order to permit the determination of the positions available for substitution in each PS.

4.3.1. Congener Record

Up to six congeners are possible for each row, this being a restriction derived from the Feldmann structure diagram graphics system used [179] (Section 3.2) and the record structure for each is shown in Table 4.3. Other graphics systems might relax the limitation, though it has not been found a particularly irritating one. For each is recorded a bond order and information about the nature of the connected node. **Fraternal** connections are those to other rows in the same PS: the relevant row number is recorded. **Filial** connections are those to other PSs "lower down" in the ECTR, details of the connection being given in the **child gate.** **Parental** connections are those to other PSs "higher up" in the ECTR, and details are given in the **parent gate.**

In the present implementation, an arbitrary limit of 32 rows is set for each connection table, which is thus the maximum number of non-hydrogen atoms permitted in a structure diagram. However, because the splitting of a generic structure into separate PSs is to a certain extent arbitrary, a large structure diagram can always be divided into two or more smaller ones, and the limit might be different in other implementations.

None	Fraternal	Filial	Parental
-	Row number of connected atom or "NOTFIXED" for variable-position connection	-	-
Bond Order			

Table 4.3: Congener Record

4.3.2. Bond Orders

In the present implementation, the different bond orders used have been derived from the Feldmann system,[179] with some modifications. Fifteen bond types are distinguished, and are shown in Table 4.4

Because the environment (chain or ring) of a particular bond may alter according to which alternative values for a particular structural variable are being considered, and because the possibilities for tautomerisation and aromaticity may change similarly, the finer distinctions between these bond types are not always helpful in generic structures. Ideally, an operational system would permit the user only to distinguish between Single, Double, Triple and "Any" bonds, and would automatically perceive rings, tautomers and aromaticity. Algorithms for such analyses in specific structures have been developed for use in synthesis analysis programs.[183-185]

Chain Single (CS)	Ring Single (RS)	Any Single (S)
Chain Double (CD)	Ring Double (RD)	Any Double (D)
Chain Triple (CT)	Ring Triple (RT)	Any Triple (T)
Chain Tautomeric (TC)	Ring Tautomeric (TR)	Any Bond (A)
Any Chain (C)	Any Ring (R)	
	Ring Alternating (RA)	

Table 4.4: Bond Orders in Connection Tables and Gates

4.4. PARAMETER LIST FORMAT

Welford[67, 174] has described a means in which the parameters applied to a **homologous series term** in a GENSAL sentence can be used to apply constraints to the chemical grammars used for generation and/or recognition of the members of the homologous series. The standard **parameter identifiers** used in GENSAL to constrain such features as atom count, branch points and unsaturations are shown in Table 3.1, and substituents in

parameter lists can be used to indicate interruptions in a chain or ring, or substitutions on it.

The full set of parameters with their values is sufficient, when used to constrain the chemical grammars, to define completely all the possible structures covered. Consequently, the PS record for the **generic** type of PS can consist simply of a list of parameter values (as integer ranges) for all the standard parameters. The non-standard parameters, represented by GENSAL substituents, are treated as substitutions on the **generic PS**, and information about them is given in **child gates**, as described below. However, when generating fragments or path tracing within the ECTR, the information about **children** of **Generic PSs** can used to apply constraints to the chemical grammars.

4.5. CHILD GATE FORMAT

Child gates indicate the connections from one PS (called the **parent PS**) to those lower down in the hierarchy to which it is connected. There may be connections to several **child PSs**, which can be additional or alternative to each other. Each **child gate** therefore describes a "one-to-many" relationship, though over the ECTR as a whole the **child gates** between successive levels of the hierarchy describe a "many-to-many" relationship, as can be seen from Figure 4.1.

In order to show the Boolean relationships between the various

child PSs, as well as information on positions of attachment, bond orders etc., the internal structure of **child gates** is quite complicated. Each **child gate** is essentially a tree, with two different types of node; the root of the tree is attached to the **parent PS,** and the nodes are arranged in layers called **bars.** Each **bar** contains only one type of node, and is either a **combination bar** containing **combination bar** item nodes, which are in AND relationship, or is an **alternative bar** containing **alternative bar** item nodes in OR relationship. The two types of **bar** follow one another alternately.

For reasons of convenience, based on the precedence of operators in GENSAL expressions (Section 3.7), the information on positions, multiplicity, bond orders etc. is stored in the **combination bars,** which form the top and bottom **bars** of each **child gate.** The number of intervening layers depends upon the complexity of the Boolean relationships, as indicated by the number of pairs of parentheses in the GENSAL expression. It is possible for there to be only a single **combination bar** in a gate.

Both types of **bar** are constructed as linked lists of items, which are alternative to each other in **alternative bars,** and additional to each other in **combination bars.** The **child gate** field of a PS record (Table 4.1) is a pointer to the first item in the top **combination bar,** and each bottom **combination bar** points to a **child PS.**

<table>
<tr><td colspan="2">Positions in Parent PS</td></tr>
<tr><td colspan="2">Multiplicity of Occurrence</td></tr>
<tr><td>Bottombar</td><td>Not Bottombar</td></tr>
<tr><td>Positions in Child PS</td><td rowspan="3">Pointer to
Alternative Bar
(next layer)</td></tr>
<tr><td>Bond Order</td></tr>
<tr><td>Pointer to Child PS</td></tr>
<tr><td colspan="2">Pointer to next item in Combination Bar list</td></tr>
</table>

Table 4.5 : Item in **combination bar** of **child gate.**

4.5.1. Combination Bars

The record structure for a **combination bar** item, shown in Table 4.5, indicates that it may take one of two possible forms, according to whether or not it is located in the bottom **bar** of the **gate.** For both **bottombar** and non-**bottombar** forms, information is given about the positions of attachment in the **parent PS,** and the multiplicity of occurrence in these positions; there is also a pointer to the next item in the **combination bar.**

For non-**bottombar** items, there is a pointer to the first item in the **alternative bar** of the next layer, and the position and multiplicity information given applies to all the alternatives in this **alternative bar.**

For **bottombar** items, no such alternatives are possible, and a pointer is given to the appropriate **child PS** record, along with information about the positions in the **child PS** at which the attachment may be made, and the order of the connecting bond.

Position information can be taken from explicit GENSAL position sets (if present) or calculated from those positions available for substitution; multiplicity information can be taken from a GENSAL selector or from the definition of a multiplier or, if the **child** has been specified in a **parameter list** for a **homologous series term** (Section 3.5), from the values given for that parameter.

If there are several **combination bars** in a **gate** then the position information may in each layer more closely specify the positions of attachment; the positions specified lowest down the gate are those that actually define the point of attachment in the **parent PS.** Not every layer necessarily has a value for the positions of attachment in the **parent PS,** but the top **bar** will always specify positions; others will only do so if there is a position set given in the GENSAL expression.

On the other hand, multiplicity information is given in every layer (and is assumed to be 1 if there is no other information), and the values in successive layers are multiplied together in the manner of Section 3.7.1.

4.5.2. Alternative Bars

These have a much simpler structure than **combination bars,** and the record structure for an **alternative bar** item is shown in Table 4.6. All the information about each alternative in the list is given in the **combination bar** pointed to.

Pointer to Combination Bar (next layer)
Pointer to next item in Alternative Bar list

Table 4.6: Item in **alternative bar** of **child gate.**

Figure 4.3 illustrates the internal structure of a single **child gate** for a moderately complicated GENSAL expression.

4.6. PARENT GATE FORMAT

The structure of **parent gates** is very much simpler than that of **child gates,** as none of the information on the Boolean relationships between the various **child PSs** is stored in them. In fact, all the information contained in a **parent gate** is also contained in the corresponding **child gates,** and the purpose of **parent gates** is simply to allow path tracing within the ECTR to

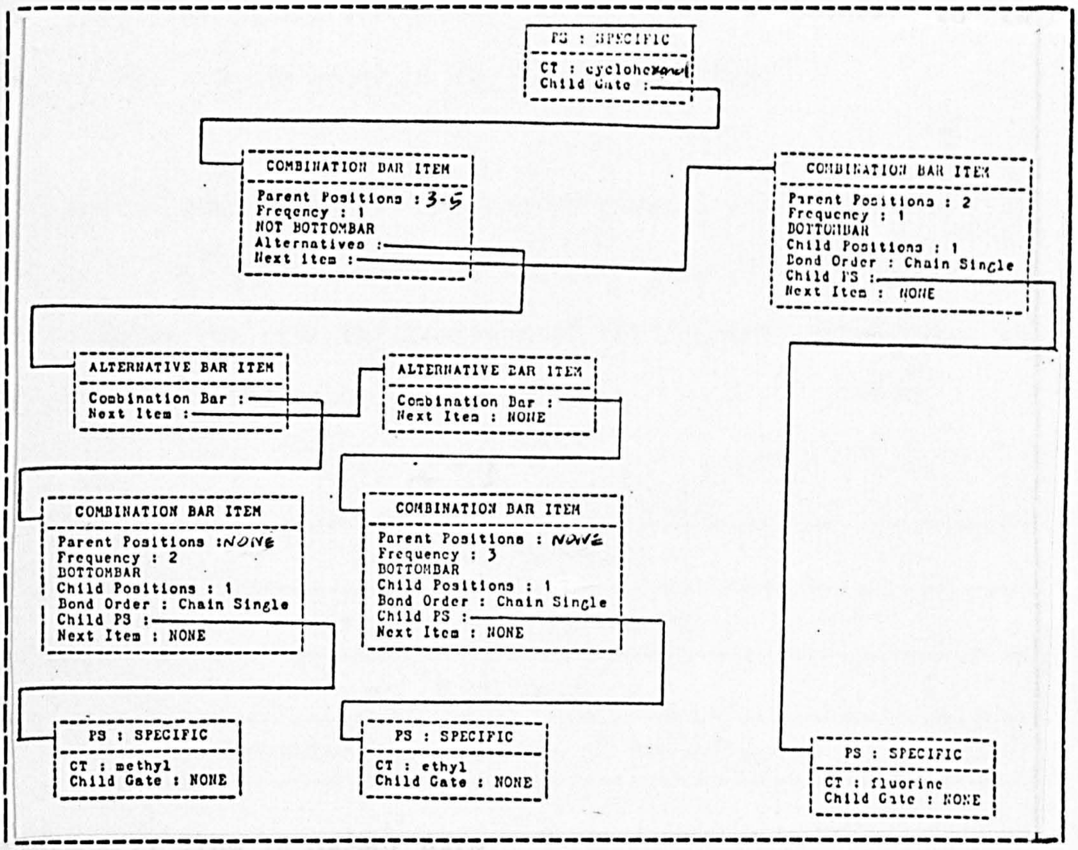


Figure 4.3: A diagrammatic representation of the structure of the child gate corresponding to the GENSAL expression:
cyclohexanol SB [3-5] (<2> methyl / <3> ethyl) & [2] F
which means that cyclohexanol is substituted in positions 3, 4, and/or 5 by either two methyl groups or three ethyl groups, and in addition to these by one fluorine in position 2.

take place from **child PS** to **parent PS** as well in the other direction; the redundancy of the information in the **parent gates** is compensated for by the substantial enhancements in path tracing ability.

Like the two types of **bar** in **child gates, parent gates** are implemented as a linked list of items, each item referring to a different possible **parent PS** for the **child** in question. The record structure is illustrated in Table 4.7. For each possible **parent PS,** the possible positions for connection in both the

child and the **parent** are given, along with a pointer to the **parent PS**, and the order of the connecting bond.

The **parent gate** field of a PS record gives a pointer to the first item in a linked list of **parent gate** items. Figure 4.4 illustrates the overall structure of the **parent gates** for the generic structure shown in Figure 4.2.

Positions in Child PS
Positions in Parent PS
Bond Order
Pointer to next item in Parent Gate

Table 4.7: Item in **parent gate.**

4.7. REPRESENTATION OF CONDITIONS AND RESTRICTIONS

The ECTR described in this chapter makes no provision for incorporating the information given in GENSAL "IF" and "RESTRICT" statements, nor for distinguishing between the five different assignment operators that can be used to indicate independent or non-independent values for substituents or multipliers in selected group assignment statements (Section 3.4.2.).

These features of GENSAL, which mirror many of the expressions found in chemical patent specifications, are used to limit the variety of possible specific compounds covered by a generic structure, by restricting the co-occurrence of particular alternatives in substituent definitions, etc. The present form of the ECTR may thus describe a greater variety of specific compounds than is actually warranted, and the limitations imposed by "IF" and "RESTRICT" statements could be implemented by indicating which of the possibilities in the ECTR should not co-occur. This might be achieved by applying some sort of selective "lock" to the **gates,** though the way in which this might be represented in the computer has yet to be determined.

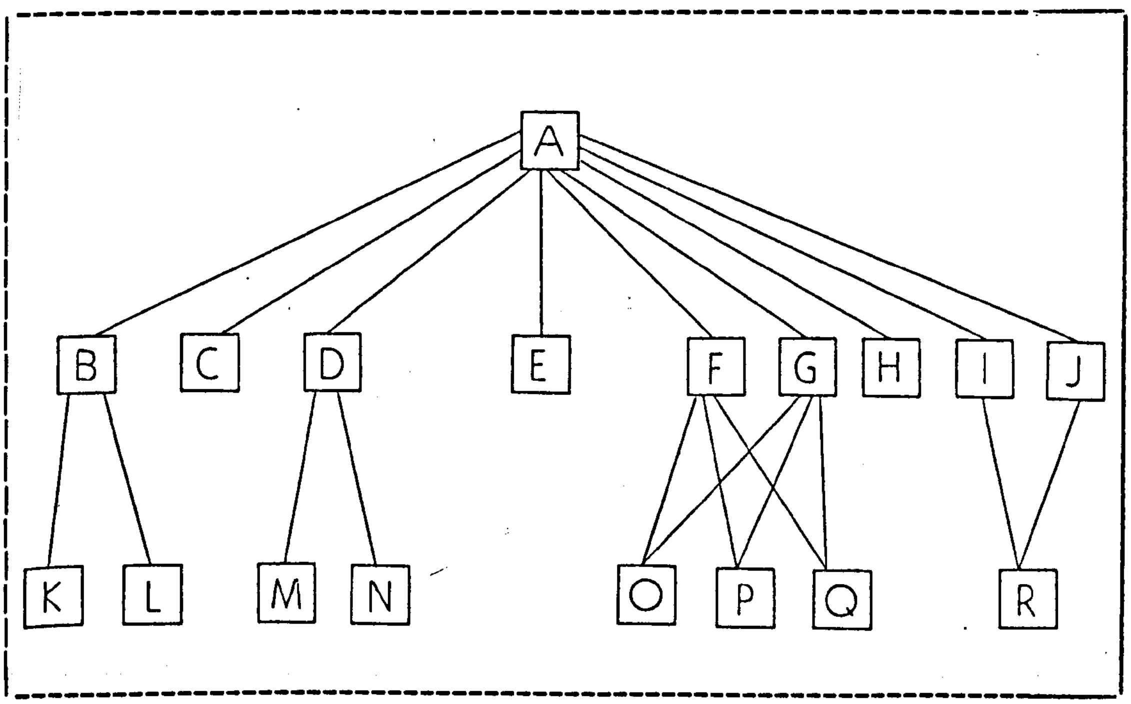


Figure 4.4: A diagrammatic representation of the ECTR, showing the **parent gates,** for the generic structure of Figure 4.2.

4.8. THE ECTR AND OTHER REPRESENTATIONS

Silk [29] has drawn attention to the similarity between a Markush structure and a nested Boolean expression, and suggested that the Boolean relationships could be incorporated into a notation-based representation for Generic structures. The ECTR, also exploits this similarity with the successive layers of **bars in child gates** representing the nested Boolean relationships, though the PSs are represented by connection tables, rather than notation strings.

An approach much closer to that described here has been proposed by Fugmann et. al. [186] It is based on an application to generic structures of the topological graphs used to represent concept relationships in the TOSAR (Topological Representation of Synthetic and Analytical Relations of Concepts) system developed by IDC. [187] Figure 4.5 shows the representation of a generic structure as a TOSAR graph which, like the structure of **child gates** in the ECTR, indicates AND and OR relationships between the different parts of the structure by means of two types of node in the graph (shown as open circles for OR and dots for AND). Fugmann et al. warn however, that the path tracing algorithms used for TOSAR graphs may be extremely expensive where tracing in generic structures is concerned.

The Chemical Abstracts Registry III System [188] employs a mechanism for compiling several partial connection tables to describe a larger specific structure. This involves the replacement of each ring system in a structure by a unique ring

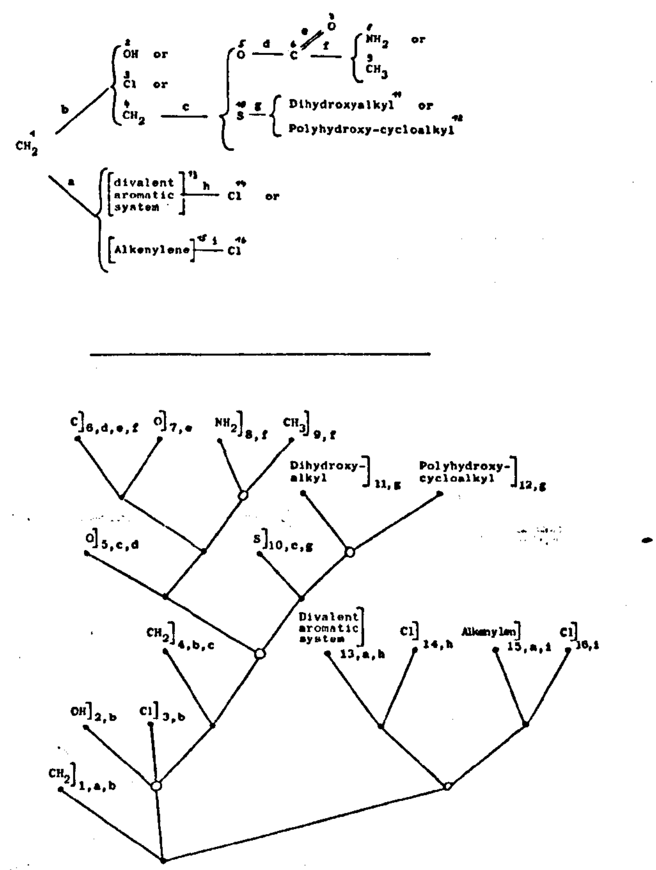


Figure 4.5: A generic structure represented as a TOSAR graph (From Fugmann et al.[186]).

identifier, which gives access to a separate file of connection tables for ring systems. This has the advantage of saving space, since only one connection table need be stored for each ring system, irrespective of the number of structures in which it occurs, and also allows cross-referencing between structures having ring systems in common, and acts as an aid to the automatic generation of systematic names [189] which are based on parent ring systems. The method is not used however as a means of describing generic structures.

4.9. IMPLEMENTATION OF THE ECTR

The ECTR has been implemented using the data structures of the programming language Pascal. Because of the variable total size of the ECTR, which depends upon both the number and nature of the PSs, and its extensive use of linked lists, it is held entirely in dynamic storage, and access to its various parts is achieved using pointer variables.

Tenenbaum and Augenstein [121] have discussed the use of dynamic variables in Pascal, and more general problems of the implementation of recursive data structures have been considered by Hoare [190] and by Burton. [191]

4.9.1. The Partial Structure Record

The Pascal TYPE declaration for a single PS record is a variant record, the variants corresponding to the four different types of PS found in the ECTR:

```
PTRPSTYPE   = ^PSTYPE;

TPSVARIETY  =(DUMMY, UNKNOWN, SPECIFIC, GENERIC, OTHER);

PSTYPE      = RECORD
                VISITED          : BOOLEAN;
                CHILDGATE        : PCOMBINLIST;
                PARENTGATE       : PPARENTLIST;
                CASE PSVARIETY : TPSVARIETY OF
                 DUMMY           : SUBSTNAME   : SUBSTITUENT);
                 UNKNOWN         : ();
                 SPECIFIC        : (CT         : CTTYPE);
                 GENERIC         : (PARAMLIST  : TPARAMLIST);
                 OTHER           : (TERM       : STRING32)
              END;
```

In this record, the VISITED field can be used as an aid to path tracing in the ECTR, and the other two invariant fields give access to the **child** and **parent gates** respectively. Of the variant fields, that for a DUMMY PS is used only for housekeeping operations in the GENSAL interpreter program, no information can be stored for UNKNOWN PSs and the record TYPEs for the other three varieties of PS are given below.

One of the advantages of using a variant record is that it is only necessary to set aside the amount of computer storage actually required for the particular type of PS in question.

4.9.1.1. Connection Tables

The Pascal TYPE declarations are

```
CTTYPE      = ARRAY[1..MAXCT] OF ^ROW;

STRING2     = PACKED ARRAY[1..2] OF CHAR;
```

```
NUMCONGENERS=0..MAXCONGENERS;

SUBSTITUENT = 0..MAXVARS;

ROW         = RECORD
               CHARGE    : -9..9;
               HYDROGENS : NUMCONGENERS;
               CONGENERS : CONGARRAY;
               CASE ATOMICROW : BOOLEAN OF
                TRUE  : (ATOM   : STRING2);
                FALSE : (NAME   : SUBSTITUENT;
                         VALUES : PCOMBINLIST)
              END;
```

The connection table consists of an array of pointers to individual ROWs of the connection table; this is also a space-saving measure, as it means that there is no requirement to set aside large amounts of space to store empty connection table ROWs. MAXCT is a CONSTant giving the maximum permissible number of ROWs, currently 32.

```
RELATIVES   =(NONE, FRATERNAL, PARENTAL, FILIAL);

ATOMNUMBER  = NOTFIXED..MAXCT;

CONGARRAY   = ARRAY [1..MAXCONGENERS] OF
              RECORD
               BOND : BONDORDER;
               CASE RELATIONSHIP : RELATIVES OF
                NONE,
                PARENTAL,
                FILIAL    : ();
                FRATERNAL : (ROWNUM    : ATOMNUMBER)
               END;
```

In the array of congeners for each ROW in the connection table, the number of congeners permitted is controlled by the CONSTant MAXCONGENERS, which is currently 6. A variant record distinguishes between the different types of connection, and in the ROWNUM recorded for FRATERNAL connections, a value of NOTFIXED (which is a CONSTant equal to 0) indicates variable-position connection. The available bond orders are

```
  BONDORDER =(NOTSPECIFIED, ANY, CHAIN, RING, SINGLE, DOUBLE,
              TRIPLE, CHAISING, CHAIDOUB, CHAITRIP, CHAITAUT,
              RINGSING, RINGDOUB, RINGTRIP, AROMATIC, RINGTAUT);
```

4.9.1.2. Parameter Lists

This consists an array of integer range records, one for each parameter, each consisting of a linked list of pairs of integers (being the lower and upper bounds of each subrange) plus a single integer to indicate the lower end of an unbounded top range:

```
     PDOUBLIST   = ^DOUBLIST;

     DOUBLIST    = RECORD
                    FIRST,
                    SECOND : INTEGER;
                    NEXT   : PDOUBLIST
                   END;

     INTRECORD   = RECORD
                    SUBRANGES : PDOUBLIST;
                    TOPRANGE  : INTEGER
                   END;
```

If there is no unbounded top range, then the TOPRANGE field is set to NOTSET, a CONSTant of value -1

The declarations for the parameter list array are thus

```
     TPARAMETERS =(ATOMCOUNT, TBRANCH, QBRANCH, EUNSATURATION,
                  YUNSATURATION, RINGCOUNT, RINGATOMS,
                  RINGSUBSTITUTION, RINGFUSIONS,
                  RINGAROMATIC, HETEROATOM);

     TPARAMLIST  = ARRAY[TPARAMETERS] OF INTRECORD;
```

4.9.1.3. Other Terms

This is simply a character string, currently of 32 characters:

```
STRING32    = PACKED ARRAY[1..32] OF CHAR;
```

4.9.2. Child Gate Record

The Pascal TYPE declarations for **combination** and **alternative bars** are:

```
PCOMBINLIST = ^COMBINLIST;

COMBINLIST  = RECORD
               PARENTPOSITIONS : PTGROUPMEMS;
               FREQUENCY       : INTRECORD;
               NEXT            : PCOMBINLIST;
               CASE BOTTOMBAR  : BOOLEAN OF
                TRUE : (CHILDPS        : PTRPSTYPE;
                        CHILDPOSITIONS : TGROUPMEMS;
                        CONNBONDS      : TCONNBONDS);
                FALSE: (ALTERNATIVES   : PALTERNLIST)
              END;

PALTERNLIST = ^ALTERNLIST;

ALTERNLIST  = RECORD
               COMBINATION : PCOMBINLIST;
               NEXT        : PALTERNLIST
              END;
```

and they can be seen to correspond with the record formats shown in Tables 4.5 and 4.6. The FREQUENCY fields have the same TYPE as the elements of the parameter list record shown above, and the position set fields are as follows:

```
INTEGSET    = SET OF 0..MAXVARS;
```

```
TGROUPMEMS  = RECORD
               CASE COMBINED      : BOOLEAN OF
                TRUE  : (COMBMEMS : PDOUBLIST);
                FALSE : (MEMBERS  : INTEGSET)
               END;

PTGROUPMEMS = ^TGROUPMEMS;
```

The BOOLEAN tag field for the variant record type TGROUPMEMS distinguishes between position sets for singly-connected substitution (COMBINED = FALSE), which are represented simply by an integer set, and position sets for doubly-connected substitution (COMBINED = TRUE), represented by a linked list of pairs of integers.

The PARENTPOSITIONS field of the **combination bar** item is a pointer to a TGROUPMEMS record, rather than a TGROUPMEMS record itself because, as was stated in Section 4.5.1., not all **combination bar** items have a record of positions in the **parent PS**, and for those that do not the PARENTPOSITIONS field can be set to NIL. Furthermore, the use of a pointer allows several different **combination bar** items to share the same PARENTPOSITIONS^ record.

In contrast, there will always be information in the CHILDPOSITIONS field where BOTTOMBAR is TRUE, and thus this is a TGROUPMEMS record, and not a pointer to one.

The CONNBONDS field, showing the bond orders for the connection, is another variant record:

```
TCONNS      = NOTSET..2;

TCONNBONDS  = RECORD
               CASE CONNECTIONS : TCONNS OF
                NOTSET,
                0               : ();
                1               : (BOND  : BONDORDER);
                2               : (BONDA,
                                   BONDB : BONDORDER)
               END;
```

The tag-field indicates whether the substituent is unconnected (CONNECTIONS = 0), singly- or doubly-connected, an appropriate number of bond orders being given in each case. The NOTSET value for the tag field is used only in the setting up of the ECTR in the GENSAL interpreter, when it may not initially be known what the connections are.

4.9.3. Parent Gate Record

This is implemented as a simple linked list of records, corresponding to Table 4.6:

```
PPARENTLIST = ^PARENTLIST;

PARENTLIST  = RECORD
               CHILDPOSITIONS,
               PARENTPOSITIONS : TGROUPMEMS;
               PARENTPS        : PTRPSTYPE;
               CONNBONDS       : TCONNBONDS;
               NEXT            : PPARENTLIST
              END;
```

4.9.4. Space Requirements

As a complete and unambiguous representation of a generic structure, the ECTR is expensive in its storage requirements. For this reason, it is not intended that it should be stored permanently as a record of the structure. It would in any case be difficult to write the ECTR to a file and read it back into the computer on account of its complicated nature as a network of pointers.

It is expected that the ECTR would be built up during interactive input of a generic structure for a database of such structures, and then immediately used for the generation of fragment descriptors which would be stored for use in the first stages of searching. The ECTR would then be discarded, and could subsequently be regenerated from the stored GENSAL statements only if required for atom-by-atom matching in the final stage of a search.

The actual amount of core storage occupied by the ECTR depends, of course, on the size and complexity of the generic structure it represents. One containing a large number of different alternative values for a structural variable, all of which would have to be stored as separate PSs, would occupy much more space than one with only a few alternatives; the number of atoms in each connection table is also an important factor. The GENSAL interpreter program described in Chapter 5 is able to count up the amount of space being used, and the ECTRs for generic structures from patents that have been processed by this program have ranged in size from 1156 to 10 674 PR1ME 750 16-bit words. The Pascal implementation used for the interpreter [138] allows 16 segments of 64 kwords each for the storage of dynamic variables, making a total of over one million 16-bit words available, though other implementations might not be so generous.

CHAPTER 5

AN INTERPRETER FOR GENSAL

"This is the interpretation of the thing"

Daniel, Ch. 5, Vs. 26

This Chapter describes an interpreter program, written in the Pascal language, which implements a subset of the GENSAL generic structure description language, and which is upwards compatible with the full language, as described in Chapter 3.

The interpreter program performs syntactic and semantic analysis on sentences in GENSAL, and generates an Extended Connection Table Representation (ECTR) of the structure described.

It is implemented as a separately-compiled procedure of a program called GENPROG, which is a prototype generic structure storage and retrieval system under joint development by the author and Welford. Appendix 3 is a listing of the interpreter, **procedure** INTERPRET, and Appendix 4 contains a line-number index to the subordinate **procedures** and **functions** within it. Appendix 5 is a listing of the **const, type** and **var** declarations that are global to GENPROG, with the addition of those **procedures** and **functions** called by INTERPRET which are also called by other parts of GENPROG.

Pascal programs are sufficiently clear to be largely self-documenting; comments at the start of each **procedure** and **function** indicate the routine's basic purpose, and list the calls to it. This Chapter gives an overall view of the strategies involved in the analysis of GENSAL sentences, and the build-up of the ECTR, with particular notes on the capabilities and limitations of the interpreter, and on the error messages given by the program. It is not intended by itself to give a complete understanding of the workings of the program, for which it should be read in conjunction with a thorough study of the program listings in Appendices 3 and 5.

Appendix 6 shows a sample interpreter session, illustrating the input of a generic structure from a patent.

5.1. INVOCATION OF THE INTERPRETER

The main part of GENPROG processes a simple command language which allows the user to invoke the interpreter, and also to perform a variety of other functions. These include filing and retrieving of structures processed by the interpreter, opening and closing of files of diagnostic information on the program, adding to a dictionary of nomenclatural terms and invoking a simple interactive editor program for structures encoded in GENSAL, which has been written by Kinsella.[192] Ultimately it is expected also to have facilities for searching a database of generic structures, using GENSAL-encoded query structures, and printing search results in a variety of formats.

The interpreter may be invoked in one of two modes: interactive mode, in which each new line of GENSAL is typed at the terminal, and non-interactive mode in which previously stored lines of GENSAL are processed. Such lines might have been stored in a file after a previous session, or be the result of editing a structure.

The lines of GENSAL are stored as a linked list of lines, with pointers to both the preceding and following lines; connection tables, representing structure diagrams within GENSAL, are encoded so that they may also be stored as character strings, as discussed in Section 5.5.3. below. The forward and backward pointers in the linked list are intended to facilitate operations in the editor module of GENPROG.[192]

5.2. LEXICAL ANALYSIS

This is the first stage of analysis in any compiler or interpreter [80, 81] and is the process by which the input string of characters is divided up into **tokens**, each representing one terminal symbol. In INTERPRET, the variable TOKEN holds the most recently-identified token for examination by the syntax analysis routines, and it is updated by the **procedures** NEXTTOKEN and LOOKAHEAD, both of which call **procedure** GETTOKEN, the lexical analyser itself.

Three different types of token are identified: GENSAL delimiter words and symbols, nomenclatural terms, and integers; the subordinate **procedures** and **functions** in GETTOKEN determine which of these is present. This is done by moving the pointer N along the global variable BUFFER, which contains an upper-cased version of the last line read.

This arrangement means that lower-case letters may be used in the input, but they are treated as if they were upper-case; the user may adopt his own conventions as to the use of lower-case letters for nomenclatural terms, or delimiter words etc. In addition, each line of input may be edited using backspacing etc. before it is processed. In the Pascal implementation used [138] a non-standard extension to the standard **procedure** READLN allows entire **packed arrays of char** to be read in a single operation, the right-hand end of the array being space-filled if necessary, and an extra variable returning the number of characters actually read.

When the end of the line is reached, **procedure** READLINE obtains a new one from the terminal, adding it to the linked list of lines, if the interpreter is operating interactively, or obtains it from the existing linked list, if the interpreter is operating non-interactively. If there are no more lines in the linked list to be read, then the interpreter automatically swops to interactive mode, and in interactive mode, the user is able to exit from the interpreter by entering a blank line.

5.3. SYNTAX ANALYSIS

The basic approach used for syntax analysis is that of top-down, recursive-descent parsing, as described by Wirth.[118] No single part of **procedure** INTERPRET is entirely concerned with syntax analysis, since the **procedures** and **functions** which carry it out are also concerned with semantic analysis and ECTR generation.

The analysis of structure description (Syntax Diagram 21) takes place in the body of **procedure** INTERPRET, and the analysis of statements (Syntax Diagram 20) in **procedure** STATEMENT. Separate **procedures** exist for the analysis of assignment statements (Syntax Diagram 17), RESTRICT statements, IF statements and compound statements, the last two being of necessity mutually-recursive with **procedure** STATEMENT.

A group of nested **procedures** analyse substituent definition expressions (Syntax Diagram 15), these being **procedure** ALTNVLIST, which encloses **procedures** ALTNTVE and ELEMENT (analysing a definition element (Syntax Diagram 14)) which recursively calls **procedure** ALTNVLIST.

Conditions (Syntax Diagram 19) have not been implemented as part of the present work, and **procedure** CONDITION simply accepts any sequence of tokens until an appropriate terminator is encountered. This means that, whilst IF and RESTRICT statements are not actually implemented, no errors are generated by their inclusion. A **boolean** flag, CONDITIONSPRESENT, controls the printing of a warning message at the end of structures containing conditions.

A number of other **procedures** carry out syntax analysis on particular syntactic constructs in GENSAL. These are **procedures** INTEGERRANGE, SELECTOR, POSITIONSET, PARAMETERLIST and SUBSTGROUP.

5.4. ERROR HANDLING

The program detects four different types of error, printing appropriate messages at the user terminal. In each case the error message required is obtained from an external file, ERRORMSGS and is printed by **procedure** WRITEMESSAGE, with the possible inclusion of some information on bond orders, atom numbers etc., if relevant. The available messages are listed in Appendix 7, and the sample session shown in Appendix 6 illustrates several of them.

5.4.1. Program Errors

A limited number of checks are performed by the program on its own working, and any error detected causes the user to be ejected from the interpreter, with display of a unique error number.

5.4.2. Structure Diagram Errors

These are errors detected during the processing of **structure diagrams**, and relate to such matters as illegal valencies etc. If any are detected, the **structure diagram** is rejected and the user required to correct it before processing can continue.

Structure diagram processing is more fully described in Section 5.5 below.

5.4.3. "Immediate" Errors

These are errors relating to invalid tokens in the GENSAL input, and they are handled by **procedure** ERROR. All syntax errors fall into this class, as do certain semantic errors.

A line of arrows is drawn under the offending token in the GENSAL input line currently being processed, followed by the error message. The remainder of the input line is ignored, and in interactive mode the user is invited to continue the GENSAL input starting with a replacement for the erroneous token. In non-interactive mode the user is ejected from the interpreter.

5.4.4. "Delayed" Errors ("Failures")

This type of error is not detected until processing has continued for some time after the token which causes it has been obtained by the lexical analyser, and it is called a **failure. Failures** relate to such matters as incompatible bond types, and are handled by **procedure** FAILURE. In all circumstances the user is ejected from the interpreter.

5.5. STRUCTURE DIAGRAM PROCESSING

As was stated in Chapter 3, the graphics system used for the input of **structure diagrams** in GENSAL is intended to be implementation-dependent, and in the present work a modification of the structure generation and display program written by Feldmann and others [179] is being used. This consists of some 4000 lines of Fortran, and is implemented as an EXTERNal **procedure** of GENPROG. In order to avoid the complexities of attempting to to link the COMMON blocks used by the Feldmann program for storage of the connection table it uses with the global variables of GENPROG, the connection table is transferred to and from the Feldmann program via a scratch disc file.

The connectivity and bonding tables in the Feldmann program are separate, and are read into the GENPROG global **arrays** FELDCT and FELDBD by **procedure** READFELDMANN. **Procedure** PROCESSCT then reformats them into the connection table format used by the ECTR and described in Section 4.9.1.1.

5.5.1. The Feldmann Program

The principal modifications made to the Feldmann program have been to allow the identification of a node in the diagram as a GENSAL substituent, or as an "apical" connection (*) or as a "variable-position" connection (#) as well as as an atom of a particular element, and to allow multipliers to be applied to a particular node.

In addition to this, the maximum number of nodes permitted has been reduced from 100 to 32, and upon exiting from the Feldmann program all "default" bonds are replaced by either chain or ring single bonds, depending upon their environment.

Some slight changes have also been made to the bond types permitted, and to the symbols used to represent them in the diagrams, and routines have been written to output the connectivity and bonding tables to a scratch file, and to read them back again.

5.5.2. Procedure PROCESSCT

This **procedure** is applicable only to the Feldmann graphics system, but is virtually the only routine in the interpreter to be so, and thus is the only one that would require replacement were a different graphics system to be incorporated.

Since the Feldmann program carries out very few checks on atom valencies etc., such checks are done by PROCESSCT, which uses **procedure** REJECT to handle any errors detected.

The Feldmann connectivity table in FELDCT is examined line by line, but only nodes representing atoms (except hydrogen) and certain substituents are added to the ECTR-format connection table (Section 4.9.1.1). **Procedure** HNUMBER is able to calculate the number of hydrogens (equivalent to positions available for further substitution) on each atom for common elements, obtaining the permissible valencies from an external file, VALENCYFILE.

The bond orders are represented by an enumerated **type** which is so arranged that the ordinal values correspond to the integers used for the bond types in the Feldmann program. Since "default" bonds are removed from the structure diagram, NOTSPECIFIED bonds cannot appear in connection tables.

PROCESSCT checks that the connectivity of each substituent is compatible with its connectivity in any previous appearances, rejecting the **structure diagram** if it is not.

If a **structure diagram** is rejected, the lines of the ECTR connection table are DISPOSEd, and in interactive mode the user is returned to the Feldmann program to correct it; in non-interactive mode, the user is ejected from the interpreter. Otherwise, **procedure** GETPOSNS is used to determine the connectivity, bond order(s) and possible position(s) of attachment of each substituent in the diagram, removing from the connection table those substituents that are attached only to atoms.

5.5.3. Storage of GENSAL Structure Diagrams

In order to allow the **structure diagrams** occuring in GENSAL sentences to be held in the same format as text lines of GENSAL, the Feldmann-format connection table is encoded as a character string (using the Pascal CHR **function** for the integers in the connection table). The conversion is carried out by the **procedures** ENCODECT and DECODECT.

Since it is the Feldmann-format connection table which is used for this, the lines of GENSAL stored in files etc. include **structure diagrams** in Feldmann connection table format. A possible minor enhancement to the program would be to remove this

Feldmann dependency, and thus make it easier to use the interpreter program with other structure graphics systems. This could however leave problems with graphics systems having differing requirements for the storage of 2-dimensional atomic co-ordinate data; the Feldmann program retains no such information, but recalculates co-ordinates every time the diagram is redrawn.

5.6. SUBSTITUENT DECLARATIONS

The program maintains a record of the substituents declared (introduced) and defined during the course of a GENSAL sentence. This allows it to check firstly that all declared substituents are defined somewhere (**procedure** CHECKALLDONE), secondly that only declared substituents are defined, and thirdly that all declarations of a given substituent are compatible in matters of connectivity and bond order(s). GENSAL substituents can be declared in one of four ways:

(a) in structure diagrams

(b) as a user-defined parameter to a homologous series term

(c) as a value in the definition of another substituent

(d) in copying a definition containing a declaration as in (c) above. (This last is internal to the program, and not apparent to the user.)

In each case, an entry is made by **procedure** DECLARESUBST in a table of substituent declarations, RDECLARATIONTABLE, which is an array of linked lists, one for each **substituent.** Each new declaration of a **substituent** is recorded as a new item in the appropriate list, which contains information about the declaration relating to such matters as the **partial structure** in which it occurs, the position(s) at which the substituent can be attached and the order(s) of the connecting bond(s).

If it is found that a substituent being declared has already been defined, then the values with which it was defined are copied into the **child gate** of the **partial structure** in which the new declaration occurs. This is done by the mutually-recursive **procedures** COPYCOMBAR and COPYALTBAR which copy **bars** of **child gates.** "Absolute" definitions of each substituent are held in the elements of an array called RDEFINITIONTABLE, in order that the definitions copied are independent of the environment (positions of attachment etc.) in which the substituent in question had previously appeared.

Not all of the information for entries in RDECLARATIONTABLE is available at the time the declaration is made, and missing items are filled in later.

Where one substituent is defined in terms of another, as in

R1 = R2 sb methyl

there may be further substitution to attach to the substituent given as a substituent value. Because, if this new substituent has not yet been defined, no **partial structure** exists to which a **child** gate can be attached, a DUMMY **partial structure** is created to represent the substituent, and the FURTHERSUB field of the entry in RDECLARATIONTABLE points to this DUMMY **partial structure**. When the substituent in question is defined, the further substitutions on it can be copied onto the **partial structures** representing its possible values.

5.7. SUBSTITUENT DEFINITIONS AND ECTR GENERATION

When a substituent group has been read, **procedure** POINTERLIST sets up a linked list, each item of which represents one RDECLARATIONTABLE entry for one substituent or substituent combination in the substituent group (plus one extra item for RDEFINITIONTABLE). This list is passed as a parameter (PARENTPSLIST) to **procedure** ALTNVLIST, which creates **alternative bars** in **child gates**, one **child gate** being built up on each of the items in PARENTPSLIST.

Procedure ALTNVLIST contains an iterative **repeat** loop which cycles round all the alternatives (separated by "/" delimiters) in a GENSAL substituent definition expression, and calls **procedure** ALTNTVE to analyse the definition elements separated by "&", "OSB" and "SB" delimiters. The PARENTPSLIST linked list is slightly reformatted before being passed as a parameter (PARALTLIST) to ALTNTVE, which passes it on to **procedure** ELEMENT, which analyses a single definition element and builds up the bulk of the ECTR.

5.7.1. Syntactic and Semantic Analysis in ELEMENT

Procedure GETLIMITPOSITIONS is used to determine the set of positions available in the **parent partial structures** for all the items in PARALTLIST, which also contains information on position sets given in previous recursions of definition element. Thus **procedure** POSITIONSET is able to give appropriate error messages if illegal positions are specified.

No such checking is performed in the analysis of selectors in definition elements. Thus no error would be detected in the following expression

R1 = phenyl sb [2] <5> methyl

There is no reason in principle why such checking should not be done, but it would involve considerable computational effort, and it has not been considered worthwhile as the the interpreter is not intended to be a teaching program. For similar reasons, no error is reported in the analysis of statements such as

R1 = phenyl sb [2] (F & Cl & Br & I)

5.7.2. Substituent Values

The analysis of substituent values is performed in a **case** statement, with separate **procedures** to handle each path.

For parenthesised substituent definition expressions, **function** NEWPARENTPSLIST sets up a new linked list, based on the items in PARALTLIST, for passing in a recursive call to **procedure** ALTNVLIST. This **function** also adds an extra non-BOTTOMBAR **combination bar** to the various **child gates** accessed via the items of PARALTLIST.

Since GENSAL treats substituents occurring as substituent values as parenthesised expressions (Section 3.7.3), an extra non-BOTTOMBAR **combination bar** is incorporated into the **child gates** with the DUMMY **partial structure** created to represent the substituent (Section 5.6) being included as one of the ALTERNATIVES leading from it. This is done by **function** EXTRALAYER.

Structure diagrams, always preceded by the delimeter "SD", are handled by calls to the Feldmann program and **procedure** PROCESSCT; appropriate **partial structure** records are also set up for "?" and "other term" substituent values.

5.7.3. Nomenclatural Terms

The analysis of nomenclatural terms is quite complicated, and is handled by **procedure** TRANSLATENOMEN. The approach used in this implementation of GENSAL has been to maintain a dictionary of nomenclatural terms (SPSDICT) which gives access to a file of structure records (SPSFILE). The entries in SPSFILE may be of three types: a connection table, a set of homologous series term parameters, or a GENSAL expression. In order to allow synonyms to be handled, several different records in SPSDICT may give access to the same record in SPSFILE.

Function RECORDHELD determines whether or not a record is held for a particular nomenclatural term; if none is, then the term is treated as an "other term" and an OTHER **partial structure** is used to store the character string itself.

If there is a record, **function** SPSVARIETY determines which of the three possible types it is. Both SPECIFIC (connection table) and GENERIC (parameter list) entries can be handled quite simply. OTHER (GENSAL expression) entries are more complicated.

This type of SPSFILE record is used for compound nomenclatural terms, which can be analysed into simpler terms: examples include "halophenyl", "diethylamino" and "N-methyl-2-propionamido". Other such terms represent a delimited series of alternatives, such as "halogen" or "alkali metal". The SPSFILE entry for halophenyl gives the expression "phenyl sb halogen", and that for halogen the expression "F / Cl / Br / I". The entries for phenyl, F, Cl, Br and I are all partial connection tables.

The processing of such an expression involves saving the current input BUFFER etc., and then calling ALTNVLIST recursively to analyse it; after the return from ALTNVLIST the original input BUFFER is restored. Such nesting of expressions can continue to any level, and the interpreter effectively treats each expression obtained from SPSFILE as if it were in parentheses (in fact the expression as it appears in SPSFILE is always terminated by a parenthesis).

The "highest-level" convention for further substitution on parenthesised expressions (Section 3.7.4) is of particular importance when dealing with compound nomenclatural terms and was chosen in preference to the alternative "lowest-level" convention on account of the problems that the latter would cause with such expressions.

If the expression "halophenyl sb methyl" were to occur in a GENSAL sentence, the dictionary-lookup operation would result in its being treated as

(phenyl sb (F / Cl / Br / I)) sb methyl

and the highest-level convention means that the methyl group is attached to the phenyl group and not to any of the halogens.

The process of dictionary lookup effectively changes the "right-rooted tree" of the compound nomenclatural term (where the rightmost part of the term is connected back to the parent structure) to the "left-rooted tree" of a GENSAL expression (where the leftmost part is connected back to the parent structure).

A minor problem remains with compound terms such as "alkoxy" which, if interpreted as

(oxy sb alkyl)

would imply in the expression

alkoxy sb chlorine

that the chlorine was substituted on the oxy group. This is not the generally-understood meaning of such expressions, and the problem is really a result of the conflict between the use of right-rooted and left-rooted trees in standard chemical nomenclature and GENSAL expressions (which derives them from the forms of statement in patents) respectively.

The interpreter program gets round the problem by a "fiddle" of dubious chemical validity and the SPSFILE entry for "alkoxy" is

alkyl sb [0/1] oxy

Thus the oxy group is regarded as being a **child** of the alkyl group rather than vice-versa, and the <u>position combination</u> [0/1] is used to indicate that the oxy group is interposed between the alkyl group and its **parent** structure.

This approach is justified firstly because it avoids a tricky problem, and secondly because it will allow the SPECIFIC oxy group **partial structure** to be handled along with the GENERIC

partial structure for the alkyl group in fragment generation.[67] Had the alkyl group been a **child** of the oxy, this would have been more difficult. In any case, so far as the GENSAL user typing "alkyl sb chlorine" is concerned, the whole arrangement is hidden, and he need not be aware of the construction of the ECTR.

It is possible that a compound nomenclatural term may be converted via SPSFILE to an expression involving a simple **homologous series term**; the terms "chloroalkyl" and "alkoxy" are examples of this. Any parameter list given after the term will thus be used to specify parameters for the simple **homologous series term** in the GENSAL expression. However, as this term may be nested in several layers of GENSAL expressions, the variable INSERTHSTPS is used to keep a note of any GENERIC **partial structure** in the expressions obtained from SPSFILE. Should more than one **homologous series term** be encountered during the processing of an expression from SPSFILE, it would not be clear which of them should be qualified by the parameter list; for this reason a program error is given in this situation, which should not arise if care is taken in the construction of SPSFILE.

Routines exist in the main part of GENPROG for adding records to SPSDICT and SPSFILE, though one of the problems in building up these files has been deciding how to interpret certain terms. This is a matter discussed by Dyson[54] and referred to in Section 1.4.6. For example, it is not always clear if the term "alkenyl" indicates exactly one double bond, or a minimum of one. Clearly, a decision of some sort has to be made for the purposes of

SPSFILE, but it is possible that an operational system might allow the user to redefine certain terms for his/her own use, and perhaps to maintain a private dictionary file.

Ultimately, the real problem is that the meaning of a term may differ from patent to patent, and may be left deliberately vague; sometimes patents define the meaning of a particular term used, but the definitions of a term like "aryl" differ widely from patent to patent. There appears to be no simple solution to this difficulty, which will only finally be overcome if the drafters of patents agree on standard meanings for the terms they use.

5.7.4. Parameter Lists

Procedure PARAMETERLIST carries out the analysis of parameter lists, and checks that the values given for each parameter are a subset of those implied in the **homologous series term** to which the list is being applied. Thus, for example, any value other than 0 for the number of rings in a parameter list applied to the term "alkyl" would be illegal.

Since it is possible for an **homologous series term** to be missing from SPSDICT, **procedure** TRANSLATENOMEN permits terms not found to be followed by parameter lists, though the information they give is not stored in the ECTR.

5.7.5. ECTR Generation

Two procedures handle the creation of child and parent gates, respectively SETCOMBARS (which calls function NEWCOMBAR) and SETPARENTGATE.

SETCOMBARS uses procedure GETCHILDPOSITIONS to determine the positions available in the child structure for the connection(s) to its parent. This procedure additionally checks that the bond orders specified are compatible. For each connection the bond order may have been specified in both the parent partial structure and the child partial structure (though in many cases either or both of these will be NOTSPECIFIED). Procedure BONDCHECK uses a table of bond orders, BONDMATCHARRAY, to determine a bond order compatible with the two specified: for example a CHAIN bond and a SINGLE bond result in a CHAISING bond, whereas a CHAISING and a CHAIDOUB bond are recognised as incompatible. Two NOTSPECIFIED bonds result in a SINGLE bond, so that no NOTSPECIFIED bonds are left when the ECTR is complete.

The child positions determined may be modified if there is a position set following the substituent value, this modification being achieved by procedure MODIFYCHILDPOSITIONS.

Where further substitution has been specified on substituents given as substituent values, as in

R1 = R2 sb methyl

Procedure ADDFURTHERSUBTN is used to copy the **partial structures** for this further substitution (methyl in the above example) onto the **partial structures** created when the substituent in question (R2 in the above example) is defined. This uses **function** PPOSNS to check that any position sets specified are actually available, before calling COPYCOMBAR to copy the **gates**.

5.8. MULTIPLIER DECLARATIONS AND DEFINITIONS

Multipliers appear only in **structure diagrams** and MDECLARATIONTABLE records information about the **partial structures** in which they occur, and also the substituents to which they apply.

As each multiplier is defined, the values for it are placed in MDEFINITIONTABLE, and only on completion of the processing of a GENSAL sentence does **procedure** RECORDMULTS actually transfer this information to the ECTR, in appropriate FREQUENCY fields in the top **bars** of **child gates**.

5.9. TIDYING THE ECTR

Before returning to the main part of GENPROG, the interpreter calls **procedure** TIDYINTREP, which DISPOSEs of certain redundant parts of the ECTR: these are mainly **partial structures** and their associated **gates** that were set up as parts of the entries in RDEFINITIONTABLE, and which were only required for checking purposes during **procedure** INTERPRET. A few other linked lists used for housekeeping purposes are also DISPOSEd by TIDYINTREP.

Procedure OUTINTREP is used to output a representation of the ECTR to a diagnostics file, if desired, but this is intended only for programmer checks on the working of INTERPRET.

Finally, control is passed back to GENPROG, where the ECTR can be used for fragment generation and other purposes.

CHAPTER 6

CONCLUSIONS AND SUGGESTIONS FOR FURTHER WORK

"Give us the tools, and we will finish the job"

Churchill

6.1. DEVELOPMENT OF A PATENT DOCUMENTATION SYSTEM

The input language GENSAL is essentially a means of describing generic chemical structures; it is not a means of encoding patent specifications as such. Many patents contain several generic structures covering, for example, the various components of a mixture, or different intermediates in a reaction pathway, and a patent documentation system might require all these to be encoded separately in GENSAL.

Considerable development work would be needed to take GENSAL, the ECTR, the interpreter program, and the work described by Welford [67] to form a comprehensive online patent information system, and further research work is also still required to make such a system viable.

From the interpreter point of view, the most immediate task is obviously to extend the program as it exists to implement the full GENSAL language, including conditions. But more important than this is likely to be the developement of fragment-based, and possibly atom-by-atom search algorithms, and approaches to this are discussed by Welford. [67]

In order to evaluate such algorithms, a database of at least some hundreds of generic structures from patents, and some sample queries, will be required. A number of companies in the chemical and pharmaceutical industries have expressed interest in participating in the encoding of structures in GENSAL for this purpose.

The building up of a database will also enable a full evaluation to be made of the power of GENSAL to encode generic structures in patents. Additionally, it will permit an analysis of the effort required to encode a generic structure from a patent in GENSAL; Pötscher [44] has pointed out that in the encoding of generic structures in the GREMAS system, the difficult part of the operation is the analysis of the structure as described, rather than the selection of GREMAS terms, and this analysis would also

- to a large extent at least - need to be carried out for GENSAL coding.

Certainly, GENSAL coding from patents or abstracts is not a clerical task, though experience and a basic knowledge of elementary chemistry would be adequate qualifications for a coder. GENSAL coding is likely to require much less training than that required for encoding in a fragment-based system.

The possibility of automatic generation of GENSAL from patent specifications or abstracts is an interesting one; Nishida and Takamatsu [193] have recently described a method for extracting information from patent claim text, though their work was not related to chemical patents. The problem would be likely to be a very difficult one in the application of artificial intelligence techniques, and any system developed would certainly require human interference at points where the specification is ambiguous. Such automatic input of generic structures might however be essential if a viable back file of patents were to be built up.

Associated with the input of a large number of structures in GENSAL will be the need to add terms to the dictionary of nomenclatural terms, and this will require many decisions to be made as to the meanings to be assigned to vague terms, as discussed in Section 5.7.3.

6.2. OTHER POTENTIAL APPLICATIONS OF GENSAL

Whilst GENSAL has been designed for the encoding of the generic structures in patents, and thus to form part of an integrated patent information system, it has a number of potential applications outside the field of computer-based patent documentation systems, and these will now be mentioned briefly.

6.2.1. Non-Computer Description of Generic Structures

Since GENSAL is designed to be a complete and unambiguous means of describing generic chemical structures, it could well have a use in non-computer contexts, just as high-level programming languages such as Algol are often used for the non-computer description of algorithms.

GENSAL is intended to be readily comprehensible to a chemist or patent agent who has had a fairly minimal training in its use (though rather more training would be required to achieve efficiency and accuracy in encoding structures), and it might therefore have applications in printed abstracts of patents, or in current awareness bulletins. If such a printed publication were produced by a computer-typesetting process, then the use of GENSAL would give the added advantage of leaving a complete and unambiguous description of the generic structure in machine-readable form, so that it could, perhaps, be incorporated into a computerised storage and retrieval system at a later date.

The clarity and lack of ambiguity of GENSAL would make such descriptions much easier to understand than those currently found in patent specifications and abstracts.

It is even possible to speculate that GENSAL might ultimately be used for generic structure descriptions in the patent documents themselves, though this is likely to remain speculation for some time to come.

6.2.2. Generic Structures in the Journal Literature

Figure 1.1 illustrated a generic structure from the journal literature, and such series of related compounds could conveniently be described using a single GENSAL structure, which might, if desired, be used for automatic generation of all the specific compounds covered, so that these could be registered in an appropriate specific-compound registry system. Integration with a quantitative structure activity relationship system might also allow the automatic identification of the compounds likely to be most active.

6.2.3. Chemical Reaction Documentation

One of the problems in the documentation of chemical reactions is the description of the "generalised" reaction process. Normally this is done in terms of substructures for the reactant and the product, which represent the "reaction centre" - i.e. the atoms and bonds actually involved in the reaction.

However, frequently the reaction is strongly influenced by the presence or absence of surrounding groups which do not actually participate in the bond changes. The description of the reaction centre as a generic structure, using GENSAL, would allow these variable surrounding groups to be taken into account, though the feasibility and development of a reaction indexing system based on this principle would need substantial research investigation.

6.2.4. Specific Structure Search Queries

Many of the chemical structure search systems currently available commercially have some features for the use of generic structure queries in searches of files of specific structures. For the most part these allow only a very restricted type of generic structure, usually the specification of a few alternative atoms or groups at particular defined points in the query structure, though the COUSIN system at Upjohn,[64] described in Section 1.4.7, allows a greater degree of sophistication with its "R_k" notation. Système DARC (Section 1.4.6) is also believed to be about to introduce substantial facilities for generic structure queries.

The use of GENSAL would permit much more complex generic structures to be input as queries for searches of a file of specific structures, potentially without any need for modification of the search software. A GENSAL interpreter program would convert the GENSAL input to the ECTR internal representation, and from this a special fragment-generation module would produce a set of search fragments compatible with those normally used for searching the file, with appropriate "AND" and "OR" logic.

6.3. CONCLUSIONS

The work described in this Thesis forms a viable basis for an improved storage and retrieval system for generic structures in patents, and it is the hope of the author that it may be used in the development of such a system.

It is possible that, as discussed in this Chapter, the work may have applications in other areas also. Improved patent documentation systems may additionally have an effect on the processes of drafting and granting patents. In 1966 Frome [194] discussed the legal problems that could be caused by computer programs able to print out all the specific compounds covered by a generic structure. Blick [195] has also pointed out a similar problem with computer-aided synthesis packages, which could affect the patentability of synthesis routes suggested by such packages.

Whatever the fate of the present work, it is certain that storage and retrieval systems for generic chemical structures will have increasing importance in many areas for many years to come.

APPENDIX 1

GENSAL SYNTAX DIAGRAMS

In these syntax diagrams, delimiter words and symbols are shown enclosed in boxes, data items are shown in upper case letters, and references to other syntax diagrams are shown in lower-case letters. A detailed discussion of the formal grammar of GENSAL is given in Section 3.13 of the text of this Thesis.

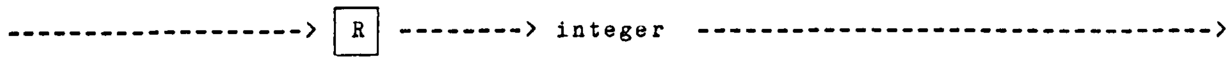
1. integer

```
-----------------------------------------------> digit ---------T--------------------------->
                                          ^                     |
                                          '---------------------'
```

2. integer range

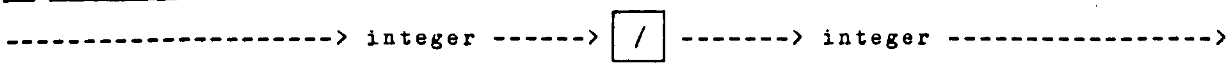
```
----+----> integer ---T------------------------------------------+------------->
    ^                 |                                          ^
    |                 |---> [-] ----T--------------------------->|
    |                 |             |                            |
    |                 |             '--------> integer --------->|
    |                 V                                          |
    '------ [,] <-----+------------------------------------------'
```

3. substituent

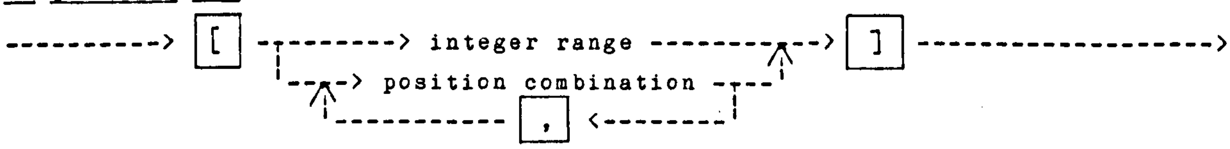


4. multiplier

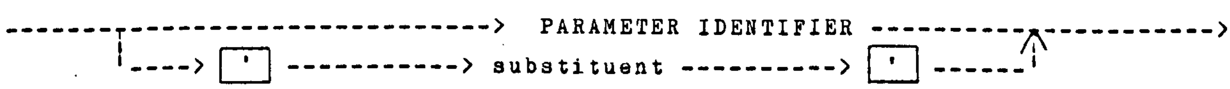
5. position combination



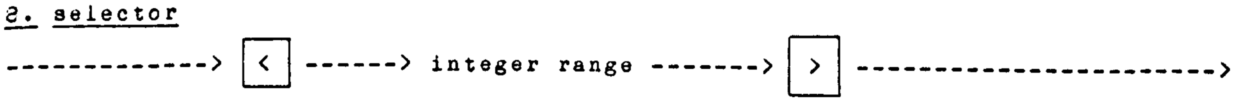
6. position set



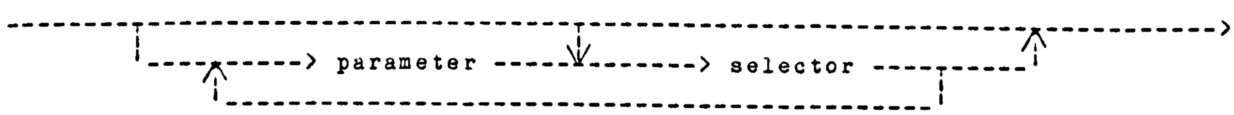
7. parameter



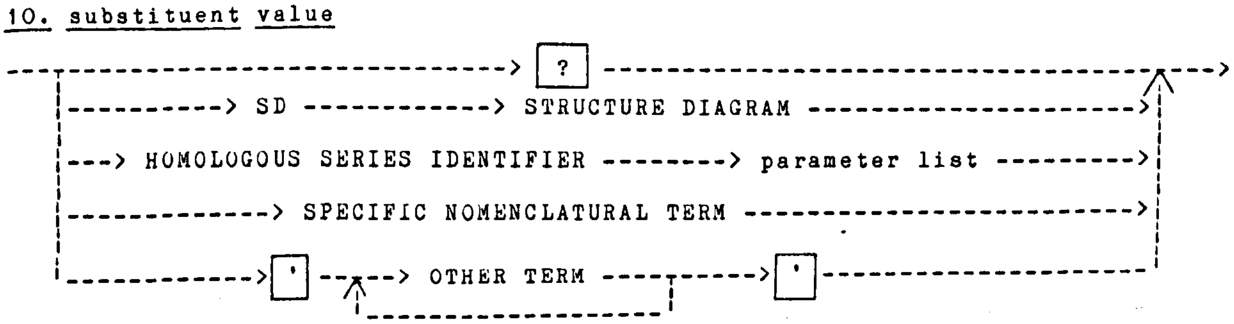
8. selector



9. parameter list



10. substituent value



11. substituent combination

```
----------> substituent ----> [+] -----> substituent ---------------------->
```

12. substituent group

```
-------------------T---------> [R] ----> integer range ------A--------------------->
                   |                                         ^
                   '--A----> substituent combination ---T--'
                      ^                                  |
                      '-------------- [,] <-------------'
```

13. multiplier group

```
----------------------> [M] ---> integer range ------------------------------->
```

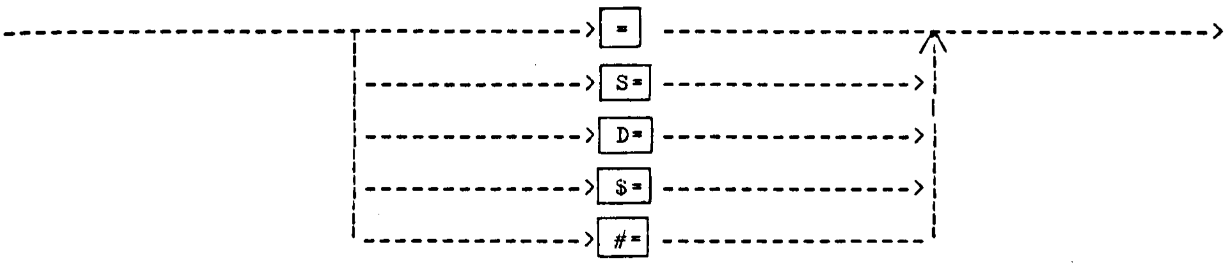
14. definition element

```
-----T----> position set ----------------------------------------.
     |                                                            |
     |<-----------------------------------------------------------'
     |
     |----> selector ---------------------------------------------.
     |                                                            |
     |<-----------------------------------------------------------'
     |
     |----> substituent value --T---------------------------------.
     |                          |                                 |
     |                          '---> position set -------------->|
     |                                                            |
     |----> substituent ----------------------------------------->|
     |                                                            |
     '----> [(] ---> substituent definition expression ---> [)] ------V------>
```

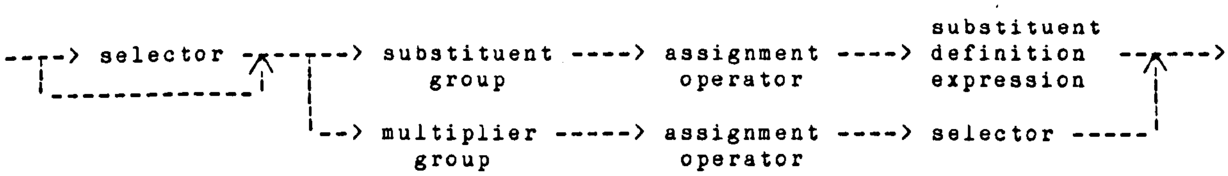
15. substituent definition expression

```
--------------A----------> definition element ----------------T-------------->
              ^                                               |
              |<--------------- [&] <-------------------------|
              |<-------------- [SB] <-------------------------|
              |<------------- [OSB] <-------------------------|
              '---------------- [/] <-------------------------'
```

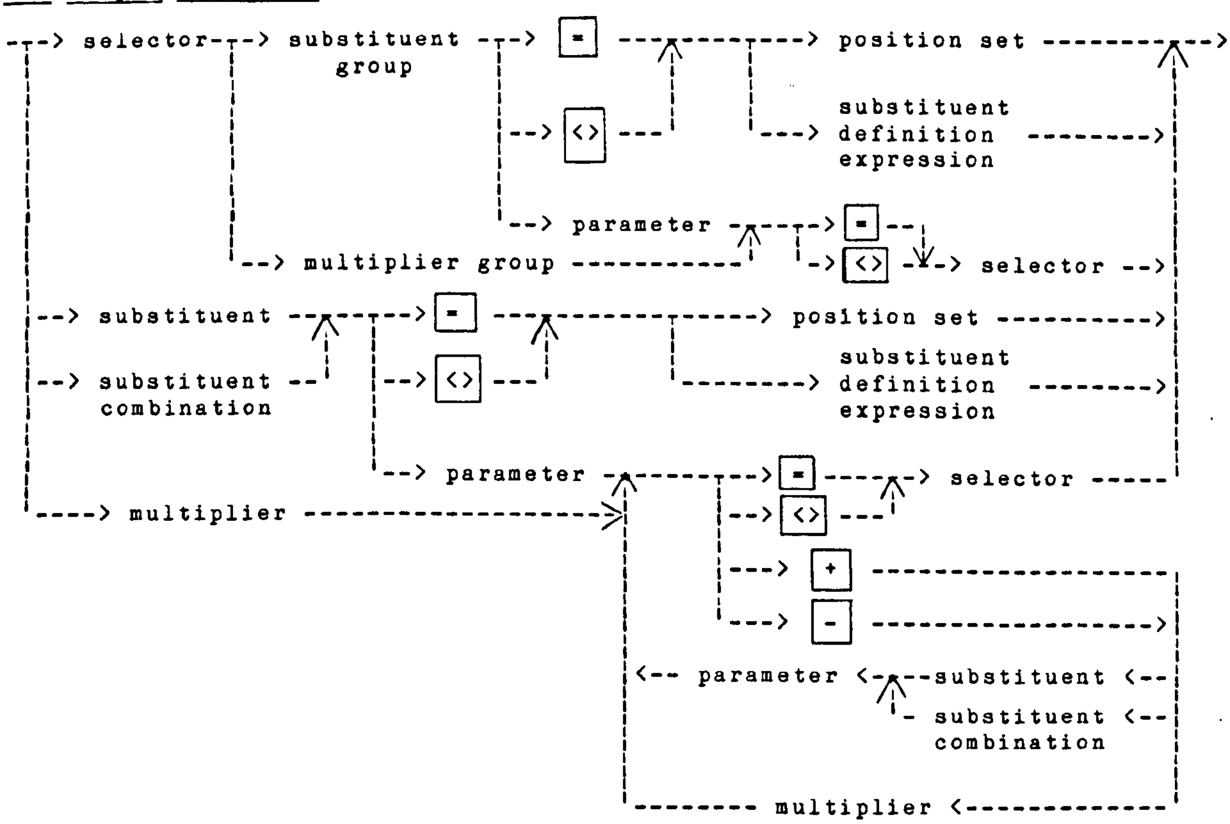
16. assignment operator



17. assignment statement



18. simple condition



19. condition

```
-----^---T-----------^-----T----> simple condition-----------^-------T------->
     |   |             |   |                                 |       |
     |   '--> [NOT] ---'   '----> [(] ---> condition ---> [)] ---'   |
     |                                                               |
     |<-------------------- [AND] <----------------------------------|
     |                                                               |
     '--------------------- [OR] <-----------------------------------'
```

20. statement

```
---T--------------------> assigment statement -----------------------^----->
   |                                                                 |
   |---> [IF] --> condition --> [THEN] --> statement ----T---------->|
   |                                                     |           |
   |                                   .-----------------'           |
   |                                   '--> [ELSE] --> statement --->|
   |                                                                 |
   |---> [RESTRICT] -----> condition ------------------------------->|
   |                                                                 |
   |---> [BEGIN] ----^---> statement ----T-----> [END] ------------->|
   |                 |                   |                           |
   |                 '------- [;] <------'                           |
   |                                                                 |
   '-----------------------------------------------------------------'
```

21. structure description

```
--T--> [INPUT] --^-> REF. NO. --> [SD] --> STRUCTURE --^--> statement --T--> [.] -->
  |              |                         DIAGRAM     |                |
  '--> [QUERY] --'                                     '------ [;] <----'
```

APPENDIX 2

BNF PRODUCTION RULES FOR GENSAL

This Appendix shows the Grammar of GENSAL using Backus-Naur Form (BNF) production rules. A slight variant of the "Extended BNF" metalanguage proposed by Wirth [101] is used, in which the syntactic constructs (non-terminal symbols) are shown enclosed in angle brackets, the symbol "::=" means "is replaced by", the symbol "|" means "or", curly brackets enclose symbols to be repeated zero or more times and square brackets enclose optional symbols. Terminal symbols included exactly as they stand are shown in **bold type** and are enclosed in double quote marks.

1. <arithmetic operator> ::= "**+**" | "**-**"

2. <assignment operator> ::= "**=**" | "**S=**" | "**D=**" | "**$=**" | "**#=**"

3. <assignment statement> ::= [<selector>]
        <unselected assignment statement>

4. <character> ::= <letter> | <digit> | <special character>

5. <compound statement> ::= "**BEGIN**" <statement>
        { "**;**" <statement> } "**END**"

6. <condition factor> ::= <simple condition> |
        "**(**" <condition> "**)**" | "**NOT**" <condition factor>

```
7. <condition term> ::= <condition factor> { "AND"
        <condition factor> }

8. <condition> ::= <condition term> { "OR" <condition term> }

9. <definition alternative> ::= <element combination> { <further
        substitution operator> <element combination> }

10. <definition element> ::= [ <position set> ] [ <selector> ]
        <unmodified definition element>

11. <definition relation> ::= <substituent variable>
        <relational operator> <position set> |
        <substituent variable> <relational operator>
        <substituent definition expression>

12. <digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
        "8" | "9"

13. <element combination> ::= <definition element> { "&"
        <definition element> }

14. <empty> ::=

15. <further substitution operator> ::= "SB" | "OSB"

16. <group definition relation> ::= <substituent group>
        <relational operator> <position set> |
        <substituent group> <relational operator>
        <substituent definition>

17. <group parameter relation> ::= <substituent group>
        <parameter> <relational operator> <selector>

18. <group relation> ::= <substituent group relation> |
        <multiplier group relation>

19. <homologous series identifier> ::= <nomenclature>

20. <IF statement> ::= "IF" <condition> "THEN" <statement>
        [ "ELSE" <statement> ]

21. <initial character> ::= <letter> | <digit>

22. <integer range> ::= { <subrange> "," } <top range>

23. <integer term> ::= <multiplier> | <substituent variable>
        <parameter>

24. <integer> ::= <digit> { <digit> }

25. <letter> ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" |
        "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" |
        "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
```

26. <multiplier assignment> ::= <multiplier group>
<assignment operator> <selector>

27. <multiplier group relation> ::= <multiplier group>
<relational operator> <selector>

28. <multiplier group> ::= "**M**" <integer range>

29. <multiplier relation> ::= <multiplier>
{ <arithmetic operator> <integer term> }
<relational operator> <selector>

30. <multiplier> ::= "**M**" <integer>

31. <nomenclature> ::= <initial character> { <character> }

32. <other term> ::= <nomenclature>

33. <parameter identifier> ::= "**C**" | "**T**" | "**Q**" | "**E**" | "**Y**" | "**RC**"
| "**RN**" | "**RS**" | "**RA**" | "**Z**"

34. <parameter list> ::= [<selector>]
{ <parameter> <selector> }

35. <parameter relation> ::= <substituent variable> <parameter>
{ <arithmetic operator> <integer term> }
<relational operator> <selector>

36. <parameter> ::= <parameter identifier> |
"*" <substituent> "*"

37. <position combination> ::= <integer> "/" <integer>

38. <position set> ::= "[" <positions> "]"

39. <positions> ::= <integer range> | <position combination>
{ "," <position combination> }

40. <reference number> ::= <integer>

41. <relational operator> ::= "=" | "<>"

42. <restrict statement> ::= "**RESTRICT**" <condition>

43. <selector> ::= "<" <integer range> ">"

44. <simple condition> ::= <selector> <group relation> |
<substituent relation> | <multiplier relation>

45. <special character> (Implementation Dependent)

46. <specific nomenclatural term> ::= <nomenclature>

47. <statement> ::= <assignment statement> | <if statement> |
<restrict statement> | <compound statement> | <empty>

48. <structure description> ::= <structure type>
<reference number> "**SD**" <structure diagram> <statement>
{ ";" <statement> } "."

49. <structure diagram> (Implementation Dependent)

50. <structure type> ::= "**INPUT**" | "**QUERY**"

51. <subrange> ::= <integer> | <integer> "-" <integer>

52. <substituent assignment> ::= <substituent group>
<assignment operator>
<substituent definition expression>

53. <substituent combination> ::= <substituent> "+" <substituent>

54. <substituent definition expression> ::=
<definition alternative>
{ "/" <definition alternative> }

55. <substituent group relation> ::= <group definition relation>
<group parameter relation>

56. <substituent group> ::= "**R**" <integer range> |
<substituent combination>
{ "," <substituent combination> }

57. <substituent relation> ::= <definition relation> | <parameter relation>

58. <substituent value> ::= "?" | "**SD**" <structure diagram> |
<homologous series identifier> <parameter list > |
<specific nomenclatural term> | "'" <other term>
{ <other term> } "'"

59. <substituent variable> ::= <substituent> |
<substituent combination>

60. <substituent> ::= "**R**" <integer>

61. <top range> ::= <subrange> | <integer> "-"

62. <unmodified definition element> ::= <substituent value> |
<substituent value> <position set> | <substituent> |
"(" <substituent definition expression> ")"

63. <unselected assignment statement> ::=
<substituent assignment> | <multiplier assignment>

APPENDIX 3

GENSAL INTERPRETER PROGRAM

```
PROCEDURE PROGERROR(ERRORCODE : INTEGER);
EXTERN;


PROCEDURE GOTOCOMMAND;
EXTERN;
{ Sends the user to GENESIS command level via a GOTO in the main program}



PROCEDURE DESTROY(VAR PTR1 : PDOUBLIST);
EXTERN;
{ This destroys the elements of a linked list of type PDOUBLIST, starting
  at the element pointed to be the parameter PTR1, returned as NIL.
  Called by INTERPRET\INTSET
            INTERPRET\GROUPRANGE
            INTERPRET\SELECTOR
            INTERPRET\POSITIONSET
            INTERPRET\MODIFYPOSITIONS\TRACEDOWNGATE
            INTERPRET\ALTNVLIST\ELEMENT\SETCOMBARS\CHECKCOMBPOSNS
            INTERPRET\ALTNVLIST\ELEMENT\PARAMETERLIST
            INTERPRET\ASSIGNMENTSTMNT\MULTASSIGNMENT}




PROCEDURE ADDINTS (VAR PTR1     : PDOUBLIST;
                   LOWER, UPPER : INTEGER);
EXTERN;
{ Adds LOWER and UPPER to the values already in PTR1 (if they are contiguous),
  or places them in a new DOUBLIST element, returned as PTR1, with the original
  PTR1 as its NEXT field.
```

```
        Called by INTERPRET\INTEGERRANGE\RANGEFRAGMENT
                  INTERPRET\SETINTS}



      PROCEDURE PRINTNOM(NOMENVAL : STRING32);
      EXTERN;
      { Prints a nomenclatural term up to the last non-space character
        Called by INTERPRET\ALTNVLIST\RECORDHELD}



      PROCEDURE DELETEGENSAL(VAR LINE1 : PLINELIST);
      EXTERN;
      { Deletes a linked list of GENSAL lines, headed by LINE1, which is returned
        with value NIL.
        Called by INTERPRET\ALTNVLIST\ELEMENT\TRANSLATENOMEN}



      PROCEDURE DECODECT (VAR CTLINE : PLINELIST;
                          DISPLAYING : BOOLEAN);
      EXTERN;
      { Decodes a connection table from character-string format, beginning in CTLINE^.
        LINE, making entries in FELDCT and FELDBD. CTLINE is left pointing at the last
        line of the connection table string. FELDMN is used to display the structure
        diagram if DISPLAYING.
        Called by INTERPRET\READSD}



      PROCEDURE ENCODECT(VAR CTLINE : PLINELIST);
      EXTERN;
      { Encodes the contents of FELDCT and FELDBD as a character string, and
        places it in successive lines, starting with CTLINE, which is
        returned pointing to the last line of encoded connection table.
        Called by INTERPRET\READSD}
```

```
FUNCTION NORECORD(NOMEN       : STRING32;
                  VAR ADDRESS : INTEGER): BOOLEAN;
EXTERN;
{ Checks whether or not there is a record held in SPSDICT for NOMEN,
  The ADDRESS from SPSDICT for the term is returned  as a side effect.
  Called by INTERPRET\ALTNVLIST\RECORDHELD}



FUNCTION TERMREAD(VAR TERM : STRING32) : BOOLEAN;
EXTERN;
{ Reads a single TERM from the terminal, upper-cases it, and returns FALSE
  if it has no characters.
  Called by INTERPRET\ALTNVLIST\RECORDHELD}




PROCEDURE LISTPARAMS(VAR OUTFILE : TEXT;
                         PARAMLIST   : TPARAMLIST);
EXTERN;
{ Lists the parameters in PARAMLIST in file OUTFILE, which must
  already have been RESET.
  Called by INTERPRET\OUTINTREP\WRITEPS}



FUNCTION SPSVARIETY(ADDRESS    : INTEGER;
                    DISPLAYING : BOOLEAN) : TPSVARIETY;
EXTERN;
{ Returns the variety of partial structure, whose record begins at ADDRESS in
  SPSFILE,optionally DISPLAYING the structure. SPECIFIC PSs are entered in
  FELDCT/FELDBD, GENERIC PSs in SPSPARAMLIST and OTHER PSs in INSERTGENEX.
  Called by INTERPRET\ALTNVLIST\ELEMENT\TRANSLATENOMEN}



PROCEDURE READFELDMANN;
EXTERN;
{ Reads the Feldmann table from FELDFIL.
```

```
  Called by INTERPRET\PROCESSCT}



PROCEDURE INTERPRET(VAR FIRSTLINE : PLINELIST;
                    VAR ECTRSIZE  : INTEGER;
                    INTERACTIVE   : BOOLEAN);

{ This is the GENSAL interpreter routine, and performs syntactic and semantic
  analysis on a GENSAL sentence, creating the ECTR. }

CONST NOTFIXED    = 0;

TYPE DELIMTYPE    =(INVALIDTOKEN,GAMPERSAND,GPRIME,GLPAREN,GRPAREN,GPLUS,GCOMMA,
                    GHYPHEN,GPERIOD,GSLASH,GSEMI,GOPENANG,GNOTEQ,GEQUALS,
                    GCLOSANG,GQUEST,GLSQUARE,GRSQUARE,GAND,GBEGIN,GC,GE,GELSE,
                    GEND,GIF,GINPUT,GM,GN,GOR,GORBY,GOSB,GP,GQ,GQUERY,GR,GRA,
                    GRC,GRESTRICT,GRF,GRN,GRS,GSB,GSD,GT,GTHEN,GY,GZ,GDEQ,GSEQ,
                    GHASHEQ,GDOLEQ);
     TOKENNATURE  =(DELIMITER,INTEGRAL,NOMENCLATURE);
     TINPUTMODE   =(TERMINAL, STOREDGENSAL, INSERTTEXT);
     TBONDMAG     = 0..3;
     DELIMSET     = SET OF DELIMTYPE;
     TOKENTYPE    = RECORD
                     CASE NATURE : TOKENNATURE OF
                      DELIMITER    : (DELIMVAL : DELIMTYPE);
                      INTEGRAL     : (INTEGVAL : INTEGER);
                      NOMENCLATURE : (NOMENVAL : STRING32)
                    END;
     PTOKENLIST   = ^TTOKENLIST;
     TTOKENLIST   = RECORD
                     TOKENVAL : TOKENTYPE;
                     NEXT     : PTOKENLIST
                    END;
     PPSLIST      = ^PSLIST;
     PSLIST       = RECORD
                     PARSTRUCT,
                     FURTHERSUB  : PTRPSTYPE;
                     COMBINS     : PCOMBINLIST;
                     CONNBONDS   : TCONNBONDS;
```

```
                        PRNTPOSNS   : PTGROUPMEMS;
                        COPYCHILDPS : BOOLEAN;
                        NEXT        : PPSLIST
                       END;
     PMDECLIST    = ^MDECLIST;
     MDECLIST     = RECORD
                        SUBSTDECN : PPSLIST;
                        NEXT      : PMDECLIST
                       END;

VAR TOKEN                : TOKENTYPE;
    TOKENLIST            : PTOKENLIST;
    LINENUMBER           : INTEGER;
    VALIDLENGTH          : 0..MAXLENGTH;
    DEFNMULT,
    DEFNSUBS,                                   { substituents so far defined }
    DECLMULT,
    DECLSUBS             : INTEGSET;
    CONDITIONSPRESENT : BOOLEAN;
    RDECLARATIONTABLE : ARRAY[SUBSTITUENT] OF PPSLIST;
    RDEFINITIONTABLE  : ARRAY[SUBSTITUENT] OF PCOMBINLIST;
    MDECLARATIONTABLE : ARRAY[MULTIPLIER] OF PMDECLIST;
    MDEFINITIONTABLE  : ARRAY[MULTIPLIER] OF INTRECORD;
    CURRENTLINE          : PLINELIST;
    SUBST                : SUBSTITUENT;
    ZEROFREQ,
    ESSENTFREQ,
    OPTFREQ              : PDOUBLIST;
    BONDMATCHARRAY       : ARRAY[BONDORDER] OF PACKED ARRAY[BONDORDER] OF BONDORDER;
    BONDSTRING           : ARRAY[BONDORDER] OF STRING2;
    INPUTMODE            : TINPUTMODE;
    INSERTHSTPS          : PTRPSTYPE;
    IRLISTBOT            : PIRLIST;



{-----------------------------------------------------------------------}
PROCEDURE INITIALISE;

{ Sets initial values for variables.
```

```
  Called by Body of INTERPRET}

VAR SUBST    : SUBSTITUENT;
    BOND     : BONDORDER;
    MULT     : MULTIPLIER;
    BONDFILE : FILE OF PACKED ARRAY[BONDORDER] OF BONDORDER;



FUNCTION NEWFREQ(ONE, TWO : INTEGER) : PDOUBLIST;

VAR NF : PDOUBLIST;

BEGIN
NEW(NF);
NF^.FIRST  := ONE;
NF^.SECOND := TWO;
NF^.NEXT   := NIL;
NEWFREQ := NF
END;



BEGIN {Body of INITIALISE}
ESSENTFREQ := NEWFREQ(1,1);
OPTFREQ := NEWFREQ(0,1);
ZEROFREQ := NEWFREQ(0,0);
ECTRSIZE := 18;
CONDITIONSPRESENT := FALSE;
WRITELN;
WRITELN;
LINENUMBER := 0;
DECLSUBS  := [];
DECLMULT := [];
DEFNMULT := [];
DEFNSUBS := [];
IF INTERACTIVE THEN INPUTMODE := TERMINAL
               ELSE INPUTMODE := STOREDGENSAL;
INTERNALREP.CONSTANTPART := NIL;
TOKENLIST := NIL;
```

```
N := MAXLENGTH;
FOR SUBST := 1 TO MAXVARS DO
   BEGIN
    RDECLARATIONTABLE[SUBST] := NIL;
    RDEFINITIONTABLE[SUBST] := NIL
   END;
FOR MULT := 1 TO MAXVARS DO
   BEGIN
    MDECLARATIONTABLE[MULT] := NIL;
    MDEFINITIONTABLE[MULT].TOPRANGE := NOTSET;
    MDEFINITIONTABLE[MULT].SUBRANGES := NIL
   END;
CURRENTLINE := NIL;

RESET(BONDFILE, 'LI2GEN>BONDFILE');
FOR BOND := NOTSPECIFIED TO RINGTAUT DO READ(BONDFILE, BONDMATCHARRAY[BOND]);
RESET(BONDFILE, '@TTY');

BONDSTRING[NOTSPECIFIED]:= 'NS';
BONDSTRING[ANY]         := 'A ';
BONDSTRING[CHAIN]       := 'C ';
BONDSTRING[RING]        := 'R ';
BONDSTRING[SINGLE]      := 'S ';
BONDSTRING[DOUBLE]      := 'D ';
BONDSTRING[TRIPLE]      := 'T ';
BONDSTRING[CHAISING]    := 'CS';
BONDSTRING[CHAIDOUB]    := 'CD';
BONDSTRING[CHAITRIP]    := 'CT';
BONDSTRING[CHAITAUT]    := 'TC';
BONDSTRING[RINGSING]    := 'RS';
BONDSTRING[RINGDOUB]    := 'RD';
BONDSTRING[RINGTRIP]    := 'RT';
BONDSTRING[AROMATIC]    := 'RA';
BONDSTRING[RINGTAUT]    := 'TR'

END;                            { of INITIALISE
--------------------------------------------------------------------}
```

```
PROCEDURE WRITEMESSAGE(ERRORCODE,
                       NUMDATA    : INTEGER;
                       STRINGDATA : STRING4);

{ Obtains an error message from LI2GEN>ERRORMSGS, and prints it at the
  terminal, interposing data where necessary.
  Called by FAILURE
            ERROR
            PROCESSCT\REJECT}

VAR STRINGPOS : 1..5;
    MSGCHAR   : CHAR;
    LINE      : INTEGER;

BEGIN
RESET(INPUT, 'LI2GEN>ERRORMSGS');
FOR LINE := 1 TO (ERRORCODE-1) DO READLN;
STRINGPOS := 1;
WHILE NOT EOLN(INPUT) DO
   BEGIN
    READ(MSGCHAR);
    CASE MSGCHAR OF
     '#'     : WRITE(NUMDATA : 1);
     '$'     : BEGIN
                WRITE(STRINGDATA[STRINGPOS]);
                STRINGPOS := STRINGPOS + 1
               END;
     OTHERWISE WRITE(MSGCHAR)
     END
   END;
WRITELN;
RESET(INPUT, '@TTY')
END;



PROCEDURE FAILURE(ERRORCODE,
                  NUMDATA    : INTEGER;
                  STRINGDATA : STRING4);
```

```
{ Called when an irrecoverable error is encountered, and processing cannot
  continue. A message is printed, and the use retruned to GENESIS command
  mode. }

BEGIN
WRITELN;
WRITELN('**** FAILURE ', ERRORCODE : 2);
WRITEMESSAGE(ERRORCODE, NUMDATA, STRINGDATA);
WRITELN;
WRITELN('Edit existing GENSAL or start again!');
WRITELN;
GOTOCOMMAND
END;



PROCEDURE REDUCEECTR(PTR : PDOUBLIST);

BEGIN
WHILE PTR <> NIL DO
   BEGIN
    ECTRSIZE := ECTRSIZE - 6;
    PTR := PTR^.NEXT
   END
END;



{-----------------------------------------------------------------------

                P R O C E D U R E    G E T T O K E N

               T H E   L E X I C A L   A N A L Y Z E R
               ----------------------------------------}


PROCEDURE GETTOKEN;

{  Places the next token in the GENSAL input stream in TOKEN.
   Called by NEXTTOKEN
```

```
                 LOOKAHEAD
                 ERROR}

    VAR M         : 1..MAXLENGTH;
        STARTED   : BOOLEAN;



    PROCEDURE READLINE;

    { Reads one line of GENSAL input according to INPUTMODE, checking for TERMINAL
      that it contains no more than 99 characters, and building up the linked list
      of lines. For all INPUTMODE replaces all lower-case alphabetics by upper-case. }

    LABEL 10;

    VAR CH          : CHAR;
        M           : 0..MAXLENGTH;
        NEWLINE     : PLINELIST;

    BEGIN
    CASE INPUTMODE OF
     TERMINAL     : BEGIN
                     LINENUMBER := LINENUMBER + 1;
                     10 : WRITE(LINENUMBER : 3);
                     IF DIAGNOSTICS THEN WRITE(ECTRSIZE : 7 );
                     WRITE('  GENSAL: ');
                     READLN(BUFFER : N);
                     IF N=0 THEN
                        BEGIN
                         WRITELN;
                         WRITELN('GENSAL input terminated by user.');
                         WRITELN;
                         GOTOCOMMAND
                        END;
                     IF N=100 THEN
                        BEGIN
                         WRITELN;
                         WRITELN('**** LINE OVERFLOW! ****');
                         WRITE('Line read as far as "...');
```

```
                FOR M := (MAXLENGTH-12) TO (MAXLENGTH-1) DO WRITE(BUFFER[M]);
                WRITELN('".');
                REPEAT
                 WRITE('                                                  OK?  (Y/N) > ');
                 READLN(CH)
                UNTIL (CH='Y') OR (CH='y') OR (CH='N') OR (CH='n');
                M := MAXLENGTH;
                REPEAT
                 BUFFER[M] := ' ';
                 M := M-1
                UNTIL (BUFFER[M]=' ') OR (M=0) OR (CH='Y') OR (CH='y');
                IF (CH='N') OR (CH='n') THEN WRITELN('Line truncated from last space.')
               END;
            NEW(NEWLINE);
            WITH NEWLINE^ DO
               BEGIN
                LAST := CURRENTLINE;
                NEXT := NIL;
                LINE := BUFFER
               END;
            IF CURRENTLINE = NIL
               THEN FIRSTLINE := NEWLINE
               ELSE CURRENTLINE^.NEXT := NEWLINE;
            CURRENTLINE := NEWLINE
           END;

STOREDGENSAL : BEGIN
                LINENUMBER := LINENUMBER + 1;
                IF LINENUMBER = 1
                   THEN NEWLINE := FIRSTLINE
                   ELSE NEWLINE := CURRENTLINE^.NEXT;
                IF NEWLINE = NIL
                   THEN BEGIN
                         WRITELN;
                         WRITELN('End of stored GENSAL.');
                         WRITELN('Input at the terminal:');
                         WRITELN;
                         INPUTMODE := TERMINAL;
                         GOTO 10
                        END
```

```
                    ELSE CURRENTLINE := NEWLINE;
                 WRITE(LINENUMBER : 3);
                 IF DIAGNOSTICS THEN WRITE(ECTRSIZE : 7);
                 WRITE('  GENSAL: ');
                 BUFFER := CURRENTLINE^.LINE;
                 WHILE (BUFFER[N]=' ') AND (N>1) DO N := N-1;
                 IF (N=1) AND (BUFFER[1]=' ') THEN N:= 0;
                 FOR M := 1 TO N DO WRITE(BUFFER[M]);
                 WRITELN
                END;

 INSERTTEXT   : BEGIN
                 IF CURRENTLINE = NIL THEN
                PROGERROR(1); {Unterminated GENSAL expression in SPSfile}
                 BUFFER := CURRENTLINE^.LINE;
                 CURRENTLINE := CURRENTLINE^.NEXT
                END
 END;

FOR M:= 1 TO MAXLENGTH DO
   IF (BUFFER[M] >= 'a') AND (BUFFER[M] <= 'z')
      THEN BUFFER[M] := CHR( ORD(BUFFER[M]) - ORD('a') + ORD('A') );
N := 1
END (* Of READLINE *);



FUNCTION CHECK (TESTDELIM : DELIMTYPE) : BOOLEAN;

{ Returns TRUE if the delimiter passed as TESTDELIM is found, correctly
  terminated in BUFFER }

VAR RESULT      : (NOTFOUND, PENDING, FOUND);
    M           : 0..MAXLENGTH;
    TERMCHARS   : SET OF CHAR;
    DELIMSTRING : ALFA;

BEGIN
CASE TESTDELIM OF
 GAND:      DELIMSTRING := 'AND      ';
```

```
   GBEGIN:     DELIMSTRING := 'BEGIN   ';
   GC:         DELIMSTRING := 'C       ';
   GE:         DELIMSTRING := 'E       ';
   GELSE:      DELIMSTRING := 'ELSE    ';
   GEND:       DELIMSTRING := 'END     ';
   GIF:        DELIMSTRING := 'IF      ';
   GINPUT:     DELIMSTRING := 'INPUT   ';
   GM:         DELIMSTRING := 'M       ';
   GN:         DELIMSTRING := 'N       ';
   GOR:        DELIMSTRING := 'OR      ';
   GORBY:      DELIMSTRING := 'ORBY    ';
   GOSB:       DELIMSTRING := 'OSB     ';
   GP:         DELIMSTRING := 'P       ';
   GQ:         DELIMSTRING := 'Q       ';
   GQUERY:     DELIMSTRING := 'QUERY   ';
   GR:         DELIMSTRING := 'R       ';
   GRA:        DELIMSTRING := 'RA      ';
   GRC:        DELIMSTRING := 'RC      ';
   GRESTRICT:  DELIMSTRING := 'RESTRICT';
   GRF:        DELIMSTRING := 'RF      ';
   GRN:        DELIMSTRING := 'RN      ';
   GRS:        DELIMSTRING := 'RS      ';
   GSB:        DELIMSTRING := 'SB      ';
   GSD:        DELIMSTRING := 'SD      ';
   GT:         DELIMSTRING := 'T       ';
   GTHEN:      DELIMSTRING := 'THEN    ';
   GY:         DELIMSTRING := 'Y       ';
   GZ:         DELIMSTRING := 'Z       ';
   GDEQ:       DELIMSTRING := 'D=      ';
   GSEQ:       DELIMSTRING := 'S=      ';
   GHASHEQ:    DELIMSTRING := '#=      ';
   GDOLEQ:     DELIMSTRING := '$=      '
   END;

TERMCHARS := [' ', '#', '$', ''''..')', '.'..'9', ';'..'?', '[', ']'];
M := 0;
RESULT := PENDING;
WHILE RESULT = PENDING DO
   IF M=8
      THEN IF BUFFER[N+M] IN TERMCHARS
```

```
                     THEN RESULT := FOUND
                     ELSE RESULT := NOTFOUND
              ELSE IF (BUFFER[N+M] = DELIMSTRING [M+1])
                      THEN (* match found *) IF BUFFER[N+M] = ' '
                           THEN RESULT := FOUND (* i.e. match is on the space *)
                           ELSE M:=M+1 (* delimiter is still being read *)
                      ELSE (* no match *) IF(DELIMSTRING[M+1] <> ' ')
                           THEN RESULT := NOTFOUND (* not end of delimiter *)
                           ELSE IF BUFFER[N+M] IN TERMCHARS
                                   THEN RESULT := FOUND (* terminated *)
                                   ELSE IF TESTDELIM IN [GDEQ..GDOLEQ]
                                           THEN RESULT := FOUND (* no termination needed *)
                                           ELSE RESULT := NOTFOUND (* not terminated *);
         IF RESULT = FOUND
            THEN BEGIN
                  CHECK := TRUE;
                  N := N + M + ORD(M=7)
                 END
            ELSE CHECK := FALSE
         END;


         PROCEDURE FINDNOMEN(VAR NOMENVAL : STRING32);

         (* Extracts characters from BUFFER until nomenclature is correctly terminated.
            If there are less than 32 characters before termination, then NOMENVAL
            is packed with spaces; if more then the excess is discarded. A number of
            of right parentheses equal to the number of left parentheses encountered
            is accepted before a right parenthesis terminates the nomenclature. *)

         VAR TERMINATED   : BOOLEAN;
             M            : 1..32;
             BRACKETCOUNT : 0..MAXLENGTH;
             TERMCHARS    : SET OF CHAR;

         BEGIN
         TERMCHARS := [';', ' ', '[', '/', '''', '<', '.'];
         TERMINATED := FALSE;
         BRACKETCOUNT := 0;
         REPEAT
```

```
 IF BUFFER[N] IN TERMCHARS
    THEN TERMINATED := TRUE
    ELSE IF BUFFER[N]='('
            THEN BRACKETCOUNT := BRACKETCOUNT+1
            ELSE IF BUFFER[N]=')'
                    THEN IF BRACKETCOUNT>0
                            THEN BRACKETCOUNT := BRACKETCOUNT-1
                            ELSE TERMINATED := TRUE;
 IF NOT TERMINATED THEN N := N+1
UNTIL TERMINATED;
FOR M := 1 TO 32 DO
   IF (M+VALIDLENGTH) < N
      THEN NOMENVAL[M] := BUFFER[M+VALIDLENGTH]
      ELSE NOMENVAL[M] := ' ';
TOKEN.NATURE := NOMENCLATURE
END;




FUNCTION CHECKINT : BOOLEAN;

{ Returns TRUE if the token beginning at the current position in BUFFER is
  an integer, and not nomenclature beginning with a digit. It checks this
  by seeing if any leading digits, hyphens and commas are followed by an
  alphabetic letter other than an R alone (as in substituent groups). }

VAR M         : 0..MAXLENGTH;

BEGIN
M := 0;
WHILE BUFFER[N+M] IN ['0'..'9', '-', ','] DO M := M+1;
IF BUFFER[N+M] IN ['A'..'Z']
   THEN IF BUFFER[N+M] = 'R'
           THEN CHECKINT := NOT (BUFFER[N+M+1] IN ['A'..'Z'])
           ELSE CHECKINT := FALSE
   ELSE CHECKINT := TRUE
END;
```

```
FUNCTION EXTRACTINT : INTEGER;

{ Returns the integer at the current position in BUFFER }

VAR INTBUFF : ARRAY[1..9] OF CHAR;
    INT,M,J,
    K,MULT  : INTEGER;

BEGIN
M := 0;
WHILE (BUFFER[N+M] IN ['0'..'9']) AND (M<9) DO
   BEGIN
    INTBUFF[M+1] := BUFFER [N+M];
    M := M+1
   END;
INT := 0;
FOR J := 0 TO (M-1) DO
   BEGIN
    MULT := 1;
    FOR K := 1 TO J DO MULT := MULT*10;
    INT := INT + MULT * ( ORD (INTBUFF[M-J]) -ORD('0'))
   END;
EXTRACTINT := INT;
TOKEN.NATURE := INTEGRAL;
N := N + M
END;






BEGIN     (* Body of Procedure GETTOKEN *)

REPEAT
 IF N=MAXLENGTH THEN READLINE;
 WHILE (BUFFER[N]=' ') AND (N<MAXLENGTH) DO N := N+1;
 STARTED := N<MAXLENGTH;
UNTIL STARTED;
VALIDLENGTH := N-1;
```

```
        WITH TOKEN DO
         IF BUFFER[N] IN ['A'..'E', 'I', 'M'..'T', 'Y', 'Z', '#', '$', ''''..')', '+'..'/', ';'..'?', '[', ']']
          THEN BEGIN
                CASE BUFFER[N] OF
                'A': DELIMVAL := GAND;
                'B': DELIMVAL := GBEGIN;
                'C': DELIMVAL := GC;
                'D': DELIMVAL := GDEQ;
                'E': IF BUFFER[N+1] = 'L'
                      THEN DELIMVAL := GELSE
                      ELSE IF BUFFER[N+1] = 'N'
                             THEN DELIMVAL := GEND
                             ELSE DELIMVAL := GE;
                'I': IF BUFFER[N+1] = 'F' THEN DELIMVAL := GIF
                                          ELSE DELIMVAL := GINPUT;
                'M': DELIMVAL := GM;
                'N': DELIMVAL := GN;
                'O': IF BUFFER[N+1] = 'R'
                      THEN IF BUFFER[N+2] = 'B' THEN DELIMVAL := GORBY
                                                ELSE DELIMVAL := GOR
                      ELSE DELIMVAL := GOSB;
                'P': DELIMVAL := GP;
                'Q': IF BUFFER[N+1] = 'U' THEN DELIMVAL := GQUERY
                                          ELSE DELIMVAL := GQ;
                'R': IF BUFFER[N+1] IN ['A', 'C', 'E', 'N', 'S']
                        THEN CASE BUFFER[N+1] OF
                              'A' : DELIMVAL := GRA;
                              'C' : DELIMVAL := GRC;
                              'E' : DELIMVAL := GRESTRICT;
                              'F' : DELIMVAL := GRF;
                              'N' : DELIMVAL := GRN;
                              'S' : DELIMVAL := GRS
                              END
                        ELSE DELIMVAL := GR;
                'S': IF BUFFER[N+1] = 'B'
                      THEN DELIMVAL := GSB
                      ELSE IF BUFFER[N+1] = 'D'
                             THEN DELIMVAL := GSD
                             ELSE DELIMVAL := GSEQ;
                'T': IF BUFFER[N+1] = 'H' THEN DELIMVAL := GTHEN
```

```
                                          ELSE DELIMVAL := GT;
            'Y': DELIMVAL := GY;
            'Z': DELIMVAL := GZ;
            '&': DELIMVAL := GAMPERSAND;
            '''': DELIMVAL := GPRIME;
            '(': DELIMVAL := GLPAREN;
            ')': DELIMVAL := GRPAREN;
            '+': DELIMVAL := GPLUS;
            ',': DELIMVAL := GCOMMA;
            '-': DELIMVAL := GHYPHEN;
            '.': DELIMVAL := GPERIOD;
            '/': DELIMVAL := GSLASH;
            ';': DELIMVAL := GSEMI;
            '<': IF BUFFER[N+1] = '>' THEN DELIMVAL := GNOTEQ
                                      ELSE DELIMVAL := GOPENANG;
            '=': DELIMVAL := GEQUALS;
            '>': DELIMVAL := GCLOSANG;
            '?': DELIMVAL := GQUEST;
            '[': DELIMVAL := GLSQUARE;
            ']': DELIMVAL := GRSQUARE;
            '#': DELIMVAL := GHASHEQ;
            '$': DELIMVAL := GDOLEQ
            END   (* of case *);
            IF DELIMVAL >= GAND
                THEN IF NOT CHECK(DELIMVAL) THEN FINDNOMEN(NOMENVAL)
                                            ELSE NATURE := DELIMITER
                ELSE BEGIN
                      NATURE := DELIMITER;
                      IF DELIMVAL=GNOTEQ THEN N := N + 2
                                         ELSE N := N + 1
                     END
           END     (* of IF BUFFER[N] THEN  *)
      ELSE IF BUFFER[N] IN ['0'..'9']
              THEN IF CHECKINT THEN INTEGVAL := EXTRACTINT
                               ELSE FINDNOMEN(NOMENVAL)
              ELSE FINDNOMEN(NOMENVAL)
   END;


            { END OF PROCEDURE GETTOKEN (THE LEXICAL ANALYZER)
```

```
--------------------------------------------------------------------------}



PROCEDURE NEXTTOKEN;

{ Obtains the next token, either from the queue of tokens already produced
  by LOOKAHEAD, or by a direct call to GETTOKEN. }

VAR TPTR : PTOKENLIST;

BEGIN
IF TOKENLIST = NIL
   THEN GETTOKEN
   ELSE BEGIN
          TOKEN := TOKENLIST^.TOKENVAL;
          TPTR := TOKENLIST;
          TOKENLIST := TOKENLIST^.NEXT;
          DISPOSE(TPTR)
        END
END;



PROCEDURE LOOKAHEAD;

{ If TOKENLIST is NIL (i.e. this is the first lookahead) then the current TOKEN
  is placed at the bottom of a queue of tokens (TOKENLIST). GETTOKEN is used to
  obtain a new token from the input stream, which is also added to the bottom
  of the queue. The next call to NEXTTOKEN will therefore restore the original
  TOKEN, and the subsequent call will return the following token.
  Called by ALTNVLIST\POSITIONSET
            ASSIGNMENTSTMNT\SUBSTGROUP}

VAR TOKENPTR : PTOKENLIST;

BEGIN
IF TOKENLIST=NIL
   THEN BEGIN
```

```
                NEW(TOKENLIST);
                TOKENLIST^.TOKENVAL := TOKEN;
                TOKENLIST^.NEXT := NIL
               END;
        TOKENPTR := TOKENLIST;
        WHILE TOKENPTR^.NEXT <> NIL DO TOKENPTR := TOKENPTR^.NEXT;
        NEW(TOKENPTR^.NEXT);
        TOKENPTR := TOKENPTR^.NEXT;
        GETTOKEN;
        TOKENPTR^.TOKENVAL := TOKEN;
        TOKENPTR^.NEXT := NIL
        END;



        PROCEDURE ERROR (ERRORCODE, DATA : INTEGER);

        { Outputs an appropriate error message, and either obtains a replacement
          TOKEN, or calls FAILURE. }

        VAR TOKENLENGTH,
            M              : INTEGER;
            TOKENPTR       : PTOKENLIST;

        BEGIN
        FOR M := 1 TO (13 + 7*ORD(DIAGNOSTICS) + VALIDLENGTH) DO WRITE(' ');
        TOKENLENGTH := N - VALIDLENGTH - 1;
        FOR M := 1 TO TOKENLENGTH DO WRITE('^');
        WRITELN;
        WRITELN('**** ERROR',ERRORCODE : 2);
        WRITEMESSAGE(ERRORCODE, DATA, '    ');
        WRITELN;
        CASE INPUTMODE OF
         STOREDGENSAL : FAILURE(40, 0, '    ');
         INSERTTEXT   : PROGERROR(2); {Error in SPSFILE expression}
         TERMINAL     : BEGIN
                          WRITELN('Remainder of input line ignored');
                          WRITELN;
                          FOR M := (VALIDLENGTH + 1) TO MAXLENGTH DO CURRENTLINE^.LINE[M] := ' ';
                          N := MAXLENGTH;
```

```
                  GETTOKEN;
                  IF TOKENLIST <> NIL THEN
                     BEGIN
                      {Need to put this token at the bottom of the list to
                      replace the erroneous one. }
                      TOKENPTR := TOKENLIST;
                      WHILE TOKENPTR^.NEXT <> NIL DO TOKENPTR := TOKENPTR^.NEXT;
                      TOKENPTR^.TOKENVAL := TOKEN
                     END
                 END
  END
 END;



 FUNCTION MAGNITUDE(BOND : BONDORDER) : INTEGER;

 { Returns an integer between 1 and 3 for the magnitude of BOND.
   Called from PROCESSCT\HNUMBER
               GETAVAILABLEPOSITIONS\MINPARENTBOND
               GETAVAILABLEPOSITIONS\SUMFILIALS
               PROCESSCT\GETPOSNS\GETSETPOSNS
               PROCESSCT\GETPOSNS
               ALTNVLIST\ELEMENT\SETCOMBARS\NEEDTOCHECK
               ALTNVLIST\ELEMENT\SETCOMBARS\COMBINEDPOSITIONS
               ALTNVLIST\ELEMENT\SETCOMBARS
               ALTNVLIST\GETCHILDPOSITIONS
               ALTNVLIST\PPOSNS\LMAGNOCHECKS
               ASSIGNMENTSTMNT\POINTERLIST\ADDCOMBSUBS}


 BEGIN
 CASE BOND OF
  NOTSPECIFIED,
  ANY, SINGLE,
  CHAIN, CHAISING,
  RING, RINGSING      : MAGNITUDE := 1;
  DOUBLE, CHAIDOUB,
  RINGDOUB, AROMATIC,
  RINGTAUT, CHAITAUT  : MAGNITUDE := 2;
```

```
  TRIPLE,
  CHAITRIP, RINGTRIP  : MAGNITUDE := 3
  END
 END;



 {-----------------------------------------------------------------------}
 PROCEDURE GETAVAILABLEPOSITIONS(PTRPS     : PTRPSTYPE;
                                 VAR POSNS : INTEGSET;
                                 BONDMAG   : TBONDMAG);

 { Returns in POSNS those positions of PTRPS^ wchich are substitutable, having a
   sufficient number of spare valencies to accomodate a bond of magnitude BONDMAG.
   Called by PROCESSCT\GETPOSNS
             ALTNVLIST\ELEMENT\TRANSLATENOMEN\MODIFYGATEPOSIIONS
             ALTNVLIST\ELEMENT\SETCOMBARS\COMBINEDPOSITIONS
             ALTNVLIST\ELEMENT\PARAMETERLIST\FINDCONNECTIONS
             ALTNVLIST\ALTNTVE\ADDPARALT
             ALTNVLIST\PPOSNS
             ASSIGNMENTSTMNT\POINTERLIST\ADDCOMBSUBS}

 VAR ROWNO   : ATOMNUMBER;



 FUNCTION MINBOND(OLDMAG,
                  NEWMAG : TBONDMAG) : TBONDMAG;

 { Returns the smaller of the two values passed as parameter }

 BEGIN
 IF NEWMAG < OLDMAG
    THEN MINBOND := NEWMAG
    ELSE MINBOND := OLDMAG
 END;



 FUNCTION MINPARENTBOND(PARENTGATE : PPARENTLIST;
```

```
                      ROWNO      : ATOMNUMBER): TBONDMAG;

{ Returns the magnitude of the smallest BOND to a parent if all items in the
  list have ROWNO as the only element of CHILDPOSITIONS (i.e. there are no
  alternatives). Otherwise, or if there is no parent list (PARENTGATE=NIL), returns 0.}

VAR VALID      : BOOLEAN;
    COMBPOSNS : PDOUBLIST;
    MINPB      : TBONDMAG;

BEGIN
VALID := PARENTGATE <> NIL;    {initialisation}
MINPB := 3;     {initialise to large value}
WHILE VALID AND (PARENTGATE<>NIL) DO WITH PARENTGATE^ DO
   BEGIN
    IF CHILDPOSITIONS.COMBINED
       THEN BEGIN
             VALID := CHILDPOSITIONS.COMBMEMS <> NIL;
             COMBPOSNS := CHILDPOSITIONS.COMBMEMS;
             IF CONNBONDS.CONNECTIONS <> 2
                THEN PROGERROR(3); {Combined childpositions with connections <> 2}
             WHILE VALID AND (COMBPOSNS<>NIL) DO WITH COMBPOSNS^ DO
                BEGIN
                 WITH CONNBONDS DO
                    IF (ROWNO=FIRST) AND (ROWNO=SECOND)
                       THEN MINPB := MINBOND(MINPB, MAGNITUDE(BONDA) + MAGNITUDE(BONDB))
                       ELSE IF ROWNO = FIRST
                               THEN MINPB := MINBOND(MINPB, MAGNITUDE(BONDA))
                               ELSE IF ROWNO = SECOND
                                       THEN MINPB := MINBOND(MINPB, MAGNITUDE(BONDB))
                                       ELSE VALID := FALSE;
                 COMBPOSNS := NEXT
                END
            END
       ELSE IF [ROWNO] = CHILDPOSITIONS.MEMBERS
               THEN BEGIN
                     IF CONNBONDS.CONNECTIONS <> 1
                        THEN PROGERROR(4); {Uncombined childpositions with CONNECTIONS <> 1}
                     MINPB := MINBOND(MINPB, MAGNITUDE(CONNBONDS.BOND))
                    END
```

```
                ELSE VALID := FALSE;
     PARENTGATE := NEXT
    END;
 IF VALID THEN MINPARENTBOND := MINPB
          ELSE MINPARENTBOND := 0
 END;



 FUNCTION SUMFILIALS(CONGENERS : CONGARRAY) : INTEGER;

 { Returns the sum of the MAGNITUDES of FILIAL bonds }

 VAR CNGNR : 1..MAXCONGENERS;
     SF    : INTEGER;

 BEGIN
 SF := 0;
 FOR CNGNR := 1 TO MAXCONGENERS DO WITH CONGENERS[CNGNR] DO
    IF RELATIONSHIP = FILIAL
       THEN SF := SF + MAGNITUDE(BOND);
 SUMFILIALS := SF
 END;



 BEGIN  {Body of GETAVAILABLEPOSITIONS}
 CASE PTRPS^.PSVARIETY OF
  DUMMY,
  UNKNOWN,
  OTHER    : POSNS := [1..MAXCT];
  GENERIC  : WITH PTRPS^, PARAMLIST[ATOMCOUNT] DO
                IF TOPRANGE = NOTSET
                   THEN IF SUBRANGES = NIL
                           THEN POSNS := []
                           ELSE POSNS := [1..SUBRANGES^.SECOND]
                   ELSE POSNS := [1..MAXCT];
  SPECIFIC : BEGIN
              POSNS := [];
              FOR ROWNO := 1 TO MAXCT DO IF PTRPS^.CT[ROWNO] <> NIL
```

```
                    THEN IF PTRPS^.CT[ROWNO]^.ATOMICROW
                            THEN IF (PTRPS^.CT[ROWNO]^. HYDROGENS
                                     - MINPARENTBOND(PTRPS^.PARENTGATE, ROWNO)
                                     - SUMFILIALS(PTRPS^.CT[ROWNO]^.CONGENERS))
                                     >= BONDMAG
                                        THEN POSNS := POSNS + [ROWNO]
                END
     END
    END;                        { of GETAVAILABLEPOSITIONS
    ---------------------------------------------------------------------------}



    PROCEDURE LISTPOSNS(VAR LISTPTR : PDOUBLIST;
                        POSNSETA,
                        POSNSETB,
                        COMBPOSNS   : INTEGSET);

    { Returns a linked list of pairs of positions, being all the possible
      combinations of the positions in POSNSETA and POSNSETB. The values in
      any one item may only be identical if that value is in COMBPOSNS
      Called by PROCESSCT\GETPOSNS
                ALTNVLIST\GETCHILDPOSITIONS
                ALTNVLIST\MODIFYCHILDPOSITIONS\GETCOMBPOSNS
                ALTNVLIST\ELEMENT\SETCOMBARS\COMBINEDPOSITIONS
                ALTNVLIST\ELEMENT\EXTRALAYER
                ALTNVLIST\ELEMENT\GETLIMITPOSITIONS
                ASSIGNMENTSTMNT\POINTERLIST\ADDCOMBSUBS
                ALTNVLIST\ELEMENT\PARAMETERLIST\FINDCONNECTIONS}

    VAR POSNA,
        POSNB   : INTEGER;
        NEWITEM : PDOUBLIST;

    BEGIN
    FOR POSNA := 0 TO MAXCT DO
       IF POSNA IN POSNSETA
          THEN FOR POSNB := 0 TO MAXCT DO
                  IF (POSNB IN POSNSETB) AND ((POSNB<>POSNA) OR (POSNB IN COMBPOSNS))
                     THEN BEGIN
```

```
                            NEW(NEWITEM);
                            ECTRSIZE := ECTRSIZE + 6;
                            NEWITEM^.FIRST := POSNA;
                            NEWITEM^.SECOND := POSNB;
                            NEWITEM^.NEXT := LISTPTR;
                            LISTPTR := NEWITEM
                           END
END;



PROCEDURE ADDTOLIST(NEWPS : PTRPSTYPE);

{ Adds a PS to the bottom of the list of PSs.
  Called by COPYPS
            ALTNVLIST\ELEMENT\SETCOMBARS
            ALTNVLIST\ELEMENT\SUBSTASVALUE}

VAR NEWIRITEM : PIRLIST;

BEGIN
NEW(NEWIRITEM);
ECTRSIZE := ECTRSIZE + 4;
NEWIRITEM^.PARSTRUCT := NEWPS;
NEWIRITEM^.NEXT := NIL;
IRLISTBOT^.NEXT := NEWIRITEM;
IRLISTBOT := NEWIRITEM
END;



FUNCTION COPYPS(OLDPS : PTRPSTYPE) : PTRPSTYPE;

{ Copies a PS.
  Called by COPYCOMBAR
            ALTNVLIST\ELEMENT\SUBSTASVALUE
            ALTNVLIST\ELEMENT\SETCOMBARS}

VAR NEWPS : PTRPSTYPE;

```

```
BEGIN
CASE OLDPS^.PSVARIETY OF
 DUMMY    : BEGIN
             NEW(NEWPS, DUMMY);
             ECTRSIZE := ECTRSIZE + 8;
             NEWPS^.SUBSTNAME := OLDPS^.SUBSTNAME
            END;
 UNKNOWN  : BEGIN
             NEW(NEWPS, UNKNOWN);
             ECTRSIZE := ECTRSIZE + 6
            END;
 OTHER    : BEGIN
             NEW(NEWPS, OTHER);
             ECTRSIZE := ECTRSIZE + 22;
             NEWPS^.TERM := OLDPS^.TERM
            END;
 SPECIFIC : BEGIN
             NEW(NEWPS, SPECIFIC);
             ECTRSIZE := ECTRSIZE + 70;
             NEWPS^.CT := OLDPS^.CT
            END;
 GENERIC  : BEGIN
             NEW(NEWPS, GENERIC);
             ECTRSIZE := ECTRSIZE + 50;
             NEWPS^.PARAMLIST := OLDPS^.PARAMLIST
            END
 END;
NEWPS^.VISITED := FALSE;
NEWPS^.PARENTGATE := NIL;
NEWPS^.CHILDGATE := NIL;
NEWPS^.PSVARIETY := OLDPS^.PSVARIETY;
ADDTOLIST(NEWPS);
COPYPS := NEWPS
END;



PROCEDURE COPYCOMBAR(VAR NEWCOMBAR : PCOMBINLIST;
                         OLDCOMBAR,
                         LASTCOMBLAYER : PCOMBINLIST;
```

```
                         LASTPPOSNS    : PTGROUPMEMS;
                         PRNTPS        : PTRPSTYPE;
                         FIRSTBAR,
                         OMITPG,
                         COPYPSS       : BOOLEAN);
FORWARD;



PROCEDURE COPYALTBAR(VAR NEWALTBAR : PALTERNLIST;
                         OLDCOMBLIST,
                         LASTCOMBLAYER : PCOMBINLIST;
                         LASTPPOSNS    : PTGROUPMEMS;
                         PRNTPS        : PTRPSTYPE;
                         FIRSTBAR,
                         OMITPG,
                         COPYPSS       : BOOLEAN);

{ Creates a new alternative bar item, and calls COPYCOMBAR to copy the
  combination bar items in OLDCOMBLIST into its COMBINATION field.
  Called by COPYCOMBAR
            ALTNVLIST\ELEMENT\SUBSTASVALUE}
VAR NEWAB : PALTERNLIST;

BEGIN
NEW(NEWAB);
ECTRSIZE := ECTRSIZE + 4;
NEWAB^.COMBINATION := NIL;
WHILE OLDCOMBLIST <> NIL DO
   BEGIN
    COPYCOMBAR(NEWAB^.COMBINATION, OLDCOMBLIST, LASTCOMBLAYER, LASTPPOSNS, PRNTPS, FIRSTBAR, OMITPG, COPYPSS);
    OLDCOMBLIST := OLDCOMBLIST^.NEXT
   END;
NEWAB^.NEXT := NEWALTBAR;
NEWALTBAR := NEWAB
END;
```

```
PROCEDURE SETCONNBONDS(VAR CONNBONDS : TCONNBONDS;
                           CONNECTIVITY  : TCONNS);

{ Returns a TCONNBONDS record with CONNECTIONS set to CONNECTIVITY and all
  bond types to NOTSPECIFIED.
  Called by SUBSTGROUP\CHECKCOMPATIBILITY
            ALTNVLIST\ELEMENT\VALIDSUBST
            ALTNVLIST\ELEMENT
            ASSIGNMENTSTMNT\POINTERLIST\ADDDEFNTABLE
            DECLARESUBST}

BEGIN
WITH CONNBONDS DO
   BEGIN
    CONNECTIONS := CONNECTIVITY;
    CASE CONNECTIONS OF
     NOTSET,
     0      : ;
     1      : BOND := NOTSPECIFIED;
     2      : BEGIN
                BONDA := NOTSPECIFIED;
                BONDB := NOTSPECIFIED
              END
     END
   END
END;



PROCEDURE UPDATEPPSCONNS(PARPSLIST : PPSLIST);

{ Copies the CONNBONDS field of the first item in PARPSLIST right down the
  list.
  Called by SUBSTGROUP\CHECKCOMPATIBILITY
            ALTNVLIST\ELEMENT\VALIDSUBST
            DECLARESUBST}

VAR NEWCONNBONDS : TCONNBONDS;

BEGIN
```

```
NEWCONNBONDS := PARPSLIST^.CONNBONDS;
REPEAT
 PARPSLIST^.CONNBONDS := NEWCONNBONDS;
 PARPSLIST := PARPSLIST^.NEXT
UNTIL PARPSLIST = NIL
END;



PROCEDURE DECLARESUBST(SUBST          : SUBSTITUENT;
                       PSADDRESS,
                       SAVPS          : PTRPSTYPE;
                       CONNBONDS      : TCONNBONDS;
                       PRNTPOSNS      : PTGROUPMEMS);

{ Adds declaration of SUBST to RDECLARATIONTABLE, referencing PSADDRESS.
  If this is the first declaration of SUBST to give a value for connectivity,
  then the connectivity is copied into all the other declarations.
  Called by PROCESSCT
            COPYCOMBAR
            ALTNVLIST\ELEMENT\PARAMETERLIST\USERPARAMETER
            ALTNVLIST\ELEMENT\SUBSTASVALUE}

VAR PTR : PPSLIST;

BEGIN
IF RDECLARATIONTABLE[SUBST] <> NIL
   THEN IF (RDECLARATIONTABLE[SUBST]^.CONNBONDS.CONNECTIONS = NOTSET)
                AND (CONNBONDS.CONNECTIONS <> NOTSET)
          THEN BEGIN
                 SETCONNBONDS(RDECLARATIONTABLE[SUBST]^.CONNBONDS, CONNBONDS.CONNECTIONS);
                 UPDATEPPSCONNS(RDECLARATIONTABLE[SUBST])
               END;
NEW(PTR);
PTR^.PARSTRUCT := PSADDRESS;
PTR^.COMBINS     := NIL;
PTR^.FURTHERSUB  := SAVPS;
PTR^.CONNBONDS   := CONNBONDS;
PTR^.PRNTPOSNS   := PRNTPOSNS;
PTR^.COPYCHILDPS := FALSE;
```

```
PTR^.NEXT := RDECLARATIONTABLE[SUBST];
RDECLARATIONTABLE[SUBST] := PTR;
DECLSUBS := DECLSUBS + [SUBST]
END;



PROCEDURE COMPARELISTS(LOWERLIST,
                       UPPERLIST : PDOUBLIST);

{ Calls FAILURE if the items in LOWERLIST are not identical with those
  in UPPERLIST.
  Called by CHECKINCLUDED}

VAR PTR   : PDOUBLIST;
    FOUND : BOOLEAN;

BEGIN
WHILE LOWERLIST <> NIL DO
   BEGIN
    PTR := UPPERLIST;
    WHILE PTR <> NIL DO
       BEGIN
        FOUND := (LOWERLIST^.FIRST = PTR^.FIRST) AND (LOWERLIST^.SECOND = PTR^.SECOND);
        IF FOUND THEN PTR := NIL
                 ELSE PTR := PTR^.NEXT
       END;
    IF FOUND THEN LOWERLIST := LOWERLIST^.NEXT
             ELSE FAILURE(39, 0, '    ')
   END
END;



PROCEDURE CHECKALLWITHIN(COMBMEMS   : PDOUBLIST;
                         AVAILPOSNS : INTEGSET;
                         FAILCODE   : INTEGER);

{ Checks that all the items in COMBMEMS are within AVAILPOSNS
  Called by ELEMENT\TRANSLATENOMEN\MODIFYGATEPOSITIONS
```

```
                ALTNVLIST\PPOSNS\LMAGCHECKS
                CHECKINCLUDED}

BEGIN
WHILE COMBMEMS <> NIL DO WITH COMBMEMS^ DO
   BEGIN
    IF [FIRST, SECOND] <= AVAILPOSNS
       THEN {OK}
       ELSE IF FIRST IN AVAILPOSNS
               THEN FAILURE(FAILCODE, SECOND, '    ')
               ELSE FAILURE(FAILCODE, FIRST, '    ');
    COMBMEMS := NEXT
   END
END;



PROCEDURE CHECKINCLUDED(LOWERPOSNS,
                        UPPERPOSNS : PTGROUPMEMS);

{ Checks that all the positions in LOWERPOSNS are also in UPPERPOSNS
 Called by COPYCOMBAR}

BEGIN
IF LOWERPOSNS^.COMBINED
   THEN IF UPPERPOSNS^.COMBINED
           THEN COMPARELISTS(LOWERPOSNS^.COMBMEMS, UPPERPOSNS^.COMBMEMS)
           ELSE CHECKALLWITHIN(LOWERPOSNS^.COMBMEMS, UPPERPOSNS^.MEMBERS, 39)
   ELSE IF UPPERPOSNS^.COMBINED
           THEN PROGERROR(26) {Trying to uncombine a position set}
           ELSE IF LOWERPOSNS^.MEMBERS <= UPPERPOSNS^.MEMBERS
                   THEN {OK}
                   ELSE FAILURE(39, 0, '    ')
END;




PROCEDURE COPYCOMBAR; {Previous FORWARD declaration}
```

```
{ Copies a combination bar item. If  FIRSTBAR then the PARENTPOSITIONS field
  is altered to LASTPPOSNS. In  other cases LASTPPOSNS is changed to
  OLDCOMBAR^.PARENTPOSITIONS after checking  that the specified positions are
  available.   For BOTTOMBARs a new parent gate is created on the existing
  Child PS (not done if OMITPG).   If COPYPSS is TRUE, then Child PSs
  themselves are copied, otherwise the new  Gate is simply made to point to the
  Child PS.   For non-BOTTOMBARs, COPYALTBAR is called to copy the ALTERNATIVES.
  Called by COPYALTBAR
            COPYCOMBAR (recursively)
            ENTERCOMBIN
            ALTNVLIST\ADDFURTHERSUBTN}

VAR NEWCB,
    SUBCB     : PCOMBINLIST;
    OLDALTBAR : PALTERNLIST;
    NEWPG     : PPARENTLIST;



BEGIN
IF OLDCOMBAR^.BOTTOMBAR
   THEN NEW(NEWCB, TRUE)
   ELSE NEW(NEWCB, FALSE);
ECTRSIZE := ECTRSIZE + 11 + ORD(OLDCOMBAR^.BOTTOMBAR) * 13;
IF FIRSTBAR
   THEN NEWCB^.PARENTPOSITIONS := LASTPPOSNS
   ELSE IF OLDCOMBAR^.PARENTPOSITIONS = NIL
           THEN NEWCB^.PARENTPOSITIONS := NIL
           ELSE BEGIN
                  NEWCB^.PARENTPOSITIONS := OLDCOMBAR^.PARENTPOSITIONS;
                  CHECKINCLUDED(NEWCB^.PARENTPOSITIONS, LASTPPOSNS);
                  LASTPPOSNS := NEWCB^.PARENTPOSITIONS
                END;
NEWCB^.FREQUENCY := OLDCOMBAR^.FREQUENCY;
NEWCB^.NEXT := NEWCOMBAR;
NEWCB^.BOTTOMBAR := OLDCOMBAR^.BOTTOMBAR;
IF NEWCB^.BOTTOMBAR
   THEN BEGIN
          IF COPYPSS
             THEN NEWCB^.CHILDPS := COPYPS(OLDCOMBAR^.CHILDPS)
```

```
                ELSE NEWCB^.CHILDPS := OLDCOMBAR^.CHILDPS;
             IF NEWCB^.CHILDPS^.PSVARIETY = DUMMY
                THEN BEGIN
                      DECLARESUBST(NEWCB^.CHILDPS^.SUBSTNAME,
                                   PRNTPS,
                                   NEWCB^.CHILDPS,
                                   OLDCOMBAR^.CONNBONDS,
                                   LASTPPOSNS);
                      RDECLARATIONTABLE[NEWCB^.CHILDPS^.SUBSTNAME]^.COMBINS := LASTCOMBLAYER
                     END;
             NEWCB^.CHILDPOSITIONS := OLDCOMBAR^.CHILDPOSITIONS;
             NEWCB^.CONNBONDS := OLDCOMBAR^.CONNBONDS;
             IF NOT OMITPG
                THEN BEGIN
                      NEW(NEWPG);
                      ECTRSIZE := ECTRSIZE + 26;
                      WITH NEWPG^ DO
                         BEGIN
                          CHILDPOSITIONS := NEWCB^.CHILDPOSITIONS;
                          PARENTPOSITIONS := LASTPPOSNS^;
                          PARENTPS := PRNTPS;
                          CONNBONDS := NEWCB^.CONNBONDS;
                          NEXT := NEWCB^.CHILDPS^.PARENTGATE
                         END;
                      NEWCB^.CHILDPS^.PARENTGATE := NEWPG
                     END;
             IF COPYPSS
                THEN BEGIN
                      SUBCB := OLDCOMBAR^.CHILDPS^.CHILDGATE;
                      WHILE SUBCB <> NIL DO
                         BEGIN
                          COPYCOMBAR(NEWCB^.CHILDPS^.CHILDGATE, SUBCB, NIL, NIL, NEWCB^.CHILDPS, FALSE, OMITPG
TRUE);
                          SUBCB := SUBCB^.NEXT
                         END
                     END
            END
         ELSE BEGIN
             OLDALTBAR := OLDCOMBAR^.ALTERNATIVES;
             NEWCB^.ALTERNATIVES := NIL;
```

```
              WHILE OLDALTBAR <> NIL DO
                 BEGIN
                  COPYALTBAR(NEWCB^.ALTERNATIVES, OLDALTBAR^.COMBINATION, NEWCB,
                           LASTPPOSNS, PRNTPS, FALSE, OMITPG, COPYPSS);
                  OLDALTBAR := OLDALTBAR^.NEXT
                 END
             END;
      NEWCOMBAR := NEWCB
      END;



      PROCEDURE ENTERCOMBIN(SUBST         : SUBSTITUENT;
                        VAR GATEENTRY : PCOMBINLIST);

      { If SUBST has been defined, copies the existing definition combination bar
        into GATEENTRY, otherwise creates a new non-BOTTOMBAR combination bar item.
        Called by PROCESSCT
                  ALTNVLIST\ELEMENT\PARAMETERLIST\USERPARAMETER}

      BEGIN
      WITH RDECLARATIONTABLE[SUBST]^ DO
         IF RDEFINITIONTABLE[SUBST] = NIL
            THEN BEGIN
                  NEW(COMBINS, TRUE);
                  ECTRSIZE := ECTRSIZE + 11;
                  COMBINS^.BOTTOMBAR := FALSE;
                  COMBINS^.PARENTPOSITIONS := PRNTPOSNS;
                  COMBINS^.FREQUENCY.TOPRANGE := NOTSET;
                  COMBINS^.FREQUENCY.SUBRANGES := ESSENTFREQ;
                  COMBINS^.ALTERNATIVES := NIL;
                  COMBINS^.NEXT := GATEENTRY;
                  GATEENTRY := COMBINS
                 END
            ELSE BEGIN
                  COPYCOMBAR(GATEENTRY,
                           RDEFINITIONTABLE[SUBST],
                           NIL,
                           PRNTPOSNS,
                           PARSTRUCT,
```

```
                        TRUE,
                        FALSE,
                        FALSE);
            COMBINS := GATEENTRY
           END
END;




{-----------------------------------------------------------------------------

                         PROCEDURE PROCESSCT

-----------------------------------------------------------------------------}

PROCEDURE PROCESSCT (VAR CT      : CTTYPE;
                     INTERACTIVE : BOOLEAN;
                     PSADDRESS   : PTRPSTYPE);

{ Carries out the reformatting of FELDCT/FELDBD, putting the result into CT.
  Called by READSD
            ALTNVLIST\ELEMENT\TRANSLATENOMEN}

TYPE TNODENATURE = (ATOMIC, APICLABEL, VARPOSNLABEL, EXPHYDROGEN, SUBSTNODE);

VAR APICCOUNT,                       { Number of apical labels present }
    ROWNO      : ATOMNUMBER;         { Row counter for CT and FELDCT }
    M          : 1..2;               { Counter for characters of atomic symbol }
    REJECTED   : BOOLEAN;            { Set to TRUE if error is found }
    CONNBONDS  : TCONNBONDS;
    PRNTPOSNS  : PTGROUPMEMS;



FUNCTION NODENATURE(NODE : ATOMNUMBER) : TNODENATURE;

{ Returns the nature of the NODE in FELDCT.
  Called by INDEPENDENT
            READCONGNERS
```

```
                   Body of PROCESSCT}

BEGIN
WITH FELDCT[NODE] DO
IF CHEM = '*   '
   THEN NODENATURE := APICLABEL
   ELSE IF CHEM = '#   '
           THEN NODENATURE := VARPOSNLABEL
           ELSE IF CHEM = 'H   '
                   THEN NODENATURE := EXPHYDROGEN
                   ELSE IF CHEM[2] IN ['0'..'9']
                           THEN NODENATURE := SUBSTNODE
                           ELSE NODENATURE := ATOMIC
END;



FUNCTION BONDVAL (NODEA,NODEB : ATOMNUMBER) : BONDORDER;

{ Finds the order of the bond between NODEA and NODEB }

VAR M      : 0..MAXCT;
    BNDVAL : BONDORDER;

BEGIN
BNDVAL := NOTSPECIFIED;
M := 0;
WHILE (M <= NUMOFBONDS) AND (BNDVAL=NOTSPECIFIED) DO
   BEGIN
    M := M+1;
    WITH FELDBD[M] DO
       IF ((NODEA=NODE1) AND (NODEB=NODE2))
          OR ((NODEA=NODE2) AND (NODEB=NODE1))
          THEN REPEAT BNDVAL := SUCC(BNDVAL)
                 UNTIL ORD(BNDVAL) = BOND
   END;
BONDVAL := BNDVAL
END;
```

```
FUNCTION SUBSTNAME(CHEM : STRING4): SUBSTITUENT;

{ Converts the name of a substituent to integer format from characters.
  Called by Body of PROCESSCT}

VAR SUBST : SUBSTITUENT;

BEGIN
IF CHEM[3] = ' '
   THEN SUBST := ORD(CHEM[2]) - ORD('0')
   ELSE SUBST := (ORD(CHEM[3]) - ORD('0')) + 10*(ORD(CHEM[2]) - ORD('0') );
SUBSTNAME := SUBST
END;



FUNCTION INDEPENDENT(NODENO : ATOMNUMBER): BOOLEAN;

{ Returns TRUE if any of the congeners of NODENO are EXPHYDROGEN, APICLABEL
  or SUBSTNODE.
  Called by Body of PROCESSCT}

VAR CNGNR : 1..MAXCONGENERS;

BEGIN
INDEPENDENT := FALSE;
FOR CNGNR := 1 TO MAXCONGENERS DO IF FELDCT[NODENO].AR[CNGNR] <> 0
  THEN IF NODENATURE(FELDCT[NODENO].AR[CNGNR]) IN [EXPHYDROGEN, APICLABEL, SUBSTNODE]
          THEN INDEPENDENT := TRUE
END;


PROCEDURE REJECT(ERRORCODE : INTEGER;
                 NODE      : ATOMNUMBER);

{ Outputs an error message.
  Called by READCONGENERS
            HNUMBER
            NUMOFCONNS
```

```
                    CHECKEARLIERDEFN
                    Body of PROCESSCT}

      BEGIN
      WRITE('SD ERROR: ');
      WRITEMESSAGE(ERRORCODE, NODE, '    ');
      REJECTED := TRUE
      END;



      PROCEDURE READCONGENERS (VAR CONGENERS : CONGARRAY;
                               VAR HYDROGENS : NUMCONGENERS;
                               ATOMICROW     : BOOLEAN;
                               ROWNO         : ATOMNUMBER);

      { Sets the values in CONGENERS, and the number
        of explicit attached HYDROGENS, for a single connection table ROW.
        Called by Body of PROCESSCT}

      VAR FELDCONG : ARRAY[1..MAXCONGENERS] OF ATOMNUMBER;
          CNGNR    : 1..MAXCONGENERS;

      BEGIN
      HYDROGENS := 0;
      FELDCONG := FELDCT[ROWNO].AR;
      FOR CNGNR := 1 TO MAXCONGENERS DO
         BEGIN
          CONGENERS[CNGNR].RELATIONSHIP := NONE;
          IF FELDCONG[CNGNR] <> 0 THEN WITH CONGENERS[CNGNR-HYDROGENS] DO
             BEGIN
              BOND := BONDVAL(ROWNO, FELDCONG[CNGNR]);
              CASE NODENATURE(FELDCONG[CNGNR]) OF
               ATOMIC       : BEGIN
                                RELATIONSHIP := FRATERNAL;
                                ROWNUM := FELDCONG[CNGNR]
                              END;
               EXPHYDROGEN  : HYDROGENS := HYDROGENS + 1;
               APICLABEL    : RELATIONSHIP := PARENTAL;
               VARPOSNLABEL : IF ATOMICROW
```

```
                                        THEN REJECT(56, ROWNO)
                                        ELSE BEGIN
                                               RELATIONSHIP := FRATERNAL;
                                               ROWNUM := NOTFIXED
                                              END;
                  SUBSTNODE     : IF INDEPENDENT(FELDCONG[CNGNR])
                                        THEN BEGIN
                                               RELATIONSHIP := FRATERNAL;
                                               ROWNUM := FELDCONG[CNGNR]
                                              END
                                        ELSE RELATIONSHIP := FILIAL
                  END
                END
            END
        END;



        PROCEDURE HNUMBER (NODE : ATOMNUMBER);

        { Sets a value for HYDROGENS at atom NODE, checking valencies in VELENCYFILE.
          Called by Body of PROCESSCT}

        CONST MAXSTATES = 5;

        TYPE TELEMVALS = RECORD
                           ELEMENT   : STRING2;
                           VALENCIES : PACKED ARRAY[1..MAXSTATES] OF NUMCONGENERS
                          END;

        VAR BONDCOUNT   : 0..18;              { Sum of bond orders }
            STATE       : INTEGER;
            ARCOUNT,                          { Number of aromatic bonds }
            TAUTCOUNT,                        { Number of tautomeric bonds }
            CNGNR       : 0..MAXCONGENERS;    { Congner counter }
            EXTERNBONDS,                      { Sum of MAGNITUDEs of external bonds }
            SPAREVALS   : INTEGER;            { Valency of common atom }
            ELEMVAL     : TELEMVALS;          { Element valency record }
            VALENCYFILE : FILE OF TELEMVALS;
```

```
BEGIN
BONDCOUNT := 0;
ARCOUNT   := 0;
TAUTCOUNT := 0;
EXTERNBONDS := 0;
FOR CNGNR := 1 TO MAXCONGENERS DO WITH CT[NODE]^.CONGENERS[CNGNR] DO
   BEGIN
    IF RELATIONSHIP <> NONE THEN
       CASE BOND OF
        NOTSPECIFIED, RINGSING, CHAISING,
         RING, CHAIN, SINGLE, ANY : BONDCOUNT := BONDCOUNT +1;
        RINGDOUB,CHAIDOUB, DOUBLE : BONDCOUNT := BONDCOUNT +2;
        RINGTRIP,CHAITRIP, TRIPLE : BONDCOUNT := BONDCOUNT +3;
        AROMATIC                  : ARCOUNT := ARCOUNT +1;
        RINGTAUT, CHAITAUT        : TAUTCOUNT := TAUTCOUNT +1
       END;
    IF RELATIONSHIP IN [FILIAL, PARENTAL]
       THEN EXTERNBONDS := EXTERNBONDS + MAGNITUDE(BOND)
   END;
CASE ARCOUNT OF
 0       : ;
 2,3     : BONDCOUNT := BONDCOUNT + ARCOUNT +1;
 1,4,5,6 : REJECT(60, NODE)
END;
CASE TAUTCOUNT OF
 0,1   : BONDCOUNT := BONDCOUNT+TAUTCOUNT;
 2,3   : BONDCOUNT := BONDCOUNT + TAUTCOUNT +1;
 4,5,6 : REJECT(61, NODE)
END;
RESET(VALENCYFILE, 'LI2GEN>VALENCYFILE');
ELEMVAL.ELEMENT := '  ';
WHILE(ELEMVAL.ELEMENT <> CT[NODE]^.ATOM) AND NOT EOF(VALENCYFILE) DO
   READ(VALENCYFILE, ELEMVAL);
IF EOF(VALENCYFILE)
   THEN {atom not in file}
   ELSE BEGIN
         STATE := 1;
         WHILE STATE <= MAXSTATES DO
            BEGIN
             SPAREVALS := ELEMVAL.VALENCIES[STATE] + CT[NODE]^.CHARGE - BONDCOUNT;
```

```
                  IF SPAREVALS < 0
                     THEN IF STATE < MAXSTATES
                             THEN STATE := STATE + 1
                             ELSE BEGIN
                                     REJECT(55, NODE);
                                     STATE := MAXSTATES+1
                                  END
                     ELSE IF SPAREVALS > 6
                             THEN PROGERROR(5) {Excessively large valency}
                             ELSE BEGIN
                                     CT[NODE]^.HYDROGENS := SPAREVALS + EXTERNBONDS;
                                     STATE := MAXSTATES + 1
                                  END
                  END
               END;
      RESET(VALENCYFILE, '@TTY')
      END;



      FUNCTION NUMOFCONNS(CONGENERS : CONGARRAY;
                          TOTCONNS  : NUMCONGENERS;
                          NODE      : ATOMNUMBER) : NUMCONGENERS;

      { Returns the number of connections specified in CONGENERS, plus the entry
        value of TOTCONNS (which corresponds to the number of HYDROGENS.
        Called by CHECKEARLIERDEFN
                  Body of PROCESSCT}

      VAR CNGNR : NUMCONGENERS;

      BEGIN
      FOR CNGNR := 1 TO MAXCONGENERS DO
         IF CONGENERS[CNGNR].RELATIONSHIP <> NONE
            THEN TOTCONNS := TOTCONNS + 1;
      IF TOTCONNS > 2 THEN REJECT(54, NODE);
      NUMOFCONNS := TOTCONNS
      END;
```

```
PROCEDURE CHECKEARLIERDEFN(SUBST    : SUBSTITUENT;
                           CONNS    : NUMCONGENERS;
                           ROWSREAD : ATOMNUMBER);

{ Examines the rows of the connection table up as far as ROWSREAD, and if a
  non-ATOMICROW is found with NAME=SUBST then compares the value of CONNS
  with the number of connections of this row.
  Called by Body of PROCESSCT}

VAR NODENO : ATOMNUMBER;

BEGIN
NODENO := 1;
WHILE NODENO < ROWSREAD DO
   IF CT[NODENO] = NIL
      THEN NODENO := NODENO + 1
      ELSE WITH CT[NODENO]^ DO
              IF ATOMICROW
                 THEN NODENO := NODENO + 1
                 ELSE IF NAME = SUBST
                         THEN BEGIN
                                IF CONNS = NUMOFCONNS(CONGENERS, HYDROGENS, NODENO)
                                   THEN {matches OK}
                                   ELSE REJECT(53, ROWSREAD);
                                NODENO := ROWSREAD
                              END
                         ELSE NODENO := NODENO + 1
END;


{..............................................................................}
PROCEDURE GETPOSNS(CONGENERS     : CONGARRAY;
               VAR CONNBONDS : TCONNBONDS;
               VAR PRNTPOSNS : PTGROUPMEMS);

{ Sets CONNBONDS, PRNTPOSNS  for a substituent, by examining CONGENERS.
  Called by Body of PROCESSCT}

VAR POSNS1,
```

```
         POSNS2,
         COMBPOSNS  : INTEGSET;
         MAGSUM     : INTEGER;
         REVERSIBLE : BOOLEAN;



      PROCEDURE GETSETPOSNS(VAR SETPOSNS : INTEGSET;
                                POSITION : ATOMNUMBER;
                                BOND     : BONDORDER);

      BEGIN
      IF POSITION = NOTFIXED
         THEN GETAVAILABLEPOSITIONS(PSADDRESS, SETPOSNS, MAGNITUDE(BOND))
         ELSE SETPOSNS := [POSITION]
      END;



      BEGIN {Body of GETPOSNS}
      NEW(PRNTPOSNS);
      ECTRSIZE := ECTRSIZE + 9;
      IF CONGENERS[1].RELATIONSHIP = NONE
         THEN BEGIN
              {substituent is unconnected}
              PRNTPOSNS^.COMBINED := FALSE;
              PRNTPOSNS^.MEMBERS := [];
              CONNBONDS.CONNECTIONS := 0
             END
         ELSE IF CONGENERS[2].RELATIONSHIP = NONE
                THEN BEGIN
                     {substituent is singly connected}
                     PRNTPOSNS^.COMBINED := FALSE;
                     CONNBONDS.CONNECTIONS := 1;
                     CONNBONDS.BOND := CONGENERS[1].BOND;
                     CASE CONGENERS[1].RELATIONSHIP OF
                      FRATERNAL : WITH CONGENERS[1] DO
                                     GETSETPOSNS(PRNTPOSNS^.MEMBERS, ROWNUM, BOND);
                      FILIAL    : PROGERROR(6); {substituent node with filial congener }
                      PARENTAL  : PRNTPOSNS^.MEMBERS := [0]
```

```
                    END
                  END
             ELSE BEGIN
                  {substituent is doubly connected}
                  PRNTPOSNS^.COMBINED := TRUE;
                  PRNTPOSNS^.COMBMEMS := NIL;
                  WITH CONNBONDS DO
                     BEGIN
                      CONNECTIONS := 2;
                      BONDA := CONGENERS[1].BOND;
                      BONDB := CONGENERS[2].BOND;
                      REVERSIBLE := (BONDMATCHARRAY[BONDA,BONDB] IN [ANY, CHAIN, RING]) OR (BONDA=BONDB)
                     END;
                  CASE CONGENERS[1].RELATIONSHIP OF
                   FRATERNAL :
                     CASE CONGENERS[2].RELATIONSHIP OF
                       FRATERNAL : BEGIN
                                   WITH CONGENERS[1] DO GETSETPOSNS(POSNS1, ROWNUM, BOND);
                                   WITH CONGENERS[2] DO GETSETPOSNS(POSNS2, ROWNUM, BOND);
                                   WITH CONNBONDS DO
                                      MAGSUM := MAGNITUDE(BONDA) + MAGNITUDE(BONDB);
                                   IF (POSNS1 * POSNS2 = []) OR (MAGSUM > 3)
                                      THEN COMBPOSNS := []
                                      ELSE BEGIN
                                             GETAVAILABLEPOSITIONS(PSADDRESS, COMBPOSNS, MAGSUM);
                                             COMBPOSNS := COMBPOSNS * POSNS1 * POSNS2
                                           END;
                                   LISTPOSNS(PRNTPOSNS^.COMBMEMS, POSNS1, POSNS2, COMBPOSNS);
                                   IF REVERSIBLE
                                      THEN LISTPOSNS(PRNTPOSNS^.COMBMEMS, POSNS2, POSNS1, COMBPOSNS
                                  END;
                       FILIAL    : PROGERROR(7); {substituent node with filial congner }
                       PARENTAL  : BEGIN
                                   WITH CONGENERS[1] DO GETSETPOSNS(POSNS1, ROWNUM, BOND);
                                   LISTPOSNS(PRNTPOSNS^.COMBMEMS, POSNS1, [0], []);
                                   IF REVERSIBLE THEN
                                      LISTPOSNS(PRNTPOSNS^.COMBMEMS, [0], POSNS1, [])
                                  END
                       END;
                   FILIAL    : PROGERROR(8); {substituent node with filial congener}
```

```
                 PARENTAL  :
                    CASE CONGENERS[2].RELATIONSHIP OF
                     FRATERNAL : BEGIN
                                  WITH CONGENERS[2] DO GETSETPOSNS(POSNS2, ROWNUM, BOND);
                                  LISTPOSNS(PRNTPOSNS^.COMBMEMS, [0], POSNS2, []);
                                  IF REVERSIBLE THEN
                                     LISTPOSNS(PRNTPOSNS^.COMBMEMS, POSNS2, [0], [])
                                 END;
                     FILIAL    : PROGERROR(9); {substituent node with filial congner }
                     PARENTAL  : BEGIN
                                  NEW(PRNTPOSNS^.COMBMEMS);
                                  ECTRSIZE := ECTRSIZE + 6;
                                  WITH PRNTPOSNS^.COMBMEMS^ DO
                                     BEGIN
                                      FIRST := 0;
                                      SECOND := 0;
                                      NEXT := NIL
                                     END;
                                 END
                     END
                 END
               END
END;
{.............................................................}



PROCEDURE DECLAREMULT(MULTIP    : MULTIPLIER;
                      MULTSUBST : SUBSTITUENT);

{ Adds an entry to MDECLARATIONTABLE for MULTIP.
  Called by Body of PROCESSCT}

VAR PMPTR : PMDECLIST;

BEGIN
NEW(PMPTR);
WITH PMPTR^ DO
   BEGIN
    SUBSTDECN := RDECLARATIONTABLE[MULTSUBST];
```

```
     NEXT := MDECLARATIONTABLE[MULTIP]
    END;
  MDECLARATIONTABLE[MULTIP] := PMPTR;
  DECLMULT := DECLMULT + [MULTIP]
  END;



  BEGIN (* Body of Procedure PROCESSCT *)
  REPEAT
   IF INTERACTIVE THEN READFELDMANN;
   APICCOUNT := 0;
   REJECTED := FALSE;
   FOR ROWNO := 1 TO MAXCT DO IF ROWNO > NUMOFNODES
     THEN CT[ROWNO] := NIL
     ELSE CASE NODENATURE(ROWNO) OF
            APICLABEL    : BEGIN
                             CT[ROWNO] := NIL;
                             IF FELDCT[ROWNO].AR[2] <> 0 THEN REJECT(58, ROWNO);
                             IF FELDCT[ROWNO].MULT <> 0 THEN REJECT(57, ROWNO);
                             IF APICCOUNT = 2
                                THEN REJECT(59, 0)
                                ELSE APICCOUNT := APICCOUNT + 1
                            END;
            VARPOSNLABEL,
            EXPHYDROGEN  : BEGIN
                             CT[ROWNO] := NIL;
                             IF FELDCT[ROWNO].AR[2] <> 0 THEN REJECT(58, ROWNO);
                             IF FELDCT[ROWNO].MULT <> 0 THEN REJECT(57, ROWNO)
                            END;
            ATOMIC       : BEGIN
                             IF FELDCT[ROWNO].MULT <> 0 THEN REJECT(57, ROWNO);
                             NEW(CT[ROWNO], TRUE);
                             ECTRSIZE := ECTRSIZE + 30;
                             WITH CT[ROWNO]^ DO
                                BEGIN
                                 ATOMICROW := TRUE;
                                 FOR M := 1 TO 2 DO ATOM[M] := FELDCT[ROWNO].CHEM[M];
                                 CHARGE := FELDCT[ROWNO].CHGE;
                                 READCONGENERS(CONGENERS, HYDROGENS, ATOMICROW, ROWNO);
```

```
                                    IF HYDROGENS=0 THEN HNUMBER(ROWNO)
                                   END
                              END;
            SUBSTNODE    : BEGIN
                               NEW(CT[ROWNO], FALSE);
                               ECTRSIZE := ECTRSIZE + 33;
                               WITH CT[ROWNO]^ DO
                                  BEGIN
                                   ATOMICROW := FALSE;
                                   NAME := SUBSTNAME(FELDCT[ROWNO].CHEM);
                                   CHARGE := FELDCT[ROWNO].CHGE;
                                   VALUES := NIL;
                                   READCONGENERS(CONGENERS, HYDROGENS, ATOMICROW, ROWNO);
                                   IF NAME IN DECLSUBS
                                      THEN WITH RDECLARATIONTABLE[NAME]^.CONNBONDS DO
                                              IF (CONNECTIONS = NUMOFCONNS(CONGENERS, HYDROGENS, ROWNO))
                                                         OR (CONNECTIONS = NOTSET)
                                                 THEN {accords with previous declaration}
                                                 ELSE REJECT(53, ROWNO)
                                      ELSE CHECKEARLIERDEFN(NAME, NUMOFCONNS(CONGENERS, HYDROGENS, ROWNO), ROWNO
                                  END
                              END
            END;
      IF REJECTED
        THEN IF INTERACTIVE
               THEN BEGIN
                     FOR ROWNO := 1 TO NUMOFNODES DO IF CT[ROWNO] <> NIL THEN
                         IF CT[ROWNO]^.ATOMICROW
                            THEN BEGIN
                                  DISPOSE(CT[ROWNO], TRUE);
                                  ECTRSIZE := ECTRSIZE - 30
                                 END
                            ELSE BEGIN
                                  DISPOSE(CT[ROWNO], FALSE);
                                  ECTRSIZE := ECTRSIZE - 33
                                 END;
                     FELDMODE := OLDDIAGRAM;
                     FELDMN(FELDMODE, FELDFIL);
                     IF FELDMODE = OLDDIAGRAM THEN FAILURE(41, 0, '    ')
                    END
```

```
                  ELSE FAILURE(41, 0, '    ')
   UNTIL NOT REJECTED;

   FOR ROWNO := 1 TO NUMOFNODES DO IF CT[ROWNO] <> NIL THEN WITH CT[ROWNO]^ DO
      IF NOT ATOMICROW THEN
         BEGIN
          GETPOSNS(CONGENERS, CONNBONDS, PRNTPOSNS);
          DECLARESUBST(CT[ROWNO]^.NAME,
                       PSADDRESS,
                       NIL,
                       CONNBONDS,
                       PRNTPOSNS);
          IF FELDCT[ROWNO].MULT <> 0 THEN DECLAREMULT(FELDCT[ROWNO].MULT, CT[ROWNO]^.NAME);
          IF INDEPENDENT(ROWNO)
             THEN WITH CT[ROWNO]^ DO ENTERCOMBIN(NAME, VALUES)
             ELSE BEGIN
                   ENTERCOMBIN(CT[ROWNO]^.NAME, PSADDRESS^.CHILDGATE);
                   DISPOSE(CT[ROWNO], FALSE);
                   ECTRSIZE := ECTRSIZE - 33
                  END
         END
   END;
                                 { OF PROCEDURE PROCESSCT
   ------------------------------------------------------------------------------}




   {-----------------------------------------------------------------------------}
   PROCEDURE READSD(VAR PSADDRESS : PTRPSTYPE;
                        INTERACTIVE   : BOOLEAN);

   { Sets up a SPECIFIC partial structure in PSADDRESS, uses SPLITLINE and
     DIVIDELINE to handle Gensal lines containing tokens after the SD, and calls
     PROCESSCT to reformat the connection table. If INTERACTIVE is TRUE then
     FELDMN and READFELDMANN are used to produce FELDCT and FELDBD. Otherwise
     they are derived by DECODECT.
     Called by ALTNVLIST\ELEMENT
               Body of INTERPRET}

```

```
VAR OLDLINE       : LINELIST;    { GENSAL source line from which READSD was called }
    LINECONTINUED : BOOLEAN;     { Indicates more GENSAL on line }



FUNCTION SPLITLINE : BOOLEAN;

{ TRUE if there is any non-space character beyond the current position (N) in
  CURRENTLINE, which is space-filled from the current position, the original
  version being saved in OLDLINE }

VAR M : 0..MAXLENGTH;

BEGIN
OLDLINE := CURRENTLINE^;
SPLITLINE := FALSE;
FOR M := N TO MAXLENGTH DO
   IF CURRENTLINE^.LINE[M] <> ' ' THEN
      BEGIN
       SPLITLINE := TRUE;
       CURRENTLINE^.LINE[M] := ' '
      END
END;



PROCEDURE DIVIDELINE(VAR CURRENTLINE : PLINELIST);

{ Places the second half of OLDLINE.LINE in a new location in the
  linked list of lines }

VAR M : 1..MAXLENGTH;

BEGIN
FOR M := 1 TO (N-1) DO OLDLINE.LINE[M] := ' ';
OLDLINE.LAST := CURRENTLINE;
OLDLINE.NEXT := CURRENTLINE^.NEXT;
NEW(CURRENTLINE^.NEXT);
CURRENTLINE^.NEXT^ := OLDLINE;
CURRENTLINE := CURRENTLINE^.NEXT
```

```
    END;



    BEGIN { body of procedure READSD  }
    NEW(PSADDRESS, SPECIFIC);
    ECTRSIZE := ECTRSIZE + 70;
    WITH PSADDRESS^ DO
       BEGIN
        PSVARIETY := SPECIFIC;
        VISITED := FALSE;
        CHILDGATE := NIL;
        PARENTGATE := NIL
       END;
    LINECONTINUED := SPLITLINE;
    IF INTERACTIVE
       THEN BEGIN
             FELDMODE := NEWDIAGRAM;
             WRITELN;
             WRITELN('FELDMANN graphics system for structure diagram input and display:');
             FELDMN(FELDMODE,FELDFIL)
            END
       ELSE BEGIN
             CURRENTLINE := CURRENTLINE^.NEXT;
             DECODECT(CURRENTLINE,TRUE);
             IF LINECONTINUED THEN DIVIDELINE(CURRENTLINE)
                              ELSE N := MAXLENGTH
            END;
    PROCESSCT(PSADDRESS^.CT, INTERACTIVE, PSADDRESS);
    IF INTERACTIVE THEN
       BEGIN
        NEW(CURRENTLINE^.NEXT);
        CURRENTLINE^.NEXT^.LAST := CURRENTLINE;
        CURRENTLINE := CURRENTLINE^.NEXT;
        CURRENTLINE^.NEXT := NIL;
        ENCODECT(CURRENTLINE);
        IF LINECONTINUED THEN DIVIDELINE(CURRENTLINE)
                         ELSE N := MAXLENGTH
       END
    END;
```

```
                              { of procedure READSD
-------------------------------------------------------------------------------}



FUNCTION CHECKDELIM (VALIDELIMS : DELIMSET) : DELIMTYPE;

BEGIN
   IF TOKEN.NATURE = DELIMITER
      THEN IF TOKEN.DELIMVAL IN VALIDELIMS
              THEN CHECKDELIM := TOKEN.DELIMVAL
              ELSE CHECKDELIM := INVALIDTOKEN
      ELSE CHECKDELIM :=INVALIDTOKEN
END;



{------------------------------------------------------------------------------

                  P R O C E D U R E   I N T E G E R R A N G E

-------------------------------------------------------------------------------}

PROCEDURE INTEGERRANGE (VAR RANGEVALUES : INTRECORD;
                        LIMITRANGE      : INTRECORD;
                        ERRORCODE       : INTEGER);

{ Carries out syntactic and semantic checking on integer ranges. LIMITRANGE is
  the range of values that all values in RANGEVALUES must fall, and is used for
  the semantic checking (functions INCREASING, WITHINLIMITS and ALLWITHINLIMITS).
  ERRORCODE is the relevant error code for passing to procedure ERROR.
  Called by GROUPRANGE
            SELECTOR}

VAR PTR : PDOUBLIST;


FUNCTION WITHINLIMITS(TESTVALUE  : INTEGER) : BOOLEAN;

{ Returns TRUE is TESTVALUE is in the range covered by LIMITRANGE
```

```
    Called by ALLWITHINLIMITS
              RANGEFRAGMENT}

  VAR PTR : PDOUBLIST;

  BEGIN
  PTR := LIMITRANGE.SUBRANGES;
  WITHINLIMITS := FALSE;
  IF (TESTVALUE >= LIMITRANGE.TOPRANGE) AND (LIMITRANGE.TOPRANGE >= 0)
     THEN WITHINLIMITS := TRUE
     ELSE WHILE PTR <> NIL DO
            IF (TESTVALUE > PTR^.SECOND)
               THEN PTR := NIL
               ELSE IF TESTVALUE < PTR^.FIRST
                      THEN PTR := PTR^.NEXT
                      ELSE BEGIN
                             WITHINLIMITS := TRUE;
                             PTR := NIL
                           END
  END;



  FUNCTION INCREASING (TESTVALUE : INTEGER ) : BOOLEAN;

  { Returns TRUE is TESTVALUE is larger than than the last integer in the range
    Called by RANGEFRAGMENT}

  BEGIN
  IF RANGEVALUES.SUBRANGES = NIL
     THEN INCREASING := TRUE  { This is the first integer in the range }
     ELSE INCREASING := TESTVALUE > RANGEVALUES.SUBRANGES^.SECOND
  END;



  FUNCTION ALLWITHINLIMITS(LOWERBOUND,
                           UPPERBOUND  : INTEGER): BOOLEAN;

  { Returns TRUE is all the values between LOWERBOUND and UPPERBOUND inclusive
```

```
      are covered by LIMITRANGE.
      Called by RANGEFRAGMENT}

    VAR VALID : BOOLEAN;

    BEGIN
    VALID := TRUE;
    WHILE (LOWERBOUND <= UPPERBOUND) AND VALID DO
       IF WITHINLIMITS(LOWERBOUND)
          THEN LOWERBOUND := LOWERBOUND + 1
          ELSE BEGIN
                 ERROR(ERRORCODE, LOWERBOUND);
                 VALID := FALSE
               END;
    ALLWITHINLIMITS := VALID
    END;



    PROCEDURE RANGEFRAGMENT;

    { Carries out syntactic/semantic checking on a single range fragment. On
      entry to the procedure TOKEN is the token immediately before the first
      integer of the fragment. On exit, TOKEN is a comma or integer range
      terminating token.
      Called by Body of INTEGERRANGE}

    VAR TERMINATORS  : DELIMSET;  { Tokens that terminate an integer range }
        VALID        : BOOLEAN;
        FIRSTINTEGER : INTEGER;   { The first integer in N1-N2 type ranges }
        DELIMCHECK   : DELIMTYPE;

    BEGIN
    TERMINATORS := [GCLOSANG, GRSQUARE, GEQUALS, GSEQ, GDEQ, GDOLEQ, GHASHEQ];
    NEXTTOKEN;
    REPEAT
     VALID := FALSE;
     WHILE TOKEN.NATURE <> INTEGRAL DO ERROR(23,0);
     IF NOT INCREASING(TOKEN.INTEGVAL)
        THEN ERROR(27,0)
```

```
                ELSE IF WITHINLIMITS(TOKEN.INTEGVAL)
                        THEN VALID := TRUE
                        ELSE ERROR(ERRORCODE,TOKEN.INTEGVAL);
            UNTIL VALID;
            FIRSTINTEGER := TOKEN.INTEGVAL;
            NEXTTOKEN;
            REPEAT
             VALID := FALSE;
             DELIMCHECK := CHECKDELIM([GCOMMA,GHYPHEN]+TERMINATORS);
             IF DELIMCHECK=INVALIDTOKEN THEN ERROR(24,0)
                                        ELSE VALID := TRUE
            UNTIL VALID;
            IF DELIMCHECK <> GHYPHEN
               THEN ADDINTS(RANGEVALUES.SUBRANGES, FIRSTINTEGER, FIRSTINTEGER)
               ELSE BEGIN
                     NEXTTOKEN;
                     REPEAT
                      VALID := FALSE;
                      WHILE (TOKEN.NATURE <> INTEGRAL) AND (CHECKDELIM(TERMINATORS) = INVALIDTOKEN)
                         DO ERROR(24,0);
                      IF TOKEN.NATURE = INTEGRAL
                         THEN IF TOKEN.INTEGVAL < FIRSTINTEGER
                                 THEN ERROR(27,0)
                                 ELSE IF ALLWITHINLIMITS(FIRSTINTEGER, TOKEN.INTEGVAL)
                                         THEN BEGIN
                                               VALID := TRUE;
                                               ADDINTS(RANGEVALUES.SUBRANGES, FIRSTINTEGER, TOKEN.INTEGVAL);
                                               NEXTTOKEN;
                                               WHILE CHECKDELIM([GCOMMA] + TERMINATORS)= INVALIDTOKEN DO ERROR(24, 0)
                                              END
                                         ELSE
                         ELSE BEGIN
                               IF LIMITRANGE.TOPRANGE = NOTSET
                                  THEN IF LIMITRANGE.SUBRANGES=NIL
                                          THEN ERROR(ERRORCODE,0)
                                          ELSE ERROR(ERRORCODE, LIMITRANGE.SUBRANGES^.SECOND + 1)
                                  ELSE VALID := ALLWITHINLIMITS(FIRSTINTEGER, LIMITRANGE.TOPRANGE);
                               IF VALID THEN RANGEVALUES.TOPRANGE := FIRSTINTEGER
                              END
                     UNTIL VALID
```

```
                END
           END;


           BEGIN  { Body of INTEGERRANGE }
           RANGEVALUES.SUBRANGES := NIL;
           RANGEVALUES.TOPRANGE := NOTSET;
           REPEAT RANGEFRAGMENT
           UNTIL TOKEN.DELIMVAL <> GCOMMA;
           PTR := RANGEVALUES.SUBRANGES;
           WHILE PTR <> NIL DO
              BEGIN
               ECTRSIZE := ECTRSIZE + 6;
               PTR := PTR^.NEXT
              END
           END;
                         { of Procedure INTEGERRANGE

           ------------------------------------------------------------------------------}




           PROCEDURE SETINTS (VAR RANGE  : INTRECORD;
                                  ONESET : INTEGSET);

           { Takes a set of integers, and converts them to integer range format. If
             MAXVARS is a member of the set, then TOPRANGE is set to the  member
             of the set above the highest absent member.
             Called from GROUPRANGE
                         SELECTOR}

           VAR N : NOTSET..MAXVARS;

           BEGIN
           WITH RANGE DO
              BEGIN
               IF MAXVARS IN ONESET
                  THEN BEGIN
                        N := MAXVARS;
```

```
                WHILE N IN ONESET DO N := N-1;
                TOPRANGE := N + 1;
                ONESET := ONESET - [TOPRANGE..MAXVARS]
               END
         ELSE TOPRANGE := NOTSET;
      SUBRANGES := NIL;
      FOR N := 0 TO MAXVARS DO
         IF N IN ONESET THEN
            ADDINTS(RANGE.SUBRANGES, N,N);
     END
   END;



   PROCEDURE INTSET(VAR ONESET : INTEGSET;
                        RANGE      : INTRECORD);

   { Converts an integer range into a set of integers, and DESTROYs the SUBRANGES
     of the integer range.
     Called from ALTNVLIST\ELEMENT\PARAMETERLIST}

   VAR PTR : PDOUBLIST;
       M   : INTEGER;

   BEGIN
   WITH RANGE DO
      BEGIN
       IF TOPRANGE=NOTSET
          THEN ONESET := []
          ELSE ONESET := [TOPRANGE..MAXVARS];
       PTR := SUBRANGES;
       WHILE PTR <> NIL DO WITH PTR^ DO
          BEGIN
           FOR M := FIRST TO SECOND DO ONESET := ONESET + [M];
           PTR := NEXT
          END;
       REDUCEECTR(SUBRANGES);
       DESTROY(SUBRANGES)
      END
   END;
```

```
PROCEDURE GROUPRANGE (VAR MEMBERS : INTEGSET;
                          LIMITSET    : INTEGSET;
                          ERRORCODE   : INTEGER);

{ Converts LIMITSET into a INTRECORD format, and uses this as the limitrange
  for a call to INTEGERRANGE. The RANGE that this returns is converted back
  to a set (MEMBERS).
  Called by ASIGNMENTSTMNT\SUBSTGROUP
            ASSIGNMENTSTMNT\MULTASSIGNMENT
            ALTNVLIST\POSITIONSET}

VAR RANGE,
    LIMITRANGE  : INTRECORD;
    PTR         : PDOUBLIST;
    VAL         : 0..MAXVARS;

BEGIN
MEMBERS := [];
SETINTS(LIMITRANGE,LIMITSET);
INTEGERRANGE(RANGE,LIMITRANGE,ERRORCODE);
PTR := RANGE.SUBRANGES;
WHILE PTR <> NIL DO WITH PTR^ DO
  BEGIN
   FOR VAL := FIRST TO SECOND DO
     MEMBERS := MEMBERS + [VAL];
   PTR := NEXT
  END;
REDUCEECTR(LIMITRANGE.SUBRANGES);
DESTROY(LIMITRANGE.SUBRANGES);
DESTROY(RANGE.SUBRANGES)
END;



PROCEDURE CHECKVALIDINT (LIMITSET  : INTEGSET;
                            ERRORCODE : INTEGER );
```

```
{ Checks that the current TOKEN is an integer within LIMITSET, and obtains
  further tokens from the input stream if it is not.
  Called by ASSIGNMENTSTMNT\SUBSTGROUP\SUBSTCOMBINATION
            ALTNVLIST\POSITIONSET\POSNCOMBINATION
            ALTNVLIST\POSITIONSET}

VAR VALID : BOOLEAN;

BEGIN
VALID := FALSE;
REPEAT
 WHILE TOKEN.NATURE <> INTEGRAL DO ERROR(23,0);
 IF TOKEN.INTEGVAL > MAXVARS
    THEN ERROR(ERRORCODE, TOKEN.INTEGVAL)
    ELSE IF TOKEN.INTEGVAL IN LIMITSET
            THEN VALID := TRUE
            ELSE ERROR(ERRORCODE, TOKEN.INTEGVAL)
UNTIL VALID
END;



PROCEDURE SELECTOR(VAR VALUERANGE : INTRECORD;
                       LIMITSET   : INTEGSET;
                       ERRORCODE  : INTEGER);

{ Analyses a Gensal selector, returning the values in VALUERANGE. Limited by
  LIMITSET. ERRORCODE is passed to INTEGERRANGE.
  Called from ALTNVLIST\ELEMENT\PARAMETERLIST\USERPARAMETER
              ALTNVLIST\ELEMENT\PARAMETERLIST
              ALTNVLIST\ELEMENT
              ASSIGNMENTSTMNT
              ASSIGNMENTSTMNT\MULTASSIGNMENT}

VAR LIMITRANGE : INTRECORD;

BEGIN
WHILE CHECKDELIM([GOPENANG])=INVALIDTOKEN DO ERROR(21,0);
SETINTS(LIMITRANGE, LIMITSET);
INTEGERRANGE(VALUERANGE, LIMITRANGE, ERRORCODE);
```

```
DESTROY (LIMITRANGE.SUBRANGES);
WHILE CHECKDELIM([GCLOSANG])=INVALIDTOKEN DO ERROR(22,0)
END;



{***************************************************************************

                  P R O C E D U R E   A L T N V L I S T

***************************************************************************}

PROCEDURE ALTNVLIST(PARENTPSLIST       : PPSLIST;
                    OPTIONALSUB        : BOOLEAN);

{ Processes alternatives separated by / delimiters.
  Called by ASSIGNMENTSTMNT\SUBSTASSIGNMENT
            ALTNVLIST\ELEMENT (recursively)
            ALTNVLIST\ELEMENT\TRANSLATENOMEN (recursively) }

TYPE PPALTBARS = ^PALTBARS;
     PALTBARS  = RECORD
                   PARSTRUCT   : PTRPSTYPE;
                   ALTBAR      : PALTERNLIST;
                   CONNBONDS   : TCONNBONDS;
                   PRNTPOSNS   : PTGROUPMEMS;
                   COPYCHILDPS : BOOLEAN;
                   NEXT        : PPALTBARS
                 END;

VAR PARALTLIST,
    WRITEPTR        : PPALTBARS;
    NEWALTERNATIVE  : PALTERNLIST;
    READPTR         : PPSLIST;



PROCEDURE UPDATEPARALTCONNS(PARALTLIST : PPALTBARS);

{ Copies the CONNBONDS field of the first item in PARALTLIST into all the
```

```
  other items in the list.
  Called by ELEMENT\VALIDSUBST
            ELEMENT}

VAR NEWCONNBONDS : TCONNBONDS;

BEGIN
NEWCONNBONDS := PARALTLIST^.CONNBONDS;
REPEAT
 PARALTLIST^.CONNBONDS := NEWCONNBONDS;
 PARALTLIST := PARALTLIST^.NEXT
UNTIL PARALTLIST = NIL
END;



{--------------------------------------------------------------------------------}
PROCEDURE POSITIONSET(VAR SETMEMS          : TGROUPMEMS;
                      AVAILABLEPOSITIONS : TGROUPMEMS;
                      CONNECTIVITY       : TCONNS;
                      ERRORCODE          : INTEGER);

{ Analyses a position set.
  Called from MODIFYCHILDPOSITIONS
            ELEMENT}

VAR AVAILFIRST : INTEGSET;



PROCEDURE FINDFIRST(VAR POSNSET : INTEGSET;
                    POSNLIST    : PDOUBLIST);

{ Returns a set consisting of the FIRST fields of all the items in POSNLIST. }

BEGIN
POSNSET := [];
WHILE POSNLIST <> NIL DO WITH POSNLIST^ DO
   BEGIN
    POSNSET := POSNSET + [FIRST];
```

```
    POSNLIST := NEXT
   END
END;



PROCEDURE FINDSECOND(VAR POSNSET : INTEGSET;
                     POSNLIST    : PDOUBLIST;
                     FIRSTPOSN   : ATOMNUMBER);

{ Returns a set consisting of the SECOND fields of the items in POSNLIST
  that have FIRSTPOSN as FIRST field. }

BEGIN
POSNSET := [];
WHILE POSNLIST <> NIL DO WITH POSNLIST^ DO
   BEGIN
    IF FIRST = FIRSTPOSN THEN POSNSET := POSNSET + [SECOND];
    POSNLIST := NEXT
   END
END;



PROCEDURE POSNCOMBINATION(AVAILPOSNS   : INTEGSET;
                          VAR COMBMEMS : PDOUBLIST);

{ Analyses a position combination, checking the validity of each position,
  and inserting it into the front of the list headed by COMBMEMS. }

VAR POSNPAIR : PDOUBLIST;

BEGIN
NEW(POSNPAIR);
ECTRSIZE := ECTRSIZE + 6;
NEXTTOKEN;
CHECKVALIDINT(AVAILPOSNS, ERRORCODE);
POSNPAIR^.FIRST := TOKEN.INTEGVAL;
NEXTTOKEN;
WHILE CHECKDELIM([GSLASH]) <> GSLASH DO ERROR(33,0);
```

```
      IF AVAILABLEPOSITIONS.COMBINED
         THEN FINDSECOND(AVAILPOSNS, AVAILABLEPOSITIONS.COMBMEMS, POSNPAIR^.FIRST)
         ELSE {leave AVAILPOSNS the same};
      NEXTTOKEN;
      CHECKVALIDINT(AVAILPOSNS, ERRORCODE);
      POSNPAIR^.SECOND := TOKEN.INTEGVAL;
      POSNPAIR^.NEXT := COMBMEMS;
      COMBMEMS := POSNPAIR;
      NEXTTOKEN;
      WHILE CHECKDELIM([GCOMMA, GRSQUARE]) = INVALIDTOKEN DO ERROR(24,0)
      END;



      BEGIN     { Body of Procedure POSITIONSET }
      IF AVAILABLEPOSITIONS.COMBINED
         THEN FINDFIRST(AVAILFIRST, AVAILABLEPOSITIONS.COMBMEMS)
         ELSE AVAILFIRST := AVAILABLEPOSITIONS.MEMBERS;
      LOOKAHEAD;
      CHECKVALIDINT(AVAILFIRST, ERRORCODE);
      LOOKAHEAD;
      CASE CONNECTIVITY OF
       NOTSET : WHILE CHECKDELIM([GSLASH, GCOMMA, GHYPHEN, GRSQUARE]) = INVALIDTOKEN DO ERROR(24,0);
       0      : PROGERROR(10); {attempting to process position set for unconnected substituent}
       1      : WHILE CHECKDELIM([GCOMMA, GHYPHEN, GRSQUARE]) = INVALIDTOKEN DO
                    IF CHECKDELIM([GSLASH])=GSLASH THEN ERROR(34,0)
                                                   ELSE ERROR(24,0);
       2      : WHILE CHECKDELIM([GSLASH]) <> GSLASH DO ERROR(33,0)
       END;
      IF TOKEN.DELIMVAL = GSLASH
         THEN BEGIN
                SETMEMS.COMBINED := TRUE;
                SETMEMS.COMBMEMS := NIL;
                NEXTTOKEN;
                REPEAT POSNCOMBINATION(AVAILFIRST, SETMEMS.COMBMEMS)
                UNTIL CHECKDELIM([GRSQUARE]) = GRSQUARE
              END
         ELSE BEGIN
                NEXTTOKEN;
                SETMEMS.COMBINED := FALSE;
```

```
            GROUPRANGE(SETMEMS.MEMBERS, AVAILFIRST, ERRORCODE)
           END;
   IF AVAILABLEPOSITIONS.COMBINED
      THEN BEGIN
            REDUCEECTR(AVAILABLEPOSITIONS.COMBMEMS);
            DESTROY(AVAILABLEPOSITIONS.COMBMEMS)
           END
   END;
           {  of Procedure POSITIONSET
   ------------------------------------------------------------------------------}



   FUNCTION COPYLIST(COMBMEMS : PDOUBLIST) : PDOUBLIST;

   { Makes a reversed copy of COMBMEMS
     Called by ALTNVLIST\MODIFYCHILDPOSITIONS\TRACEDOWNGATE
               ALTNVLIST\ELEMENT\SETCOMBARS\CHECKCOMBPOSNS
               ALTNVLIST\ELEMENT\GETLIMITPOSITIONS}

   VAR NEWLIST,
       NEWITEM : PDOUBLIST;

   BEGIN
   NEWLIST := NIL;
   WHILE COMBMEMS <> NIL DO WITH COMBMEMS^ DO
      BEGIN
       NEW(NEWITEM);
       ECTRSIZE := ECTRSIZE + 6;
       NEWITEM^.FIRST := FIRST;
       NEWITEM^.SECOND := SECOND;
       NEWITEM^.NEXT := NEWLIST;
       NEWLIST := NEWITEM;
       COMBMEMS := NEXT
      END;
   COPYLIST := NEWLIST
   END;
```

```
PROCEDURE REDUCE(VAR LIMITLIST : PDOUBLIST;
                     COMPSET   : TGROUPMEMS);

{ Removes those items in LIMITLIST that do not appear in COMPSET
  Called by ALTNVLIST\MODIFYCHILDPOSITIONS\TRACEDOWNGATE
            ALTNVLIST\ELEMENT\TRANSLATENOMEN\MODIFYGATEPOSITIONS
            ALTNVLIST\ELEMENT\GETLIMITPOSITIONS}

VAR LISTPTR,
    LASTPTR,
    COMPPTR : PDOUBLIST;
    FOUND   : BOOLEAN;

BEGIN
LISTPTR := LIMITLIST;
LASTPTR := NIL;
WHILE LISTPTR <> NIL DO
   BEGIN
    IF COMPSET.COMBINED
       THEN BEGIN
              FOUND := FALSE;
              COMPPTR := COMPSET.COMBMEMS;
              WHILE (COMPPTR <> NIL) AND NOT FOUND DO
                 BEGIN
                  FOUND := (COMPPTR^.FIRST=LISTPTR^.FIRST) AND (COMPPTR^.SECOND=LISTPTR^.SECOND);
                  COMPPTR := COMPPTR^.NEXT
                 END
             END
       ELSE FOUND := [LISTPTR^.FIRST, LISTPTR^.SECOND] <= COMPSET.MEMBERS;

    IF FOUND THEN BEGIN
                    LASTPTR := LISTPTR;
                    LISTPTR := LASTPTR^.NEXT
                   END
             ELSE IF LASTPTR = NIL
                     THEN BEGIN
                             LIMITLIST := LISTPTR^.NEXT;
                             DISPOSE(LISTPTR);
                             ECTRSIZE := ECTRSIZE - 6;
                             LISTPTR := LIMITLIST
```

```
                                END
                            ELSE BEGIN
                                 LASTPTR^.NEXT := LISTPTR^.NEXT;
                                 DISPOSE(LISTPTR);
                                 ECTRSIZE := ECTRSIZE - 6;
                                 LISTPTR := LASTPTR^.NEXT
                                END
       END
    END;



    PROCEDURE CONCATENATETERMS(VAR GATEPS : PTRPSTYPE);

    { Sets up a PS of variety OTHER, and concatenates NOMENCLATURE tokens up to a
      maximum of TERMLENGTH chars into it.
      Called from ELEMENT}

    VAR DELIMCHECK : DELIMTYPE;
        TERMEND    : BOOLEAN;
        M, M2      : 0..TERMLENGTH;

    BEGIN
    NEW(GATEPS, OTHER);
    ECTRSIZE := ECTRSIZE + 22;
    WITH GATEPS^ DO
       BEGIN
        PSVARIETY := OTHER;
        VISITED := FALSE;
        CHILDGATE := NIL;
        PARENTGATE := NIL;
        FOR M := 1 TO TERMLENGTH DO TERM[M] := ' '
       END;
    M := 0;
    NEXTTOKEN;
    REPEAT
     DELIMCHECK := CHECKDELIM([GPRIME]);
     IF DELIMCHECK=INVALIDTOKEN
        THEN IF TOKEN.NATURE <> NOMENCLATURE
                THEN ERROR(25,0)
```

```
                ELSE BEGIN
                      TERMEND := M = TERMLENGTH;
                      M2 := 0;
                      WHILE NOT TERMEND DO
                         BEGIN
                          M  := M  + 1;
                          M2 := M2 + 1;
                          GATEPS^.TERM[M] := TOKEN.NOMENVAL[M2];
                          TERMEND := (M=TERMLENGTH) OR (TOKEN.NOMENVAL[M2]=' ')
                         END;
                      NEXTTOKEN
                     END
    UNTIL DELIMCHECK=GPRIME
    END;



    FUNCTION RECORDHELD(TERM        : STRING32;
                        VAR ADDRESS : INTEGER) : BOOLEAN;

    { Determines whether or not a record is held for TERM. Requests synonyms
      for the term (using TERMREAD) if initially unsuccessful.
      The search is abandoned if a record is found, or if TERMREAD returns
      FALSE.
      Called by ELEMENT\TRANSLATENOMEN}

    VAR STILLLOOKING,
        SYNONYMREAD   : BOOLEAN;

    BEGIN
    SYNONYMREAD := FALSE;
    REPEAT
     STILLLOOKING := NORECORD(TERM, ADDRESS);
     RECORDHELD := NOT STILLLOOKING;
     IF STILLLOOKING
        THEN BEGIN
              WRITE('No record held for "');
              PRINTNOM(TERM);
              WRITELN('".');
              STILLLOOKING := FALSE;
```

```
                CASE INPUTMODE OF
                 TERMINAL     : BEGIN
                                 WRITE('Enter synonym or <CR>:  > ');
                                 STILLLOOKING := TERMREAD(TERM);
                                 SYNONYMREAD := STILLLOOKING
                                END;
                 STOREDGENSAL : ;
                 INSERTTEXT   : WRITELN('(Term in inserted Gensal expression)')
                END
               END
          ELSE IF SYNONYMREAD
                 THEN BEGIN
                       WRITE('Record found for "');
                       PRINTNOM(TERM);
                       WRITELN('".');
                      END
      UNTIL NOT STILLLOOKING
      END;



      FUNCTION DEFNTABLEENTRY(PARSTRUCT : PTRPSTYPE) : BOOLEAN;

      { Returns TRUE if PARSTRUCT is NIL or has no PARENTGATE (provided it is not
        INTERNALREP.CONSTANTPART). Since the parameter passed is the PARSTRUCT
        field of a PARALTLIST element, this indicates whether or not it is in the
        chain of structures pointed at by RDEFINITIONTABLE.
        Called by ELEMENT\SETCOMBARS
                  ELEMENT\SUBSTASVALUE
                  ADDFURTHERSUBTN}

      BEGIN
      IF PARSTRUCT = NIL
         THEN DEFNTABLEENTRY := TRUE
         ELSE IF PARSTRUCT^.PARENTGATE = NIL
                 THEN DEFNTABLEENTRY := PARSTRUCT <> INTERNALREP.CONSTANTPART
                 ELSE DEFNTABLEENTRY := FALSE
      END;
```

```
PROCEDURE FINDPOSITIONS(PTRPS          : PTRPSTYPE;
                        VAR AVAILPOSNS : INTEGSET;
                        BONDMAG        : TBONDMAG);

{ Returns the positions in PTRPS^ which are substitutable by a bond of
  magnitude BONDMAG
  Called by GETCHILDPOSITIONS
            MODIFYCHILDPOSITIONS\GETCOMBPOSNS}

VAR ROWNO   : ATOMNUMBER;

BEGIN
WITH PTRPS^ DO CASE PSVARIETY OF
 DUMMY,
 UNKNOWN,
 OTHER    : AVAILPOSNS := [1..MAXCT];
 GENERIC  : AVAILPOSNS := [1];
 SPECIFIC : BEGIN
             AVAILPOSNS := [];
             FOR ROWNO := 1 TO MAXCT DO IF CT[ROWNO] <> NIL
                 THEN IF CT[ROWNO]^.HYDROGENS >= BONDMAG
                        THEN AVAILPOSNS := AVAILPOSNS + [ROWNO]
            END
 END
END;



{...........................................................................}
PROCEDURE GETCHILDPOSITIONS(PTRPS              : PTRPSTYPE;
                            VAR CONNBONDS      : TCONNBONDS;
                            VAR CHILDPOSITIONS : TGROUPMEMS);

{ Makes initial determination of CHILDPOSITIONS for PTRPS (which points to a
  Child PS), also modifying CONNBONDS as necessary. The CHILDPOSITIONS field
  may be further modified by a post substituent value position set.
  Called by ELEMENT\SETCOMBARS
            ELEMENT\SUBSTASVALUE}

```

```
VAR POSNA,
    POSNB       : ATOMNUMBER;
    POSNSETA,
    POSNSETB,
    POSNSETC    : INTEGSET;
    MAGSUM      : INTEGER;
    NUMMARKERS  : TCONNS;
    BONDA,
    BONDB       : BONDORDER;
    FAILSTRING  : STRING4;



FUNCTION BONDCHECK(PARENTBOND,
                   CHILDBOND  : BONDORDER) : BONDORDER;

{ Checks compatability of the two bonds, ejecting user to the editor if the
  bonds are found to be incompatible. The global variable BONDMATCHARRAY is
  used to check the compatibility.
  Called from THISWAYROUND
              Body of GETCHILDPOSITIONS}

VAR NEWBOND   : BONDORDER;
    FAILDATA : STRING4;

BEGIN
NEWBOND := BONDMATCHARRAY[PARENTBOND, CHILDBOND];
IF NEWBOND = NOTSPECIFIED
   THEN BEGIN
          FAILDATA[1] := BONDSTRING[PARENTBOND,1];
          FAILDATA[2] := BONDSTRING[PARENTBOND,2];
          FAILDATA[3] := BONDSTRING[CHILDBOND, 1];
          FAILDATA[4] := BONDSTRING[CHILDBOND, 2];
          FAILURE(42, 0, FAILDATA)
        END
   ELSE BONDCHECK := NEWBOND
END;
```

```
PROCEDURE GETMARKEDPOSNS(CT        : CTTYPE;
                          VAR POSNA,
                              POSNB : ATOMNUMBER;
                          VAR BONDA,
                              BONDB : BONDORDER);

{ Returns those positions in CT which have PARENTAL bonds, with their bond
  orders. On entry the parameters are NPTFIXED or NOTSPECIFIED. PROCESSCT
  will only have permitted a maximum of two marked positions.}

VAR ROWNO : ATOMNUMBER;
    CNGNR : 1..MAXCONGENERS;

BEGIN
FOR ROWNO := 1 TO MAXCT DO IF CT[ROWNO] <> NIL
    THEN FOR CNGNR := 1 TO MAXCONGENERS DO
             WITH CT[ROWNO]^.CONGENERS[CNGNR] DO
                IF RELATIONSHIP = PARENTAL
                   THEN IF POSNA = NOTFIXED
                           THEN BEGIN
                                  POSNA := ROWNO;
                                  BONDA := BOND
                                END
                           ELSE BEGIN
                                  POSNB := ROWNO;
                                  BONDB := BOND
                                END
END;



FUNCTION HYDROGENPS(PTRPS : PTRPSTYPE) : BOOLEAN;

{ Returns TRUE if PTRPS represents hydrogen (i.e. has no non-hydrogen atoms).}

VAR ROWNO : ATOMNUMBER;

BEGIN
IF PTRPS^.PSVARIETY = SPECIFIC
   THEN BEGIN
```

```
            HYDROGENPS := TRUE;
            FOR ROWNO := 1 TO MAXCT DO
               IF PTRPS^.CT[ROWNO] <> NIL THEN HYDROGENPS := FALSE
           END
      ELSE HYDROGENPS := FALSE
   END;



   FUNCTION THISWAYROUND(PARENTA,
                         CHILDA,
                         PARENTB,
                         CHILDB  : BONDORDER) : BOOLEAN;

   { Determines whether or not the bonds in a doubly-connected child need to be
     reversed for compatability with the parent. If either way will do, the way
     given is prefered unless the other way round matches identical (as opposed
     to merely compatible) bonds. }

   VAR FITSTHISWAY,
       FITSOTHERWAY : BOOLEAN;

   BEGIN
   FITSTHISWAY := (BONDMATCHARRAY[PARENTA,CHILDA] <> NOTSPECIFIED) AND (BONDMATCHARRAY[PARENTB,CHILDB] <> NOT
   PECIFIED);
   FITSOTHERWAY := (BONDMATCHARRAY[PARENTA,CHILDB] <> NOTSPECIFIED) AND (BONDMATCHARRAY[PARENTB,CHILDA] <> NO
   SPECIFIED);
   IF FITSTHISWAY
      THEN IF FITSOTHERWAY
              THEN THISWAYROUND := NOT ((PARENTA = CHILDB) AND (PARENTB = CHILDA))
              ELSE THISWAYROUND := TRUE
      ELSE IF FITSOTHERWAY
              THEN THISWAYROUND := FALSE
              ELSE {Bond match failure - use BONDCHECK to give error message}
                   IF BONDMATCHARRAY[PARENTA, CHILDA] = NOTSPECIFIED
                      THEN PARENTA := BONDCHECK(PARENTA, CHILDA)
                      ELSE PARENTB := BONDCHECK(PARENTB, CHILDB)
   END;
```

```
PROCEDURE FINDNONAPICPOSNS(CPARAM      : INTRECORD;
                           VAR POSNSET : INTEGSET);

{ Returns a position set containing the possible "right-hand end" terminal
  positions in a GENERIC PS, based on the possible values for the ATOMCOUNT
  parameter, passed as CPARAM. }

VAR PTR : PDOUBLIST;

BEGIN
POSNSET := [];
PTR := CPARAM.SUBRANGES;
WHILE PTR <> NIL DO WITH PTR^ DO
   BEGIN
    IF SECOND <= MAXCT
       THEN POSNSET := POSNSET + [FIRST..SECOND]
       ELSE IF FIRST <= MAXCT
               THEN POSNSET := POSNSET + [FIRST..MAXCT];
    PTR := NEXT
   END;
IF CPARAM.TOPRANGE <> NOTSET
   THEN POSNSET := POSNSET + [CPARAM.TOPRANGE..MAXCT]
END;



BEGIN {Body of GETCHILDPOSITIONS}
POSNA := NOTFIXED;
POSNB := NOTFIXED;
BONDA := NOTSPECIFIED;
BONDB := NOTSPECIFIED;
IF PTRPS^.PSVARIETY = SPECIFIC
   THEN BEGIN
          GETMARKEDPOSNS(PTRPS^.CT, POSNA, POSNB, BONDA, BONDB);
          NUMMARKERS := ORD(POSNA<>NOTFIXED) + ORD(POSNB<>NOTFIXED)
        END
   ELSE NUMMARKERS := 0;
CASE CONNBONDS.CONNECTIONS OF
 NOTSET : CASE NUMMARKERS OF
```

```
                2 : BEGIN
                     CONNBONDS.CONNECTIONS := 2;
                     CONNBONDS.BONDA := BONDCHECK(CONNBONDS.BONDA, NOTSPECIFIED);
                     CONNBONDS.BONDB := BONDCHECK(CONNBONDS.BONDB, NOTSPECIFIED);
                     CHILDPOSITIONS.COMBINED := TRUE;
                     CHILDPOSITIONS.COMBMEMS := NIL;
                     LISTPOSNS(CHILDPOSITIONS.COMBMEMS, [POSNA], [POSNB], [POSNA, POSNB]);
                     IF (BONDA=BONDB) OR (BONDMATCHARRAY[BONDA,BONDB] IN [ANY, CHAIN, RING])
                        THEN LISTPOSNS(CHILDPOSITIONS.COMBMEMS, [POSNB], [POSNA], [])
                    END;
                1 : BEGIN
                     CONNBONDS.CONNECTIONS := 1;
                     CONNBONDS.BOND := BONDA;
                     CHILDPOSITIONS.COMBINED := FALSE;
                     CHILDPOSITIONS.MEMBERS := [POSNA]
                    END;
                0 : BEGIN
                     CONNBONDS.CONNECTIONS := 1; {assumption}
                     CONNBONDS.BOND := CHAISING; {assumption}
                     CHILDPOSITIONS.COMBINED := FALSE;
                     FINDPOSITIONS(PTRPS, CHILDPOSITIONS.MEMBERS, 1)
                    END
                END;
      0     : BEGIN
                CHILDPOSITIONS.COMBINED := FALSE;
                CHILDPOSITIONS.MEMBERS := []
               END;
      1     : BEGIN
                CHILDPOSITIONS.COMBINED := FALSE;
                CASE NUMMARKERS OF
                 2 : FAILURE(44, 0, '    ');
                 1 : BEGIN
                       CHILDPOSITIONS.MEMBERS := [POSNA];
                       CONNBONDS.BOND := BONDCHECK(CONNBONDS.BOND, BONDA);
                      END;
                 0 : BEGIN
                       FINDPOSITIONS(PTRPS, CHILDPOSITIONS.MEMBERS, MAGNITUDE(CONNBONDS.BOND));
                       CONNBONDS.BOND := BONDCHECK(CONNBONDS.BOND, NOTSPECIFIED)
                      END
                END
```

```
          END;
  2     : BEGIN
           CHILDPOSITIONS.COMBINED := TRUE;
           CHILDPOSITIONS.COMBMEMS := NIL;
           CASE NUMMARKERS OF
            0 : BEGIN
                  FINDPOSITIONS(PTRPS, POSNSETA, MAGNITUDE(CONNBONDS.BONDA));
                  IF PTRPS^.PSVARIETY = GENERIC
                     THEN FINDNONAPICPOSNS(PTRPS^.PARAMLIST[ATOMCOUNT], POSNSETB)
                     ELSE FINDPOSITIONS(PTRPS, POSNSETB, MAGNITUDE(CONNBONDS.BONDB));
                  WITH CONNBONDS DO MAGSUM := MAGNITUDE(BONDA)+MAGNITUDE(BONDB);
                  IF (MAGSUM <= 3) AND (POSNSETA * POSNSETB <> [])
                     THEN FINDPOSITIONS(PTRPS, POSNSETC, MAGSUM)
                     ELSE POSNSETC := [];
                  POSNSETC := POSNSETA * POSNSETB * POSNSETC;
                  LISTPOSNS(CHILDPOSITIONS.COMBMEMS, POSNSETA, POSNSETB, POSNSETC);
                  CONNBONDS.BONDA := BONDCHECK(CONNBONDS.BONDA, BONDA);
                  CONNBONDS.BONDB := BONDCHECK(CONNBONDS.BONDB, BONDB)
                 END;
            1 : BEGIN
                  FINDPOSITIONS(PTRPS, POSNSETB, MAGNITUDE(CONNBONDS.BONDB));
                  POSNSETA := [POSNA];
                  WITH CONNBONDS DO MAGSUM := MAGNITUDE(BONDA)+MAGNITUDE(BONDB);
                  IF (MAGSUM <= 3) AND (POSNSETA * POSNSETB <> [])
                     THEN FINDPOSITIONS(PTRPS, POSNSETC, MAGSUM)
                     ELSE POSNSETC := [];
                  POSNSETC := POSNSETA * POSNSETB * POSNSETC;
                  IF THISWAYROUND(CONNBONDS.BONDA, BONDA, CONNBONDS.BONDB, BONDB)
                       THEN BEGIN
                              LISTPOSNS(CHILDPOSITIONS.COMBMEMS, POSNSETA, POSNSETB, POSNSETC);
                              IF BONDMATCHARRAY[CONNBONDS.BONDB, BONDA] <> NOTSPECIFIED
                                 THEN LISTPOSNS(CHILDPOSITIONS.COMBMEMS, POSNSETB, POSNSETA, []);
                              CONNBONDS.BONDA := BONDCHECK(CONNBONDS.BONDA, BONDA);
                              CONNBONDS.BONDB := BONDCHECK(CONNBONDS.BONDB, BONDB)
                            END
                       ELSE BEGIN
                              LISTPOSNS(CHILDPOSITIONS.COMBMEMS, POSNSETB, POSNSETA, POSNSETC);
                              CONNBONDS.BONDA := BONDCHECK(CONNBONDS.BONDA, BONDB);

                              CONNBONDS.BONDB := BONDCHECK(CONNBONDS.BONDB, BONDA)
```

```
                          END
                     END;
                  2 : IF THISWAYROUND(CONNBONDS.BONDA, BONDA, CONNBONDS.BONDB, BONDB)
                        THEN BEGIN
                               LISTPOSNS(CHILDPOSITIONS.COMBMEMS, [POSNA], [POSNB], [POSNA, POSNB]);
                               IF (BONDA=BONDB) OR (BONDMATCHARRAY[BONDA,BONDB] IN [ANY, CHAIN, RING])
                                  THEN BEGIN
                                         CONNBONDS.BONDA := BONDCHECK(CONNBONDS.BONDA, BONDCHECK(BONDA,BONDB));
                                         CONNBONDS.BONDB := BONDCHECK(CONNBONDS.BONDB, BONDCHECK(BONDA,BONDB));
                                         LISTPOSNS(CHILDPOSITIONS.COMBMEMS, [POSNB], [POSNA], [])
                                       END
                                  ELSE BEGIN
                                         CONNBONDS.BONDA := BONDCHECK(CONNBONDS.BONDA, BONDA);
                                         CONNBONDS.BONDB := BONDCHECK(CONNBONDS.BONDB, BONDB)
                                       END
                             END
                        ELSE BEGIN
                               LISTPOSNS(CHILDPOSITIONS.COMBMEMS, [POSNB], [POSNA], [POSNA,POSNB]);
                               CONNBONDS.BONDA := BONDCHECK(CONNBONDS.BONDA, BONDB);
                               CONNBONDS.BONDB := BONDCHECK(CONNBONDS.BONDB, BONDA)
                             END
                  END {CASE}
                END
     END; {CASE}
    IF CHILDPOSITIONS.COMBINED
       THEN IF CHILDPOSITIONS.COMBMEMS = NIL
              THEN BEGIN
                     FAILSTRING[1] := BONDSTRING[CONNBONDS.BONDA, 1];
                     FAILSTRING[2] := BONDSTRING[CONNBONDS.BONDA, 2];
                     FAILSTRING[3] := BONDSTRING[CONNBONDS.BONDB, 1];
                     FAILSTRING[4] := BONDSTRING[CONNBONDS.BONDB, 2];
                     FAILURE(52, 0, FAILSTRING)
                   END
              ELSE IF (CHILDPOSITIONS.MEMBERS = []) AND NOT HYDROGENPS(PTRPS)
                     THEN BEGIN
                            FAILSTRING[1] := BONDSTRING[CONNBONDS.BOND, 1];
                            FAILSTRING[2] := BONDSTRING[CONNBONDS.BOND, 2];
                            FAILSTRING[3] := ' ';
                            FAILSTRING[4] := ' ';
                            FAILURE(43, 0, FAILSTRING)
```

```
                                  END
      END;              {of GETCHILDPOSITIONS
      ...........................................................................}




      {---------------------------------------------------------------------------}
      PROCEDURE MODIFYCHILDPOSITIONS(PARALTLIST : PPALTBARS);

      { This procedure modifies the CHILDPOSITIONS fields of the bottom bars of the
        childgates, and also those of the parentgates, in accordance with the values
        given in the post substituent value position set.
        Called from ELEMENT}

      TYPE PGBLIST = ^TBGLIST;
           TBGLIST = RECORD
                       GBOTTOM : PCOMBINLIST;
                       NEXT    : PGBLIST
                      END;

      VAR PTR,
          GATEBOTTOMS         : PGBLIST;
          PG                  : PPARENTLIST;
          CHILDGATEPOSITIONS,
          LIMITPOSITIONS      : TGROUPMEMS;
          LIMITINITIALISED    : BOOLEAN;
          CONNECTIVITY        : TCONNS;


      PROCEDURE GETCOMBPOSNS(CHILDPS            : PTRPSTYPE;
                             POSNSA             : INTEGSET;
                             VAR COMBAVAILPOSNS : TGROUPMEMS);

      { Returns COMBAVAILPOSNS with a COMBINED position set of all possible position
        pairs in CHILDPS having members of POSNSA as their first member.}

      VAR POSNSB,
          POSNSC : INTEGSET;

```

```
      BEGIN
      FINDPOSITIONS(CHILDPS, POSNSB, 1);
      FINDPOSITIONS(CHILDPS, POSNSC, 2);
      COMBAVAILPOSNS.COMBINED := TRUE;
      COMBAVAILPOSNS.COMBMEMS := NIL;
      LISTPOSNS(COMBAVAILPOSNS.COMBMEMS, POSNSA, POSNSB, POSNSC)
      END;



PROCEDURE TRACEDOWNGATE(COMBINBAR  : PCOMBINLIST;
                        CONNSFIXED : BOOLEAN);

{ Traces down a child gate, adding the BOTTOMBARs to the GATEBOTTOMS list.
  LIMITPOSITIONS (in MODIFYCHILDPOSITIONS) is also initialised or updated
  appropriately. If CONNSFIXED is FALSE, then the connectivity of 1 recorded
  in COMBINBAR^.CONNBONDS is only an assumption, and could be modified by the
  position set about to be read. Therefore LIMITPOSITIONS must be COMBINED,
  GETCOMBPOSNS identifying all the possible second positions for the first
  positions identified by GETCHILDPOSITIONS. }

VAR NEWGB          : PGBLIST;
    ALTERNBAR      : PALTERNLIST;
    SUBCB          : PCOMBINLIST;
    COMBAVAILPOSNS : TGROUPMEMS;

BEGIN
IF COMBINBAR^.BOTTOMBAR
   THEN BEGIN
          NEW(NEWGB);
          NEWGB^.NEXT := GATEBOTTOMS;
          NEWGB^.GBOTTOM := COMBINBAR;
          GATEBOTTOMS := NEWGB;
          WITH COMBINBAR^ DO IF LIMITINITIALISED
             THEN IF LIMITPOSITIONS.COMBINED
                     THEN IF CHILDPOSITIONS.COMBINED
                             THEN REDUCE(LIMITPOSITIONS.COMBMEMS, CHILDPOSITIONS)
                             ELSE IF CONNSFIXED
                                     THEN PROGERROR(11) {mismatched combined fields}
                                     ELSE BEGIN
```

```
                                   GETCOMBPOSNS(CHILDPS, CHILDPOSITIONS.MEMBERS, COMBAVAILPOSNS);
                                   REDUCE(LIMITPOSITIONS.COMBMEMS, COMBAVAILPOSNS);
                                   REDUCEECTR(COMBAVAILPOSNS.COMBMEMS);
                                   DESTROY(COMBAVAILPOSNS.COMBMEMS)
                                  END
                     ELSE IF CHILDPOSITIONS.COMBINED
                             THEN PROGERROR(12) {mismatched combined fields}
                             ELSE LIMITPOSITIONS.MEMBERS := LIMITPOSITIONS.MEMBERS * CHILDPOSITIONS.MEMBERS
                 ELSE BEGIN
                       IF CHILDPOSITIONS.COMBINED
                          THEN BEGIN
                                LIMITPOSITIONS.COMBINED := TRUE;
                                LIMITPOSITIONS.COMBMEMS := COPYLIST(CHILDPOSITIONS.COMBMEMS)
                               END
                          ELSE IF CONNSFIXED
                                  THEN LIMITPOSITIONS := CHILDPOSITIONS
                                  ELSE GETCOMBPOSNS(CHILDPS, CHILDPOSITIONS.MEMBERS, LIMITPOSITIONS);
                       LIMITINITIALISED := TRUE
                      END
            END
       ELSE BEGIN
             ALTERNBAR := COMBINBAR^.ALTERNATIVES;
             WHILE ALTERNBAR <> NIL DO
                BEGIN
                 SUBCB := ALTERNBAR^.COMBINATION;
                 WHILE SUBCB <> NIL DO
                    BEGIN
                     TRACEDOWNGATE(ALTERNBAR^.COMBINATION, CONNSFIXED);
                     SUBCB := SUBCB^.NEXT
                    END;
                 ALTERNBAR := ALTERNBAR^.NEXT
                END
            END
     END;



     PROCEDURE ALTERCONNBONDS(VAR CONNBONDS : TCONNBONDS);

     { Adds a second bond to CONNBONDS, changing CONNECTIONS to 2 }
```

```
VAR NEWCONNBONDS : TCONNBONDS;

BEGIN
WITH NEWCONNBONDS DO
   BEGIN
    CONNECTIONS := 2;
    BONDA := CONNBONDS.BOND;
    BONDB := CHAISING
   END;
CONNBONDS := NEWCONNBONDS
END;


BEGIN {Body of MODIFYCHILDPOSITIONS}
LIMITINITIALISED := FALSE;
CONNECTIVITY := PARALTLIST^.CONNBONDS.CONNECTIONS;
GATEBOTTOMS := NIL;
WHILE PARALTLIST <> NIL DO WITH PARALTLIST^ DO
   BEGIN
    IF ALTBAR = NIL
       THEN TRACEDOWNGATE(PARSTRUCT^.CHILDGATE, (CONNECTIVITY<>NOTSET))
       ELSE TRACEDOWNGATE(ALTBAR^.COMBINATION, (CONNECTIVITY<>NOTSET));
    PARALTLIST := NEXT
   END;
POSITIONSET(CHILDGATEPOSITIONS, LIMITPOSITIONS, CONNECTIVITY, 6);
WHILE GATEBOTTOMS <> NIL DO
   BEGIN
    WITH GATEBOTTOMS^.GBOTTOM^ DO
       BEGIN
        CHILDPOSITIONS := CHILDGATEPOSITIONS;
        IF CHILDPOSITIONS.COMBINED AND (CONNBONDS.CONNECTIONS = 1)
           THEN ALTERCONNBONDS(CONNBONDS);
        PG :=  CHILDPS^.PARENTGATE
       END;
    WHILE PG <> NIL DO WITH PG^ DO
       BEGIN
        CHILDPOSITIONS := CHILDGATEPOSITIONS;
```

```
              IF CHILDPOSITIONS.COMBINED AND (CONNBONDS.CONNECTIONS=1)
                 THEN ALTERCONNBONDS(CONNBONDS);
              PG := NEXT
             END;
        PTR := GATEBOTTOMS^.NEXT;
        DISPOSE(GATEBOTTOMS);
        GATEBOTTOMS := PTR
       END;
     NEXTTOKEN
     END;                { of MODIFYCHILDPOSITIONS
     ---------------------------------------------------------------------------}



     {---------------------------------------------------------------------------
                                PROCEDURE ELEMENT                              }

     PROCEDURE ELEMENT(PARALTLIST      : PPALTBARS;
                       OPTIONALSUB     : BOOLEAN);

     { Analyses a substituent definition element, building up the ECTR.
       Called by ALTNTVE}

     VAR OPENERS,                                {Valid tokens to begin an element}
         TERMINATORS           : DELIMSET;       {Valid tokens to end an element}
         VALID                 : BOOLEAN;
         DELIMCHECK            : DELIMTYPE;
         LIMITPOSITIONS        : TGROUPMEMS;
         GATEPARENTPOSITIONS : PTGROUPMEMS;      {Position set for inclusion in the gate}
         GATEFREQUENCY         : INTRECORD;      {Frequencies for inclusion in the gate}
         GATEPS                : PTRPSTYPE;      {The child partial structure}



     PROCEDURE SETPARENTGATE(COMBIN : PCOMBINLIST;
                             PARALT : PPALTBARS);

     { Sets up a single new parent gate at the head of the list on COMBIN^.CHILDPS^.
       Called by SETCOMBARS
                 SUBSTASVALUE}
```

```
VAR NEWPG : PPARENTLIST;

BEGIN
NEW(NEWPG);
ECTRSIZE := ECTRSIZE + 26;
NEWPG^.CHILDPOSITIONS := COMBIN^.CHILDPOSITIONS;
IF GATEPARENTPOSITIONS=NIL
   THEN NEWPG^.PARENTPOSITIONS := PARALT^.PRNTPOSNS^
   ELSE NEWPG^.PARENTPOSITIONS := GATEPARENTPOSITIONS^;
NEWPG^.PARENTPS := PARALT^.PARSTRUCT;
NEWPG^.CONNBONDS := COMBIN^.CONNBONDS;
NEWPG^.NEXT := COMBIN^.CHILDPS^.PARENTGATE;
COMBIN^.CHILDPS^.PARENTGATE := NEWPG
END;



FUNCTION NEWCOMBAR(PARALT    : PPALTBARS;
                   BARBOTTOM : BOOLEAN) : PCOMBINLIST;

{ Creates a combination bar, variant BARBOTTOM, and sets the non-variant fields,
  returning the bar as the result of the function.
  Called by SETCOMBARS
            EXTRALAYER
            NEWPARENTPSLIST}

VAR NEWCB : PCOMBINLIST;

BEGIN
IF BARBOTTOM THEN NEW(NEWCB, TRUE)
             ELSE NEW(NEWCB, FALSE);
ECTRSIZE := ECTRSIZE + 11 + (ORD(BARBOTTOM) * 13);
WITH NEWCB^ DO
   BEGIN
    PARENTPOSITIONS := GATEPARENTPOSITIONS;
    FREQUENCY := GATEFREQUENCY;
    IF PARALT^.ALTBAR = NIL
       THEN BEGIN
```

```
                {There is no alternative bar. The combination list needs to be
                 attached directly to the PS record, and PARENTPOSITIONS
                 needs to be taken from PARALT^.PRNTPOSNS if none has been obtained
                 from GATEPARENTPOSITIONS}
                NEXT := PARALT^.PARSTRUCT^.CHILDGATE;
                PARALT^.PARSTRUCT^.CHILDGATE := NEWCB;
                IF PARENTPOSITIONS = NIL THEN PARENTPOSITIONS := PARALT^.PRNTPOSNS
               END
         ELSE BEGIN
                {The combination list needs to be attached to the alternative bar}
                NEXT := PARALT^.ALTBAR^.COMBINATION;
                PARALT^.ALTBAR^.COMBINATION := NEWCB
               END;
      BOTTOMBAR := BARBOTTOM
     END;
  NEWCOMBAR := NEWCB
  END;



  {-----------------------------------------------------------------------------}
  PROCEDURE SETCOMBARS(PARALTLIST : PPALTBARS;
                       GATEPS     : PTRPSTYPE);

  { Sets up the child and parent gates for all the items in PARALTLIST
    Called by Body of ELEMENT}

  VAR NEWCOMBIN : PCOMBINLIST;



  FUNCTION NEEDTOCHECK(PARENTPS     : PTRPSTYPE;
                       NEWCONNBONDS,
                       OLDCONNBONDS : TCONNBONDS) : BOOLEAN;

  BEGIN
  IF PARENTPS = NIL
     THEN NEEDTOCHECK := FALSE
     ELSE IF PARENTPS^.PSVARIETY = SPECIFIC
            THEN CASE NEWCONNBONDS.CONNECTIONS OF
```

```
                    NOTSET,
                    0       : PROGERROR(13);
                    1       : CASE OLDCONNBONDS.CONNECTIONS OF
                               NOTSET : NEEDTOCHECK := MAGNITUDE(NEWCONNBONDS.BOND) > 1;
                               0, 2   : PROGERROR(14);
                               1      : NEEDTOCHECK := MAGNITUDE(NEWCONNBONDS.BOND) > MAGNITUDE(OLDCONNBONDS.B
ND)
                               END;
                    2       : CASE OLDCONNBONDS.CONNECTIONS OF
                               NOTSET : NEEDTOCHECK := (MAGNITUDE(NEWCONNBONDS.BONDA) > 1)
                                            OR (MAGNITUDE(NEWCONNBONDS.BONDB) > 1);
                               0, 1   : PROGERROR(15);
                               2      : NEEDTOCHECK := (MAGNITUDE(NEWCONNBONDS.BONDA) > MAGNITUDE(OLDCONNBONDS
BONDA))
                                            OR (MAGNITUDE(NEWCONNBONDS.BONDB) > MAGNITUDE(OLDCONNBONDS.BONDB))

                               END
                    END
                ELSE NEEDTOCHECK := FALSE
END;



FUNCTION ORIGINALPOSNS(PARENTPS     : PTRPSTYPE;
                       LASTPARPOSNS : PTGROUPMEMS) : BOOLEAN;

{ Returns TRUE if LASTPARPOSNS is the PRNTPOSNS field of any of the child gates
  leading from PARENTPS}

VAR CGPTR : PCOMBINLIST;
    ROWNO : ATOMNUMBER;
    FOUND : BOOLEAN;

BEGIN
FOUND := FALSE;
CGPTR := PARENTPS^.CHILDGATE;
WHILE (CGPTR <> NIL) AND NOT FOUND DO WITH CGPTR^ DO
   BEGIN
    FOUND := (PARENTPOSITIONS = LASTPARPOSNS);
    CGPTR := NEXT
```

```
       END;
    IF NOT FOUND THEN WITH PARENTPS^ DO
       FOR ROWNO := 1 TO MAXCT DO IF CT[ROWNO] <> NIL
          THEN WITH CT[ROWNO]^ DO IF NOT ATOMICROW
                 THEN BEGIN
                        CGPTR := VALUES;
                        WHILE (CGPTR <> NIL) AND NOT FOUND DO WITH CGPTR^ DO
                           BEGIN
                            FOUND := (PARENTPOSITIONS = LASTPARPOSNS);
                            CGPTR := NEXT
                           END
                        END;
    ORIGINALPOSNS := FOUND
    END;



    PROCEDURE CHECKPOSNS(VAR BOTPARPOSNS : PTGROUPMEMS;
                             LASTPARPOSNS    : PTGROUPMEMS;
                             PARENTPS        : PTRPSTYPE;
                             BONDMAG         : TBONDMAG);

    { Checks that any positions given in position sets are actually available for
      a bond of the MAGNITUDE in question, as this was not previously known. }

    VAR NOPOSNSGIVEN : BOOLEAN;
        GIVENPOSNS   : INTEGSET;
        POSN         : ATOMNUMBER;

    BEGIN
    IF BOTPARPOSNS = NIL
       THEN BEGIN
             NOPOSNSGIVEN := ORIGINALPOSNS(PARENTPS, LASTPARPOSNS);
             IF NOPOSNSGIVEN
                THEN BEGIN
                      NEW(BOTPARPOSNS);
                      ECTRSIZE := ECTRSIZE + 9;
                      BOTPARPOSNS^ := LASTPARPOSNS^
                     END;
             GIVENPOSNS := LASTPARPOSNS^.MEMBERS
```

```
                END
           ELSE BEGIN
                 NOPOSNSGIVEN := FALSE;
                 GIVENPOSNS := BOTPARPOSNS^.MEMBERS
                END;
        FOR POSN := 1 TO MAXCT DO IF POSN IN GIVENPOSNS
           THEN WITH PARENTPS^.CT[POSN]^ DO IF ATOMICROW
                   THEN IF HYDROGENS < BONDMAG
                           THEN IF NOPOSNSGIVEN
                                   THEN WITH BOTPARPOSNS^ DO MEMBERS := MEMBERS - [POSN]
                                   ELSE FAILURE(45, POSN, '    ');
        IF BOTPARPOSNS^.MEMBERS = [] THEN FAILURE(46, 0, '    ')
        END;



        FUNCTION COMBINEDPOSITIONS(CONNBONDS  : TCONNBONDS;
                                   LIMITPOSNS : INTEGSET;
                                   PARENTPS   : PTRPSTYPE) : PTGROUPMEMS;

        { Returns a COMBINED position set based on the available positions in PARENTPS.
          If the bond magnitudes are 1, then the previously-determined LIMITPOSNS can
          be used instead of calling GETAVAILABLEPOSITIONS.}

        VAR COMBPOSNS : PTGROUPMEMS;
            MAGNIT    : INTEGER;
            POSNSETA,
            POSNSETB,
            POSNSETC  : INTEGSET;

        BEGIN
        NEW(COMBPOSNS);
        ECTRSIZE := ECTRSIZE + 9;
        COMBPOSNS^.COMBINED := TRUE;
        COMBPOSNS^.COMBMEMS := NIL;
        MAGNIT := MAGNITUDE(CONNBONDS.BONDA);
        IF MAGNIT > 1 THEN GETAVAILABLEPOSITIONS(PARENTPS, POSNSETA, MAGNIT)
                      ELSE POSNSETA := LIMITPOSNS;
        MAGNIT := MAGNITUDE(CONNBONDS.BONDB);
        IF MAGNIT > 1 THEN GETAVAILABLEPOSITIONS(PARENTPS, POSNSETB, MAGNIT)
```

```
                        ELSE POSNSETB := LIMITPOSNS;
MAGNIT := MAGNIT + MAGNITUDE(CONNBONDS.BONDA);
IF MAGNIT <= 3 THEN GETAVAILABLEPOSITIONS(PARENTPS, POSNSETC, MAGNIT)
                        ELSE POSNSETC := [];
POSNSETC := POSNSETA * POSNSETB * POSNSETC;
LISTPOSNS(COMBPOSNS^.COMBMEMS, POSNSETA, POSNSETB, POSNSETC);
IF COMBPOSNS^.COMBMEMS = NIL THEN FAILURE(47, 0, '    ');
COMBINEDPOSITIONS := COMBPOSNS
END;



PROCEDURE CHECKCOMBPOSNS(VAR BOTPARPOSNS : PTGROUPMEMS;
                             LASTPARPOSNS    : PTGROUPMEMS;
                             PARENTPS        : PTRPSTYPE;
                             MAGA,
                             MAGB            : TBONDMAG);

{ Checks that any positions specified in position sets are actually available
  for bonds of the MAGNITUDEs in question, which were not previously known.}

VAR REMOVEA,
    REMOVEB      : INTEGSET;
    NOPOSNSGIVEN : BOOLEAN;
    GIVENPOSNS,
    LISTPTR,
    DELPTR       : PDOUBLIST;

BEGIN
IF BOTPARPOSNS= NIL
   THEN BEGIN
          NOPOSNSGIVEN := ORIGINALPOSNS(PARENTPS, LASTPARPOSNS);
          GIVENPOSNS := COPYLIST(LASTPARPOSNS^.COMBMEMS)
        END
   ELSE BEGIN
          NOPOSNSGIVEN := FALSE;
          GIVENPOSNS :=  COPYLIST(BOTPARPOSNS^.COMBMEMS)
        END;
REMOVEA := [];
REMOVEB := [];
```

```
LISTPTR := GIVENPOSNS;
WHILE LISTPTR <> NIL DO
   BEGIN
    WITH PARENTPS^.CT[LISTPTR^.FIRST]^ DO IF ATOMICROW
       THEN IF HYDROGENS < MAGA
               THEN IF NOPOSNSGIVEN
                       THEN REMOVEA := REMOVEA + [LISTPTR^.FIRST]
                       ELSE FAILURE(45, LISTPTR^.FIRST, '    ');
    WITH PARENTPS^.CT[LISTPTR^.SECOND]^ DO IF ATOMICROW
       THEN IF HYDROGENS < MAGB
               THEN IF NOPOSNSGIVEN
                       THEN REMOVEB := REMOVEB + [LISTPTR^.SECOND]
                       ELSE FAILURE(45, LISTPTR^.SECOND, '    ');
    LISTPTR := LISTPTR^.NEXT
   END;
 IF REMOVEA + REMOVEB = []
    THEN BEGIN
          REDUCEECTR(GIVENPOSNS);
          DESTROY(GIVENPOSNS)
         END
    ELSE BEGIN
          LISTPTR := GIVENPOSNS;
          WHILE LISTPTR <> NIL DO IF (LISTPTR^.FIRST IN REMOVEA) OR (LISTPTR^.SECOND IN REMOVEB)
             THEN BEGIN
                   IF LISTPTR = GIVENPOSNS
                      THEN GIVENPOSNS := LISTPTR^.NEXT;
                   DELPTR := LISTPTR;
                   LISTPTR := LISTPTR^.NEXT;
                   DISPOSE(DELPTR);
                   ECTRSIZE := ECTRSIZE - 6
                  END
             ELSE LISTPTR := LISTPTR^.NEXT;
          IF GIVENPOSNS = NIL THEN FAILURE(48, 0, '    ');
          IF BOTPARPOSNS = NIL
             THEN BEGIN
                   NEW(BOTPARPOSNS);
                   ECTRSIZE := ECTRSIZE + 9;
                   BOTPARPOSNS^.COMBINED := TRUE
                  END;
          BOTPARPOSNS^.COMBMEMS := GIVENPOSNS
```

```
              END
        END;



        BEGIN {Body of SETCOMBARS}
        ADDTOLIST(GATEPS);
        WHILE PARALTLIST <> NIL DO
           BEGIN
            NEWCOMBIN := NEWCOMBAR(PARALTLIST, FALSE);
            IF PARALTLIST^.COPYCHILDPS
               THEN NEWCOMBIN^.CHILDPS := COPYPS(GATEPS)
               ELSE NEWCOMBIN^.CHILDPS := GATEPS;
            NEWCOMBIN^.CONNBONDS := PARALTLIST^.CONNBONDS;
            GETCHILDPOSITIONS(GATEPS,NEWCOMBIN^. CONNBONDS, NEWCOMBIN^.CHILDPOSITIONS);
            CASE NEWCOMBIN^.CONNBONDS.CONNECTIONS OF
             NOTSET : PROGERROR(16);
             0      : ;
             1      : IF NEEDTOCHECK(PARALTLIST^.PARSTRUCT, NEWCOMBIN^.CONNBONDS, PARALTLIST^.CONNBONDS)
                        THEN CHECKPOSNS(NEWCOMBIN^.PARENTPOSITIONS,
                                        PARALTLIST^.PRNTPOSNS,
                                        PARALTLIST^.PARSTRUCT,
                                        MAGNITUDE(NEWCOMBIN^.CONNBONDS.BOND));
             2      : IF (NEWCOMBIN^.PARENTPOSITIONS = NIL) AND NOT PARALTLIST^.PRNTPOSNS^.COMBINED
                        THEN {no position set can have been given, as it would have had to have been
                             COMBINED. Consequently we can reestablish PARENTPOSITIONS from scratch}
                             IF PARALTLIST^.PARSTRUCT <> NIL THEN
                                NEWCOMBIN^.PARENTPOSITIONS := COMBINEDPOSITIONS(NEWCOMBIN^.CONNBONDS,
                                                                 PARALTLIST^.PRNTPOSNS^.MEMBERS,
                                                                 PARALTLIST^.PARSTRUCT)
                        ELSE {any position sets we have must be COMBINED}
                             IF NEEDTOCHECK(PARALTLIST^.PARSTRUCT, NEWCOMBIN^.CONNBONDS, PARALTLIST^.CONNBONDS)
                                THEN CHECKCOMBPOSNS(NEWCOMBIN^.PARENTPOSITIONS,
                                                    PARALTLIST^.PRNTPOSNS,
                                                    PARALTLIST^.PARSTRUCT,
                                                    MAGNITUDE(NEWCOMBIN^.CONNBONDS.BONDA),
                                                    MAGNITUDE(NEWCOMBIN^.CONNBONDS.BONDB));
             END;
            IF NOT DEFNTABLEENTRY(PARALTLIST^.PARSTRUCT)
               THEN SETPARENTGATE(NEWCOMBIN, PARALTLIST);
```

```
       PARALTLIST := PARALTLIST^.NEXT
      END
   END;    {of SETCOMBARS
   ------------------------------------------------------------------------}



   FUNCTION NEWPARENTPSLIST(PARALTLIST : PPALTBARS): PPSLIST;

   { Sets up a new PPSLIST based on the items in PARALTLIST, which can be passed
     in a recursive call to ALTNVLIST. This procedure also establishes non-BOTTOMBAR
     combination bars in the gates for all items in PARALTLIST.
     Called from TRANSLATENOMEN
                 Body of ELEMENT}

   VAR LISTPTR,                    { Points to the top of the growing list }
       WRITEPTR  : PPSLIST;        { New addition to the list }
       NEWCOMBIN : PCOMBINLIST;    { New combination gate }

   BEGIN
   LISTPTR := NIL;
   WHILE PARALTLIST <> NIL DO
      BEGIN
       NEWCOMBIN := NEWCOMBAR(PARALTLIST, TRUE);
       NEWCOMBIN^.ALTERNATIVES := NIL;
       NEW(WRITEPTR);
       WRITEPTR^.NEXT := LISTPTR;
       WRITEPTR^.PARSTRUCT := PARALTLIST^.PARSTRUCT;
       WRITEPTR^.CONNBONDS := PARALTLIST^.CONNBONDS;
       IF GATEPARENTPOSITIONS = NIL
          THEN WRITEPTR^.PRNTPOSNS := PARALTLIST^.PRNTPOSNS
          ELSE WRITEPTR^.PRNTPOSNS := GATEPARENTPOSITIONS;
       WRITEPTR^.COMBINS := NEWCOMBIN;
       WRITEPTR^.COPYCHILDPS := PARALTLIST^.COPYCHILDPS;
       WRITEPTR^.FURTHERSUB := NIL;
       LISTPTR := WRITEPTR;
       PARALTLIST := PARALTLIST^.NEXT
      END;
   NEWPARENTPSLIST := LISTPTR
   END;
```

```
PROCEDURE GETSPSPARAMS(VAR GATEPS : PTRPSTYPE);

{ Sets up a GENERIC PS, and initialises the PARAMLIST with the global
  SPSPARAMLIST, which has been set up by SPSVARIETY.
  Called by TRANSLATENOMEN}

VAR PARAM : TPARAMETERS;
    PTR   : PDOUBLIST;

BEGIN
NEW(GATEPS, GENERIC);
ECTRSIZE := ECTRSIZE + 50;
WITH GATEPS^ DO
   BEGIN
    PSVARIETY := GENERIC;
    VISITED := FALSE;
    PARENTGATE := NIL;
    CHILDGATE := NIL;
    PARAMLIST := SPSPARAMLIST;
    FOR PARAM := ATOMCOUNT TO HETEROATOM DO
       BEGIN
        PTR := PARAMLIST[PARAM].SUBRANGES;
        WHILE PTR <> NIL DO
           BEGIN
            ECTRSIZE := ECTRSIZE + 6;
            PTR := PTR^.NEXT
           END
       END
   END
END;



{----------------------------------------------------------------------------}
PROCEDURE PARAMETERLIST(GATEPS : PTRPSTYPE);

{ Analyses a Gensal parameter list. For standard parameters, the information
  is stored in the appropriate element of PARAMLIST, this being determined
```

```
 by the function PARAMETER. The existing parameter values are used
 to limit those analysed, by creating LIMITSET, to be passed to SELECTOR.
 In SELECTOR this is converted back to an INTRECORD for passing to
 INTEGERRANGE, but this is not as silly as it seems, as otherwise
 INTEGERRANGE would be using the same INTRECORD linked list both as
 LIMITRANGE and RANGEVALUES. If GATEPS is not GENERIC then the values for
 each parameter are placed in DUMMYPARAM, and any PDOUBLIST linked list
 immediately DESTROYed.  Non-standard parameters are handled by USERPARAMETER.
 Called by TRANSLATENOMEN}

VAR PARAMIDS   : DELIMSET;
    DELIMCHECK : DELIMTYPE;
    DUMMYPARAM : INTRECORD;
    LIMITSET   : INTEGSET;



FUNCTION PARAMETER(PARAMDELIM : DELIMTYPE) : TPARAMETERS;

{ Returns the PARAMETER that is equivalent to the delimiter passed as PARAMDELIM}

BEGIN
CASE PARAMDELIM OF
 GC  : PARAMETER := ATOMCOUNT;
 GT  : PARAMETER := TBRANCH;
 GQ  : PARAMETER := QBRANCH;
 GE  : PARAMETER := EUNSATURATION;
 GY  : PARAMETER := YUNSATURATION;
 GRC : PARAMETER := RINGCOUNT;
 GRN : PARAMETER := RINGATOMS;
 GRS : PARAMETER := RINGSUBSTITUTION;
 GRF : PARAMETER := RINGFUSIONS;
 GRA : PARAMETER := RINGAROMATIC;
 GZ  : PARAMETER := HETEROATOM
 END;
END;



PROCEDURE FINDCONNECTIONS(VAR CONNBONDS : TCONNBONDS;
```

```
                              VAR PRNTPOSNS : PTGROUPMEMS;
                              PSADDRESS     : PTRPSTYPE;
                              SUBST         : SUBSTITUENT);

{ Sets CONNBONDS and PRNTPOSNS for the declaration of SUBST referencing
  PSADDRESS.  Any previous declaration of SUBST will give a value for
  CONNBONDS - otherwise CONNBONDS.CONNECTIONS is set to NOTSET.
  Since PSADDRESS^.PSVARIETY is GENERIC or OTHER, the bond magnitude is
  irrelevant to the determination of the available positions - hence a
  dummy value of 1 is passed to GETAVAILABLEPOSITIONS.
  Called by USERPARAMETER}

VAR AVAILPOSNS : INTEGSET;

BEGIN
IF SUBST IN DECLSUBS THEN CONNBONDS := RDECLARATIONTABLE[SUBST]^.CONNBONDS
                     ELSE CONNBONDS.CONNECTIONS := NOTSET;
NEW(PRNTPOSNS);
ECTRSIZE := ECTRSIZE + 9;
PRNTPOSNS^.COMBINED := CONNBONDS.CONNECTIONS = 2;
CASE CONNBONDS.CONNECTIONS OF
 0      : PRNTPOSNS^.MEMBERS := [];
 NOTSET,
 1      : GETAVAILABLEPOSITIONS(PSADDRESS, PRNTPOSNS^.MEMBERS, 1);
 2      : BEGIN
           GETAVAILABLEPOSITIONS(PSADDRESS, AVAILPOSNS, 1);
           PRNTPOSNS^.COMBMEMS := NIL;
           LISTPOSNS(PRNTPOSNS^.COMBMEMS, AVAILPOSNS+[0], AVAILPOSNS+[0], AVAILPOSNS)
          END
 END
END;



PROCEDURE USERPARAMETER(GATEPS : PTRPSTYPE);

{ Analyses a user-defined parameter. FINDCONNECTIONS is used to determine
  CONNBONDS and PARENTPOSITIONS, for passing to DECLARESUBST.}

VAR NEWCOMBIN       : PCOMBINLIST;
```

```
    CONNBONDS       : TCONNBONDS;
    PARENTPOSITIONS : PTGROUPMEMS;

BEGIN
NEXTTOKEN;
WHILE CHECKDELIM([GR])=INVALIDTOKEN DO ERROR(26,0);
NEXTTOKEN;
REPEAT
 WHILE TOKEN.NATURE<>INTEGRAL DO ERROR(23,0);
 IF NOT (TOKEN.INTEGVAL IN [1..MAXVARS]) THEN ERROR(28,0)
UNTIL TOKEN.INTEGVAL IN [1..MAXVARS];
FINDCONNECTIONS(CONNBONDS, PARENTPOSITIONS, GATEPS, TOKEN.INTEGVAL);
DECLARESUBST(TOKEN.INTEGVAL,      {subst}
             GATEPS,              {psaddress}
             NIL,                 {savps}
             CONNBONDS,           {connbonds}
             PARENTPOSITIONS);    {prntposns}
ENTERCOMBIN(TOKEN.INTEGVAL, GATEPS^.CHILDGATE);
NEXTTOKEN;
WHILE CHECKDELIM([GPRIME]) = INVALIDTOKEN DO ERROR(29,0);
NEXTTOKEN;
SELECTOR(GATEPS^.CHILDGATE^.FREQUENCY, [0..MAXVARS], 5)
END;



BEGIN  {Body of PARAMETERLIST}
PARAMIDS := [GC, GT, GQ, GE, GY, GRC, GRN, GRS, GRF, GRA, GZ];
REPEAT
 DELIMCHECK := CHECKDELIM([GOPENANG,GLSQUARE,GPRIME]+PARAMIDS+TERMINATORS);
 IF DELIMCHECK=INVALIDTOKEN THEN ERROR(24,0)
UNTIL DELIMCHECK <> INVALIDTOKEN;
IF DELIMCHECK = GOPENANG
   THEN DELIMCHECK := GC {Parameter identifier not needed}
   ELSE IF DELIMCHECK IN PARAMIDS THEN NEXTTOKEN;
WHILE DELIMCHECK IN PARAMIDS+[GPRIME] DO
   BEGIN
    IF DELIMCHECK = GPRIME
       THEN USERPARAMETER(GATEPS)
       ELSE IF GATEPS^.PSVARIETY = GENERIC
```

```
                  THEN BEGIN
                        INTSET(LIMITSET, GATEPS^.PARAMLIST[PARAMETER(DELIMCHECK)]);
                        SELECTOR(GATEPS^.PARAMLIST[PARAMETER(DELIMCHECK)], LIMITSET, 5)
                       END
                  ELSE BEGIN
                        SELECTOR(DUMMYPARAM, [0..MAXVARS], 5);
                        REDUCEECTR(DUMMYPARAM.SUBRANGES);
                        DESTROY(DUMMYPARAM.SUBRANGES)
                       END;
       NEXTTOKEN;
       REPEAT
        DELIMCHECK := CHECKDELIM([GPRIME,GLSQUARE]+PARAMIDS+TERMINATORS);
        IF DELIMCHECK=INVALIDTOKEN THEN ERROR(24,0)
       UNTIL DELIMCHECK <> INVALIDTOKEN;
       IF DELIMCHECK IN PARAMIDS THEN NEXTTOKEN
      END
   END;
          { of PARAMETERLIST
   -----------------------------------------------------------------------------}



   {-----------------------------------------------------------------------------}
   PROCEDURE TRANSLATENOMEN(VAR GATEPS : PTRPSTYPE);

   { Determines whether or not a record is held for the current TOKEN.NOMENVAL
     (taking synonyms in RECORDHELD, if terminal input), and sets up an
     appropriate child PS, or pushes down the input environment and makes a
     recursive call to ALTNVLIST. If no record is held, then an OTHER PS is set
     up.
     Called from body of ELEMENT}

   VAR ADDRESS              : INTEGER;       { SPSfile address for TOKEN.NOMENVAL }
       OLDN                 : 0..MAXLENGTH;  { Saved value of N }
       OLDBUFFER            : LINESTRING;    { Saved value of BUFFER }
       INSERTLINES,                          { Lines being inserted }
       OLDCURRENTLINE       : PLINELIST;     { Saved value of CURRENTLINE }
       OLDMODE              : TINPUTMODE;    { Saved INPUTMODE }
```

```

PROCEDURE MODIFYGATEPOSITIONS(COMBINBAR     : PCOMBINLIST;
                              HSTAVAILPOSNS : PTGROUPMEMS);

{ Traces down a child gate (headed by COMBINBAR), altering the PARENTPOSITIONS
  field in each, to conform to HSTAVAILPOSNS. If the  bar is BOTTOMBAR
  then the PARENTPOSITIONS fields of the  corresponding parent gates from the
  CHILDPS are altered similarly; otherwise the procedure recurses on itself
  for each COMBINATION  in the ALTERNATIVES.}

VAR ALTERNBAR  : PALTERNLIST;
    PARENTGATE : PPARENTLIST;
    TOPBAR     : BOOLEAN;
    POSN       : ATOMNUMBER;

BEGIN
TOPBAR := (HSTAVAILPOSNS = NIL);
IF TOPBAR THEN
   BEGIN
    NEW(HSTAVAILPOSNS);
    WITH HSTAVAILPOSNS^ DO
       BEGIN
        COMBINED := FALSE;
        GETAVAILABLEPOSITIONS(INSERTHSTPS, MEMBERS, 1);
        MEMBERS := MEMBERS + [0]
       END
   END;
WHILE COMBINBAR <> NIL DO WITH COMBINBAR^ DO
   BEGIN
    IF PARENTPOSITIONS <> NIL THEN WITH PARENTPOSITIONS^ DO
       IF TOPBAR
          THEN IF COMBINED
                  THEN BEGIN
                        REDUCE(COMBMEMS, HSTAVAILPOSNS^);
                        IF COMBMEMS=NIL THEN FAILURE(4, 0, '    ')
                       END
                  ELSE BEGIN
                        MEMBERS := MEMBERS * HSTAVAILPOSNS^.MEMBERS;
                        IF MEMBERS = [] THEN FAILURE(4, 0, '    ')
                       END
```

```
                ELSE IF COMBINED
                        THEN CHECKALLWITHIN(COMBMEMS, HSTAVAILPOSNS^.MEMBERS, 3)
                        ELSE IF MEMBERS <= HSTAVAILPOSNS^.MEMBERS
                                THEN {OK}
                                ELSE FOR POSN := 1 TO MAXCT DO
                                        IF (POSN IN MEMBERS) AND NOT (POSN IN HSTAVAILPOSNS^.MEMBERS)
                                           THEN FAILURE(3, POSN, '     ');
           IF BOTTOMBAR
              THEN BEGIN
                    PARENTGATE := CHILDPS^.PARENTGATE;
                    WHILE PARENTGATE <> NIL DO WITH PARENTGATE^ DO
                       BEGIN
                        IF PARENTPS = INSERTHSTPS
                           THEN WITH PARENTPOSITIONS DO
                                   IF COMBINED
                                      THEN {no need to do anything - pointers point to same PDOUBLIST in parent a
d child gates}
                                      ELSE MEMBERS := MEMBERS * HSTAVAILPOSNS^.MEMBERS;
                        PARENTGATE := NEXT
                       END
                   END
              ELSE BEGIN
                    ALTERNBAR := ALTERNATIVES;
                    WHILE ALTERNBAR <> NIL DO WITH ALTERNBAR^ DO
                       BEGIN
                        MODIFYGATEPOSITIONS(COMBINATION, HSTAVAILPOSNS);
                        ALTERNBAR := NEXT
                       END
                   END;
           COMBINBAR := NEXT
          END;
       IF TOPBAR THEN DISPOSE(HSTAVAILPOSNS)

       END;



       BEGIN  { Body of Procedure TRANSLATENOMEN }
       IF RECORDHELD(TOKEN.NOMENVAL, ADDRESS)
          THEN CASE SPSVARIETY(ADDRESS, FALSE) OF
```

```
SPECIFIC : BEGIN
             NEW(GATEPS, SPECIFIC);
             ECTRSIZE := ECTRSIZE + 70;
             WITH GATEPS^ DO
                BEGIN
                 PSVARIETY := SPECIFIC;
                 VISITED := FALSE;
                 PARENTGATE := NIL;
                 CHILDGATE := NIL;
                 PROCESSCT(CT,FALSE, GATEPS)
                END;
             NEXTTOKEN
            END;

GENERIC  : BEGIN
             GETSPSPARAMS(GATEPS);
             NEXTTOKEN;
             IF CHECKDELIM([GLSQUARE]+TERMINATORS)=INVALIDTOKEN
                THEN PARAMETERLIST(GATEPS);
             IF INPUTMODE = INSERTTEXT
                THEN IF INSERTHSTPS = NIL
                       THEN INSERTHSTPS := GATEPS
                       ELSE PROGERROR(17) { multiple HSTs in SPSfile expression}
            END;

OTHER    : BEGIN
             GATEPS := NIL;
             { Save current environment }
              OLDCURRENTLINE := CURRENTLINE;
              CURRENTLINE := INSERTGENEX;
              OLDMODE := INPUTMODE;
              INPUTMODE := INSERTTEXT;
              OLDBUFFER := BUFFER;
              OLDN := N;
              N := MAXLENGTH;
             IF OLDMODE <> INSERTTEXT THEN INSERTHSTPS := NIL;
             INSERTLINES := INSERTGENEX;
             ALTNVLIST(NEWPARENTPSLIST(PARALTLIST),  FALSE);
             IF CHECKDELIM( [GRPAREN])=INVALIDTOKEN THEN
                PROGERROR(18); {missing ")" in SPSfile expression}
```

```
                              { Restore former environment }
                               CURRENTLINE := OLDCURRENTLINE;
                               BUFFER := OLDBUFFER;
                               INPUTMODE := OLDMODE;
                               N := OLDN;
                              DELETEGENSAL(INSERTLINES);
                              NEXTTOKEN;
                              IF (INPUTMODE <> INSERTTEXT) AND (INSERTHSTPS <> NIL)
                                 THEN IF CHECKDELIM([GLSQUARE]+TERMINATORS)=INVALIDTOKEN
                                         THEN BEGIN
                                                PARAMETERLIST(INSERTHSTPS);
                                                IF INSERTHSTPS^.CHILDGATE <> NIL THEN
                                                   MODIFYGATEPOSITIONS(INSERTHSTPS^.CHILDGATE, NIL)
                                              END
                            END
                END
     ELSE BEGIN
                 NEW(GATEPS, OTHER);
                 ECTRSIZE := ECTRSIZE + 22;
                 WITH GATEPS^ DO
                    BEGIN
                     PSVARIETY := OTHER;
                     VISITED := FALSE;
                     PARENTGATE := NIL;
                     CHILDGATE := NIL;
                     TERM := TOKEN.NOMENVAL
                    END;
                 NEXTTOKEN;
                 IF CHECKDELIM([GLSQUARE]+TERMINATORS)=INVALIDTOKEN
                    THEN PARAMETERLIST(GATEPS)
                END
  END;                    { of TRANSLATENOMEN
  ----------------------------------------------------------------------}



  FUNCTION EXTRALAYER(PARALT     : PPALTBARS;
                      DUMMYSAVPS : PTRPSTYPE) : PCOMBINLIST;

  { Creates an extra layer in the gate of which PARALT gives an ALTERNATIVE bar,
```

```
  inserts one alternative into it, for DUMMYSAVPS.
  Called by SUBSTASVALUE}

VAR XLAYER  : PCOMBINLIST;

BEGIN
XLAYER := NEWCOMBAR(PARALT, TRUE);
NEW(XLAYER^.ALTERNATIVES);
ECTRSIZE := ECTRSIZE + 4;
WITH XLAYER^, ALTERNATIVES^ DO
   BEGIN
    NEXT := NIL;
    NEW(COMBINATION, FALSE);
    ECTRSIZE := ECTRSIZE + 24;
    WITH COMBINATION^ DO
       BEGIN
        PARENTPOSITIONS := NIL;
        FREQUENCY.TOPRANGE := NOTSET;
        FREQUENCY.SUBRANGES := ESSENTFREQ;
        NEXT := NIL;
        BOTTOMBAR := TRUE;
        CHILDPS := DUMMYSAVPS;
        CONNBONDS := PARALT^.CONNBONDS;
        WITH CHILDPOSITIONS DO
           BEGIN
            COMBINED := CONNBONDS.CONNECTIONS = 2;
            IF COMBINED
               THEN BEGIN
                      COMBMEMS := NIL;
                      LISTPOSNS(COMBMEMS, [1..MAXCT], [1..MAXCT], [1..MAXCT])
                    END
               ELSE MEMBERS := [1..MAXCT]
           END
       END
   END;
EXTRALAYER := XLAYER
END;
```

```
FUNCTION VALIDSUBST(PARALTLIST : PPALTBARS) : SUBSTITUENT;

{ Obtains a susbtituent name and checks that it is in the range 1-63, and
  that if it is previously declared, its connectivity is compatible with
  that of the items in PARALTLIST.
  Called by SUBSTASVALUE}

LABEL 20;

BEGIN
20 :
WHILE TOKEN.NATURE <> INTEGRAL DO ERROR(23,0);
IF NOT (TOKEN.INTEGVAL IN [1..MAXVARS])
   THEN BEGIN
          ERROR(28,0);
          GOTO 20
        END
   ELSE IF TOKEN.INTEGVAL IN DECLSUBS
          THEN WITH RDECLARATIONTABLE[TOKEN.INTEGVAL]^ DO
                 IF CONNBONDS.CONNECTIONS = NOTSET
                   THEN IF PARALTLIST^.CONNBONDS.CONNECTIONS = NOTSET
                          THEN {no further information}
                          ELSE BEGIN
                                 SETCONNBONDS(CONNBONDS, PARALTLIST^.CONNBONDS.CONNECTIONS);
                                 UPDATEPPSCONNS(RDECLARATIONTABLE[TOKEN.INTEGVAL])
                               END
                   ELSE IF PARALTLIST^.CONNBONDS.CONNECTIONS = NOTSET
                          THEN BEGIN
                                 SETCONNBONDS(PARALTLIST^.CONNBONDS, CONNBONDS.CONNECTIONS);
                                 UPDATEPARALTCONNS(PARALTLIST)
                               END
                          ELSE IF CONNBONDS.CONNECTIONS = PARALTLIST^.CONNBONDS.CONNECTIONS
                                 THEN {compatible}
                                 ELSE BEGIN
                                        ERROR(31,0);
                                        GOTO 20
                                      END;
VALIDSUBST := TOKEN.INTEGVAL
END;
```

```
PROCEDURE SUBSTASVALUE(PARALTLIST : PPALTBARS);

{ Analyses a substituent given as a substituent value, creates a DUMMY PS, and
  declares it using DECLARESUBST for each of the items in PARALTLIST. If
  the substituent has already been defined, copies the definition.
  Called from the body of ELEMENT}

VAR SUBST          : SUBSTITUENT;
    DUMMYCHILD,
    DUMMYSAVPS     : PTRPSTYPE;
    DECNPPOSNS     : PTGROUPMEMS;
    PREVDEFN       : PALTERNLIST;
    OMITPG         : BOOLEAN;
    GATECONNBONDS  : TCONNBONDS;
    DUMMYPOSNS     : TGROUPMEMS;

BEGIN
NEXTTOKEN;
SUBST := VALIDSUBST(PARALTLIST);
NEW(DUMMYSAVPS, DUMMY);
ECTRSIZE := ECTRSIZE + 8;
WITH DUMMYSAVPS^ DO
   BEGIN
    PSVARIETY := DUMMY;
    VISITED := FALSE;
    CHILDGATE := NIL;
    PARENTGATE := NIL;
    SUBSTNAME := SUBST
   END;
ADDTOLIST(DUMMYSAVPS);
GATECONNBONDS := PARALTLIST^.CONNBONDS;
GETCHILDPOSITIONS(DUMMYSAVPS, GATECONNBONDS, DUMMYPOSNS);

WHILE PARALTLIST <> NIL DO WITH PARALTLIST^ DO
   BEGIN
    IF GATEPARENTPOSITIONS=NIL
       THEN DECNPPOSNS := PRNTPOSNS
       ELSE DECNPPOSNS := GATEPARENTPOSITIONS;
```

```
      IF PARALTLIST^.COPYCHILDPS THEN DUMMYCHILD := COPYPS(DUMMYSAVPS)
                                  ELSE DUMMYCHILD := DUMMYSAVPS;
      DECLARESUBST(SUBST,
                   PARSTRUCT,
                   DUMMYCHILD,
                   CONNBONDS,
                   DECNPPOSNS);
      OMITPG := DEFNTABLEENTRY(PARSTRUCT);
      IF SUBST IN DEFNSUBS
         THEN PREVDEFN := RDEFINITIONTABLE[SUBST]^.ALTERNATIVES
         ELSE PREVDEFN := NIL;
      RDECLARATIONTABLE[SUBST]^.COMBINS := EXTRALAYER(PARALTLIST, DUMMYCHILD);
      IF NOT OMITPG THEN
         SETPARENTGATE(RDECLARATIONTABLE[SUBST]^.COMBINS^.ALTERNATIVES^.COMBINATION, PARALTLIST);
      WHILE PREVDEFN <> NIL DO WITH RDECLARATIONTABLE[SUBST]^ DO
         BEGIN
          COPYALTBAR(COMBINS^.ALTERNATIVES,  {newaltbar}
                     PREVDEFN^.COMBINATION,  {oldcomblist}
                     COMBINS,                {lastcomblayer}
                     PRNTPOSNS,              {lastpposns}
                     PARSTRUCT,              {prntps}
                     FALSE,                  {firstbar}
                     OMITPG,                 {omitpg}
                     TRUE);                  {copypss}
          PREVDEFN := PREVDEFN^.NEXT
         END;
      PARALTLIST := NEXT
     END
  END;



  FUNCTION NOVARIABLESUBTN(LIMITPOSITIONS : TGROUPMEMS) : BOOLEAN;

  { True if LIMITPOSITIONS is empty.
    Called by Body of ELEMENT}

  BEGIN
  WITH LIMITPOSITIONS DO
     IF COMBINED THEN NOVARIABLESUBTN := (COMBMEMS=NIL)
```

```
                        ELSE NOVARIABLESUBTN := (MEMBERS=[])
END;



PROCEDURE ADDZERO(VAR GATEFREQUENCY : INTRECORD);

{ Adds zero to the integer range in GATEFREQUENCY.
  Called from body of ELEMENT}

VAR PTR : PDOUBLIST;

BEGIN
WITH GATEFREQUENCY DO
  IF TOPRANGE = 0
    THEN { zero already present - no action needed }
    ELSE IF TOPRANGE = 1
           THEN TOPRANGE := 0
           ELSE IF SUBRANGES = NIL
                  THEN SUBRANGES := ZEROFREQ
                  ELSE BEGIN
                         PTR := SUBRANGES;
                         WHILE PTR^.NEXT <> NIL DO PTR := PTR^.NEXT;
                         IF PTR^.FIRST = 0
                           THEN { zero already present - no action needed }
                           ELSE IF PTR^.FIRST = 1
                                  THEN PTR^.FIRST := 0
                                  ELSE PTR^.NEXT := ZEROFREQ
                       END
END;






{------------------------------------------------------------------------}
PROCEDURE GETLIMITPOSITIONS(PARALTLIST          : PPALTBARS;
                            VAR LIMITPOSITIONS : TGROUPMEMS);

```

```
{ Establishes LIMITPOSITIONS from the contents of PARALTLIST.
  Called by Body of ELEMENT}

VAR ALLGENERIC : BOOLEAN;

BEGIN
LIMITPOSITIONS.COMBINED := (PARALTLIST^.CONNBONDS.CONNECTIONS=2);
WITH PARALTLIST^.PRNTPOSNS^ DO
   IF LIMITPOSITIONS.COMBINED
      THEN IF COMBINED
              THEN LIMITPOSITIONS.COMBMEMS := COPYLIST(COMBMEMS)
              ELSE BEGIN
                    LIMITPOSITIONS.COMBMEMS := NIL;
                    LISTPOSNS(LIMITPOSITIONS.COMBMEMS, MEMBERS, MEMBERS, MEMBERS)
                   END
      ELSE IF COMBINED
              THEN PROGERROR(19) {combined position set with connectivity <> 2}
              ELSE LIMITPOSITIONS.MEMBERS := MEMBERS;
WITH PARALTLIST^ DO
   BEGIN
    IF PARSTRUCT = NIL
       THEN ALLGENERIC := TRUE
       ELSE ALLGENERIC := (PARSTRUCT^.PSVARIETY = GENERIC);
    PARALTLIST := NEXT
   END;
WHILE PARALTLIST <> NIL DO WITH PARALTLIST^ DO
   BEGIN
    IF ALLGENERIC AND (PARSTRUCT <> NIL)
       THEN ALLGENERIC := (PARSTRUCT^.PSVARIETY = GENERIC);
    IF LIMITPOSITIONS.COMBINED
       THEN REDUCE(LIMITPOSITIONS.COMBMEMS, PRNTPOSNS^)
       ELSE WITH PRNTPOSNS^ DO
               IF COMBINED
                  THEN PROGERROR(20) { combined position set with connectivity <> 2}
                  ELSE LIMITPOSITIONS.MEMBERS := LIMITPOSITIONS.MEMBERS * MEMBERS;
    PARALTLIST := NEXT
   END;
WITH LIMITPOSITIONS DO
   IF ALLGENERIC AND NOT COMBINED
      THEN MEMBERS := MEMBERS + [0]
```

```
END;
{----------------------------------------------------------------------------}



BEGIN   {Body of Procedure ELEMENT }
OPENERS     := [GLSQUARE, GOPENANG, GR, GQUEST, GSD, GPRIME, GLPAREN];
TERMINATORS := [GRPAREN, GAMPERSAND, GSB, GOSB, GSLASH, GSEMI, GELSE, GEND, GPERIOD, GAND, GOR];
NEXTTOKEN;
VALID := FALSE;
REPEAT
 DELIMCHECK := CHECKDELIM(OPENERS);
 IF (DELIMCHECK=INVALIDTOKEN) AND (TOKEN.NATURE<>NOMENCLATURE)
    THEN ERROR(18,0)
    ELSE VALID := TRUE
UNTIL VALID;

GATEPARENTPOSITIONS := NIL;
IF DELIMCHECK = GLSQUARE THEN
   BEGIN
    GETLIMITPOSITIONS(PARALTLIST, LIMITPOSITIONS);
    IF NOVARIABLESUBTN(LIMITPOSITIONS)
       THEN WRITELN('Common definition of variable-substitution positions not possible.')
       ELSE BEGIN
             NEW(GATEPARENTPOSITIONS);
             ECTRSIZE := ECTRSIZE + 9;
             POSITIONSET(GATEPARENTPOSITIONS^, LIMITPOSITIONS, PARALTLIST^.CONNBONDS.CONNECTIONS, 7);
             IF PARALTLIST^.CONNBONDS.CONNECTIONS = NOTSET
                THEN BEGIN
                      IF GATEPARENTPOSITIONS^.COMBINED
                         THEN SETCONNBONDS(PARALTLIST^.CONNBONDS, 2)
                         ELSE SETCONNBONDS(PARALTLIST^.CONNBONDS, 1);
                      UPDATEPARALTCONNS(PARALTLIST)
                     END;
             NEXTTOKEN
            END;
   VALID := FALSE;
   REPEAT
    DELIMCHECK := CHECKDELIM( OPENERS-[GLSQUARE]);
    IF (DELIMCHECK=INVALIDTOKEN) AND (TOKEN.NATURE <> NOMENCLATURE)
```

```
        THEN ERROR(24,0)
        ELSE VALID := TRUE
    UNTIL VALID
   END;

IF DELIMCHECK=GOPENANG
   THEN BEGIN
         SELECTOR(GATEFREQUENCY, [0..MAXVARS], 8);
         IF OPTIONALSUB THEN ADDZERO(GATEFREQUENCY);
         NEXTTOKEN;
         VALID := FALSE;
         REPEAT
          DELIMCHECK := CHECKDELIM( OPENERS - [GLSQUARE, GOPENANG]);
          IF (DELIMCHECK=INVALIDTOKEN) AND (TOKEN.NATURE <> NOMENCLATURE)
             THEN ERROR(24,0)
             ELSE VALID := TRUE
         UNTIL VALID
        END
   ELSE WITH GATEFREQUENCY DO
           BEGIN
            TOPRANGE := NOTSET;
            IF OPTIONALSUB THEN SUBRANGES := OPTFREQ
                           ELSE SUBRANGES := ESSENTFREQ
           END;

CASE DELIMCHECK OF

 GPRIME        : CONCATENATETERMS(GATEPS);

 GQUEST        : BEGIN
                  NEW(GATEPS, UNKNOWN);
                  ECTRSIZE := ECTRSIZE + 6;
                  WITH GATEPS^ DO
                     BEGIN
                      PSVARIETY := UNKNOWN;
                      VISITED := FALSE;
                      PARENTGATE := NIL;
                      CHILDGATE := NIL
                     END
                 END;
```

```

    GSD             : READSD(GATEPS, INPUTMODE=TERMINAL);

    INVALIDTOKEN    : TRANSLATENOMEN(GATEPS);

    GR              : BEGIN
                        SUBSTASVALUE(PARALTLIST);
                        GATEPS := NIL
                      END;

    GLPAREN         : BEGIN
                        ALTNVLIST(NEWPARENTPSLIST(PARALTLIST), FALSE);
                        WHILE CHECKDELIM( [GRPAREN]) = INVALIDTOKEN DO ERROR(14,0);
                        GATEPS := NIL
                      END

    END;  { of case }

  IF GATEPS <> NIL THEN SETCOMBARS(PARALTLIST, GATEPS);
  IF DELIMCHECK <> INVALIDTOKEN THEN NEXTTOKEN; { TRANSLATENOMEN has taken NEXTTOKEN }
  WHILE CHECKDELIM( [GLSQUARE]+TERMINATORS)=INVALIDTOKEN DO ERROR(24,0);
  IF TOKEN.DELIMVAL = GLSQUARE THEN MODIFYCHILDPOSITIONS(PARALTLIST);
  WHILE CHECKDELIM(TERMINATORS)=INVALIDTOKEN DO ERROR(24,0)
  END;
            { of Procedure ELEMENT
  ---------------------------------------------------------------------}



  {--------------------------------------------------------------------}
  PROCEDURE ALTNTVE(VAR PARALTLIST  : PPALTBARS;
                        OPTIONALSUB    : BOOLEAN);

  { Called once for each alternative in a definition expresssion
    and cycles round any number of levels of further substitution. Beyond the
    very first level, the bond type attachment is NOTSPECIFIED until the child
    structure itself is reached (in ELEMENT).
    At the end of each cycle, the former PARALTLIST
    is destroyed as a new one is created. The last one remaining is passed back
    (via the VAR parameter) to ALTNVLIST, where it is destroyed.
```

```
  Called by ALTNVLIST}

VAR DELIMCHECK    : DELIMTYPE;
    COMB          : PCOMBINLIST;
    NEWPARALTLIST,
    PTR           : PPALTBARS;



FUNCTION ALREADYINLIST(PTRPS      : PTRPSTYPE;
                       PARALTLIST : PPALTBARS) : BOOLEAN;

{ Returns TRUE if PTRPS is the PARSTRUCT field of any item in PARALTLIST.
  Called by ADDCOMBINPSS}

VAR PSFOUND : BOOLEAN;

BEGIN
PSFOUND := FALSE;
WHILE (PARALTLIST <> NIL) AND NOT PSFOUND DO WITH PARALTLIST^ DO
   BEGIN
    PSFOUND := PARSTRUCT = PTRPS;
    PARALTLIST := NEXT
   END;
ALREADYINLIST := PSFOUND
END;



PROCEDURE ADDPARALT(VAR NEWPARALTLIST : PPALTBARS;
                    NEWPARENT         : PTRPSTYPE);

{ Adds one new item to NEWPARALTLIST, setting the ALTBAR field to NIL (because,
  as SB and OSB have a higher precedence than /, there cannot be any
  alternatives for this level)
  Called by ADDCOMBINPSS}

VAR NEWPA : PPALTBARS;

BEGIN
```

```
NEW(NEWPA);
WITH NEWPA^ DO
   BEGIN
    PARSTRUCT := NEWPARENT;
    ALTBAR := NIL;
    CONNBONDS.CONNECTIONS := NOTSET;
    COPYCHILDPS := FALSE;
    NEW(PRNTPOSNS);
    ECTRSIZE := ECTRSIZE + 9;
    PRNTPOSNS^.COMBINED := FALSE;
    GETAVAILABLEPOSITIONS(PARSTRUCT, PRNTPOSNS^.MEMBERS, 1);
    NEXT := NEWPARALTLIST
   END;
NEWPARALTLIST := NEWPA
END;



PROCEDURE ADDCOMBINPSS(COMBIN            : PCOMBINLIST;
                       VAR NEWPARALTLIST : PPALTBARS);

{ Adds to NEWPARALTLIST the partial structures pointed to by the combination
  in COMBIN. If COMBIN is non-BOTTOMBAR then the procedure calls itself recursively
  for each of the ALTERNATIVES. A check is made (FUNCTION ALREADYINLIST) to
  prevent the same PS being included in the list more than once.
  If COMBIN is BOTTOMBAR, NEWPARALT is used to insert the  new item in the
  list.
  Called by Body of ALTNTVE}

VAR ALTERN     : PALTERNLIST;

BEGIN
IF COMBIN^.BOTTOMBAR
   THEN IF ALREADYINLIST(COMBIN^.CHILDPS, NEWPARALTLIST)
           THEN {don't duplicate}
           ELSE ADDPARALT(NEWPARALTLIST, COMBIN^.CHILDPS)
   ELSE BEGIN
        ALTERN := COMBIN^.ALTERNATIVES;
        REPEAT
         ADDCOMBINPSS(ALTERN^.COMBINATION, NEWPARALTLIST);
```

```
              ALTERN := ALTERN^.NEXT
             UNTIL ALTERN = NIL
            END
    END;



    BEGIN  {Body of Procedure ALTNTVE}
    REPEAT
     REPEAT ELEMENT(PARALTLIST, OPTIONALSUB)
     UNTIL CHECKDELIM([GAMPERSAND])=INVALIDTOKEN;
     DELIMCHECK := CHECKDELIM([GSB,GOSB]);
     IF DELIMCHECK <> INVALIDTOKEN THEN
        BEGIN
         NEWPARALTLIST := NIL;
         WHILE PARALTLIST <> NIL DO
            BEGIN
             IF PARALTLIST^.ALTBAR = NIL
                THEN COMB := PARALTLIST^.PARSTRUCT^.CHILDGATE
                ELSE COMB := PARALTLIST^.ALTBAR^.COMBINATION;
             WHILE COMB <> NIL DO
                BEGIN
                 ADDCOMBINPSS(COMB, NEWPARALTLIST);
                 COMB := COMB^.NEXT
                END;
             PTR := PARALTLIST^.NEXT;
             DISPOSE(PARALTLIST);
             PARALTLIST := PTR
            END;
         PARALTLIST := NEWPARALTLIST;
         OPTIONALSUB := DELIMCHECK = GOSB
        END
    UNTIL DELIMCHECK=INVALIDTOKEN
    END;
              { of Procedure ALTNTVE
    -----------------------------------------------------------------------}



    {---------------------------------------------------------------------}
```

```
FUNCTION PPOSNS(DUMMYCOMBIN : PCOMBINLIST;
                NEWPARENT   : PTRPSTYPE;
                DUMMYSUBST  : SUBSTITUENT) : PTGROUPMEMS;

{ Returns an appropriate parent positions set for NEWPARENT, after considering
  the positions available in it for the appropriate MAGNITUDE, and those
  positions specified for substitution in DUMMYCOMBIN. Outputs appropriate
  error messages if incompatibilities are detected.
  Called by ADDFURTHERSUBTN}

VAR AVAILPOSNS : ARRAY[1..3] OF INTEGSET;
    POSNS      : PTGROUPMEMS;
    LMAG       : TBONDMAG;



FUNCTION FINDBOTTOM(ALTERN       : PALTERNLIST;
                    FUNCTION MAG : TBONDMAG) : TBONDMAG;

{ Traces down all the aternatives in the list headed by ALTERN using the formal
  function MAG. Returns the largest bond magnitude.
  Called by LMAGNOCHECKS
            LMAGCHECKS}

VAR LBOND,
    MBOND : TBONDMAG;
    COMB  : PCOMBINLIST;

BEGIN
LBOND := 0;
WHILE ALTERN <> NIL DO
   BEGIN
    COMB := ALTERN^.COMBINATION;
    WHILE COMB <> NIL DO
       BEGIN
        MBOND := MAG(COMB);
        IF MBOND > LBOND THEN LBOND := MBOND;
        COMB := COMB^.NEXT
       END;
    ALTERN := ALTERN^.NEXT
```

```
   END;
FINDBOTTOM := LBOND
END;



FUNCTION LMAGNOCHECKS(COMBIN : PCOMBINLIST) : TBONDMAG;

{ Traces down COMBIN by calling itself recursively via FINDBOTOM.
  At the bottom bar, returns the largest bond MAGNITUDE.
  Called by LMAGCHECKS
            FINDBOTTOM (as formal parameter) }

VAR LMAG : TBONDMAG;

BEGIN
IF COMBIN^.BOTTOMBAR
   THEN WITH COMBIN^, CONNBONDS DO
          CASE CONNECTIONS OF
           NOTSET : LMAG := 1;
           0      : LMAG := 0;
           1      : LMAG := MAGNITUDE(BOND);
           2      : BEGIN
                     LMAG := MAGNITUDE(BONDA);
                     IF MAGNITUDE(BONDB) > LMAG
                        THEN LMAG := MAGNITUDE(BONDB)
                    END
           END
   ELSE LMAG := FINDBOTTOM(COMBIN^.ALTERNATIVES, LMAGNOCHECKS);
LMAGNOCHECKS := LMAG
END;



FUNCTION LMAGCHECKS(COMBIN : PCOMBINLIST) : TBONDMAG;

{ Traces down COMBIN, calling itself recursively via FINDBOTTOM,
  until it encounters a bar  containing a PARENTPOSITIONS field,
  or reaches the bottom of the gate. In either case, calls LMAGNOCHECKS
  to obtain the largest bond magnitude from the
```

```
      bottom of the gate, and ensures that the PARENTPOSITIONS field (if given) is
      a subset of the positions available for this magnitude. The value of the
      magnitude is returned.
      Called by PPOSNS
                FINDBOTTOM (as formal parameter)}

    VAR LMAG : TBONDMAG;

    BEGIN
    IF COMBIN^.PARENTPOSITIONS <> NIL
       THEN BEGIN
             LMAG := LMAGNOCHECKS(COMBIN);
             WITH COMBIN^, PARENTPOSITIONS^ DO
                IF COMBINED
                   THEN CHECKALLWITHIN(COMBMEMS, AVAILPOSNS[LMAG], 3)
                   ELSE IF MEMBERS <= AVAILPOSNS[LMAG]
                           THEN { specified positions are all available }
                           ELSE FAILURE(49, DUMMYSUBST, '    ')
            END
       ELSE IF COMBIN^.BOTTOMBAR
               THEN LMAG := LMAGNOCHECKS(COMBIN)
               ELSE LMAG := FINDBOTTOM(COMBIN^.ALTERNATIVES, LMAGCHECKS);
    LMAGCHECKS := LMAG
    END;



    BEGIN {Body of PPOSNS}
    FOR LMAG := 1 TO 3 DO GETAVAILABLEPOSITIONS(NEWPARENT, AVAILPOSNS[LMAG], LMAG);
    IF AVAILPOSNS[1] = []
       THEN FAILURE(50, DUMMYSUBST, '    ');
    WITH DUMMYCOMBIN^ DO
       IF PARENTPOSITIONS^.COMBINED
          THEN BEGIN
                LMAG := LMAGCHECKS(DUMMYCOMBIN);
                PPOSNS := PARENTPOSITIONS
               END
          ELSE IF PARENTPOSITIONS^.MEMBERS = [1..MAXCT]
                  THEN BEGIN
                        IF NOT DUMMYCOMBIN^.BOTTOMBAR
```

```
                              THEN LMAG := FINDBOTTOM(DUMMYCOMBIN^.ALTERNATIVES, LMAGCHECKS);
                          NEW(POSNS);
                          ECTRSIZE := ECTRSIZE + 9;
                          POSNS^.COMBINED := FALSE;
                          POSNS^.MEMBERS := AVAILPOSNS[LMAG];
                          PPOSNS := POSNS
                         END
                   ELSE BEGIN
                          LMAG := LMAGCHECKS(DUMMYCOMBIN);
                          PPOSNS := PARENTPOSITIONS
                         END
END;
                             { of PPOSNS
-----------------------------------------------------------------------}



PROCEDURE ADDFURTHERSUBTN(COMBINBAR : PCOMBINLIST;
                          DUMMYPS   : PTRPSTYPE);

{ Copies the further substitution on DUMMYPS onto the PSs at the bottom of
  COMBINBAR.
  Called by Body of ALTNVLIST}

VAR ALTERNBAR : PALTERNLIST;
    FSUBCOMB  : PCOMBINLIST;

BEGIN
WHILE COMBINBAR <> NIL DO WITH COMBINBAR^ DO
   BEGIN
    IF BOTTOMBAR
      THEN BEGIN
             FSUBCOMB := DUMMYPS^.CHILDGATE;
             REPEAT
              COPYCOMBAR({newcombar}     CHILDPS^.CHILDGATE,
                         {oldcombar}     FSUBCOMB,
                         {lastcomblayer} NIL,
                         {lastpposns}    PPOSNS(FSUBCOMB, CHILDPS, DUMMYPS^.SUBSTNAME),
                         {prntps}        CHILDPS,
                         {firstbar}      TRUE,
```

```
                                {omitpg}        DEFNTABLEENTRY(CHILDPS),
                                {copypss}       FALSE);
                      FSUBCOMB := FSUBCOMB^.NEXT
                     UNTIL FSUBCOMB = NIL
                    END
               ELSE BEGIN
                     ALTERNBAR := ALTERNATIVES;
                     REPEAT
                      ADDFURTHERSUBTN(ALTERNBAR^.COMBINATION, DUMMYPS);
                      ALTERNBAR := ALTERNBAR^.NEXT
                     UNTIL ALTERNBAR = NIL
                    END;
           COMBINBAR := NEXT
          END
      END;



      BEGIN { Body of Procedure ALTNVLIST }
      REPEAT
       PARALTLIST := NIL;
       READPTR := PARENTPSLIST;
       WHILE READPTR <> NIL DO
          BEGIN
           NEW(NEWALTERNATIVE);
           ECTRSIZE := ECTRSIZE + 4;
           NEWALTERNATIVE^.COMBINATION := NIL;
           NEWALTERNATIVE^.NEXT := READPTR^.COMBINS^.ALTERNATIVES;
           READPTR^.COMBINS^.ALTERNATIVES := NEWALTERNATIVE;

           NEW(WRITEPTR);
           WRITEPTR^.PARSTRUCT := READPTR^.PARSTRUCT;
           WRITEPTR^.CONNBONDS := READPTR^.CONNBONDS;
           WRITEPTR^.PRNTPOSNS := READPTR^.PRNTPOSNS;
           WRITEPTR^.ALTBAR    := NEWALTERNATIVE;
           IF READPTR^.COPYCHILDPS
              THEN WRITEPTR^.COPYCHILDPS := TRUE
              ELSE IF READPTR^.FURTHERSUB = NIL
                      THEN WRITEPTR^.COPYCHILDPS := FALSE
                      ELSE WRITEPTR^.COPYCHILDPS := READPTR^.FURTHERSUB^.CHILDGATE <> NIL;
```

```
          WRITEPTR^.NEXT := PARALTLIST;
          PARALTLIST := WRITEPTR;

          READPTR := READPTR^.NEXT
         END;
      ALTNTVE(PARALTLIST, OPTIONALSUB);
      WHILE PARALTLIST <> NIL DO
        BEGIN
         WRITEPTR := PARALTLIST^.NEXT;
         DISPOSE(PARALTLIST);
         PARALTLIST := WRITEPTR
        END;

     READPTR := PARENTPSLIST;
     WHILE READPTR <> NIL DO WITH READPTR^ DO
        BEGIN
         IF FURTHERSUB <> NIL
            THEN IF FURTHERSUB^.CHILDGATE <> NIL
                    THEN ADDFURTHERSUBTN(COMBINS^.ALTERNATIVES^.COMBINATION, FURTHERSUB);
         READPTR := NEXT
        END

     UNTIL CHECKDELIM([GSLASH])=INVALIDTOKEN;
     WHILE PARENTPSLIST <> NIL DO
        BEGIN
         READPTR := PARENTPSLIST^.NEXT;
         DISPOSE(PARENTPSLIST);
         PARENTPSLIST := READPTR
        END
     END;

                              { OF PROCEDURE ALTNVLIST
     ******************************************************************************}






     {******************************************************************************
```

```
                              PROCEDURE ASSIGNMENTSTMNT

*********************************************************************************}
PROCEDURE ASSIGNMENTSTMNT;

 { Analyses an assignment statement.
   Called by STATEMENT}

VAR SELECTEDFREQ : INTRECORD;



FUNCTION ASSGNTOP : TSELECTMODE;

{ Returns the value of an assignment operator.
  Called by SUBSTASSIGNMENT
            MULTASSIGNMENT}

BEGIN
WHILE CHECKDELIM([GDEQ,GSEQ,GHASHEQ,GDOLEQ,GEQUALS]) = INVALIDTOKEN DO ERROR(32,0);
CASE TOKEN.DELIMVAL OF
 GEQUALS : ASSGNTOP := INDEPENDENT;
 GSEQ    : ASSGNTOP := ALLSAME;
 GDEQ    : ASSGNTOP := ALLDIFF;
 GDOLEQ  : ASSGNTOP := NOTALLSAME;
 GHASHEQ : ASSGNTOP := NOTALLDIFF
 END
END;



{----------------------------------------------------------------------------}
PROCEDURE SUBSTGROUP(VAR GROUPMEMS : TGROUPMEMS;
                     LIMITSET      : INTEGSET;
                     ERRORCODE     : INTEGER);

{ Analyses a substituent group, all members of which must fall in LIMITSET.
  Called by SUBSTASSIGNMENT}
```

```
LABEL 10;

VAR TERMINATORS  : DELIMSET;
    CONNECTIVITY : TCONNS;



PROCEDURE REVISELIMITS(VAR LIMITSET : INTEGSET;
                       CONNECTIVITY : TCONNS);

{ Removes those elements in LIMITSET for which the connectivity shown in
  RDECLARATIONTABLE is not compatible with CONNECTIVITY}

VAR SUBST : SUBSTITUENT;

BEGIN
FOR SUBST := 1 TO MAXVARS DO
   IF SUBST IN LIMITSET
      THEN WITH RDECLARATIONTABLE[SUBST]^.CONNBONDS DO
              IF (CONNECTIONS=CONNECTIVITY) OR (CONNECTIONS=NOTSET)
                 THEN {would be compatible}
                 ELSE LIMITSET := LIMITSET - [SUBST]
END;



PROCEDURE CHECKCOMPATABILITY(GROUP        : INTEGSET;
                             CONNECTIVITY : TCONNS);

{ Checks that the substituents in GROUP have compatible CONNECTIVITY }

VAR SUBST : SUBSTITUENT;

BEGIN
IF CONNECTIVITY = NOTSET
   THEN FOR SUBST := 1 TO MAXVARS DO
           IF SUBST IN GROUP
              THEN WITH RDECLARATIONTABLE[SUBST]^, CONNBONDS DO
                      IF CONNECTIONS <> NOTSET
                         THEN CONNECTIVITY := CONNECTIONS;
```

```
       IF CONNECTIVITY = NOTSET
          THEN {still no information on connectivity. No compatability checking possible}
          ELSE FOR SUBST := 1 TO MAXVARS DO
                 IF SUBST IN GROUP
                    THEN WITH RDECLARATIONTABLE[SUBST]^ DO
                            IF CONNBONDS.CONNECTIONS = NOTSET
                               THEN BEGIN
                                      SETCONNBONDS(CONNBONDS, CONNECTIVITY);
                                      UPDATEPPSCONNS(RDECLARATIONTABLE[SUBST])
                                    END
                               ELSE IF CONNBONDS.CONNECTIONS = CONNECTIVITY
                                       THEN {compatible}
                                       ELSE FAILURE(51, 0, '     ')
       END;



       FUNCTION LISTSMATCH(SUBSTA, SUBSTB : SUBSTITUENT) : BOOLEAN;

       { Checks that the declarations for SUBSTA and SUBSTB all refer to the same PSs }

       VAR ADECNS,
           BDECNS  : PPSLIST;
           FOUND   : BOOLEAN;

       BEGIN
       ADECNS := RDECLARATIONTABLE[SUBSTA];
       REPEAT
          BDECNS := RDECLARATIONTABLE[SUBSTB];
          REPEAT
             FOUND := (ADECNS^.PARSTRUCT = BDECNS^.PARSTRUCT) AND (ADECNS <> BDECNS);
             BDECNS := BDECNS^.NEXT
          UNTIL FOUND OR (BDECNS = NIL);
          IF FOUND
             THEN ADECNS := ADECNS^.NEXT
             ELSE BEGIN
                    ERROR(37,0);
                    ADECNS := NIL
                  END
          UNTIL ADECNS = NIL;
```

```
LISTSMATCH := FOUND
END;



PROCEDURE SUBSTCOMBINATION(VAR COMBMEMS : PDOUBLIST);

 { Analyses a substituent combination }

VAR SUBCOMB : PDOUBLIST;

BEGIN
NEW(SUBCOMB);
NEXTTOKEN;
WHILE CHECKDELIM([GR]) = INVALIDTOKEN DO ERROR(26,0);
NEXTTOKEN;
CHECKVALIDINT(LIMITSET,35);
SUBCOMB^.FIRST := TOKEN.INTEGVAL;
NEXTTOKEN;
WHILE CHECKDELIM([GPLUS]) = INVALIDTOKEN DO ERROR(36, 0);
NEXTTOKEN;
WHILE CHECKDELIM([GR]) = INVALIDTOKEN DO ERROR(26, 0);
NEXTTOKEN;
WITH SUBCOMB^ DO
   REPEAT
    CHECKVALIDINT(LIMITSET, 35);
    SECOND := TOKEN.INTEGVAL
   UNTIL LISTSMATCH(FIRST, SECOND) AND LISTSMATCH(SECOND, FIRST);
NEXTTOKEN;
WHILE CHECKDELIM([GCOMMA] + TERMINATORS) = INVALIDTOKEN DO ERROR(24, 0);
SUBCOMB^.NEXT := COMBMEMS;
COMBMEMS := SUBCOMB
END;



BEGIN  { Body of Procedure SUBSTGROUP }
TERMINATORS := [GNOTEQ,GEQUALS,GDEQ, GSEQ, GDOLEQ, GHASHEQ, GPERIOD];
LOOKAHEAD;
CHECKVALIDINT(LIMITSET, ERRORCODE);
```

```
CONNECTIVITY := RDECLARATIONTABLE[TOKEN.INTEGVAL]^.CONNBONDS.CONNECTIONS;
LOOKAHEAD;
10:
WHILE CHECKDELIM([GPLUS, GCOMMA, GHYPHEN]+TERMINATORS)=INVALIDTOKEN DO ERROR(24,0);
GROUPMEMS.COMBINED := TOKEN.DELIMVAL = GPLUS;
IF GROUPMEMS.COMBINED
   THEN CASE CONNECTIVITY OF
          0, 2    : BEGIN
                     ERROR(30,0);
                     GOTO 10
                    END;
          NOTSET,
          1       : BEGIN
                     REVISELIMITS(LIMITSET, 1);
                     GROUPMEMS.COMBMEMS := NIL;
                     REPEAT SUBSTCOMBINATION(GROUPMEMS.COMBMEMS)
                     UNTIL CHECKDELIM([GCOMMA])=INVALIDTOKEN
                    END
         END
   ELSE BEGIN
          IF CONNECTIVITY <> NOTSET THEN REVISELIMITS(LIMITSET, CONNECTIVITY);
          NEXTTOKEN;
          GROUPRANGE(GROUPMEMS.MEMBERS, LIMITSET, ERRORCODE);
          CHECKCOMPATABILITY(GROUPMEMS.MEMBERS, CONNECTIVITY)
         END;
WHILE CHECKDELIM(TERMINATORS)=INVALIDTOKEN DO ERROR(24,0)
END;

            { of Procedure SUBSTGROUP
--------------------------------------------------------------------}



{-------------------------------------------------------------------}
FUNCTION POINTERLIST(GROUPMEMS : TGROUPMEMS) : PPSLIST;

VAR READPTR,
    WRITEPTR,
    LISTPTR  : PPSLIST;
    SUBST    : SUBSTITUENT;
```

```
{ Sets up a linked list of declarations for use in gate-setting.
  One element in the  list represents one declaration of one substituent,
  so the same parent may  appear several times in the list.
  The list is built up from the entries in  RDECLARATIONTABLE.
  Called by SUBSTASSIGNMENT}



PROCEDURE GETBINFO(VAR BONDB  : BONDORDER;
                   VAR POSNSB : INTEGSET;
                   BPTR       : PPSLIST;
                   PARENTPS   : PTRPSTYPE);

{ Obtains bond order and positions from the second substituent of a combination.
  They are compiled from the information given in all the items in the BPTR
  list which reference PARENTPS.
  Called by ADDCOMBSUBS}

VAR FAILSTRING : STRING4;

BEGIN
BONDB := NOTSPECIFIED;
POSNSB := [];
WHILE BPTR <> NIL DO WITH BPTR^ DO
   BEGIN
    IF PARSTRUCT = PARENTPS THEN
       BEGIN
        IF PRNTPOSNS^.COMBINED
           THEN PROGERROR(22) {COMBINED position set in combined substituent}
           ELSE POSNSB := POSNSB + PRNTPOSNS^.MEMBERS;
        CASE CONNBONDS.CONNECTIONS OF
         NOTSET : ;
         0, 2   : PROGERROR(26); {Only connectivity of 1 permitted in combined substituents}
         1      : IF CONNBONDS.BOND <> NOTSPECIFIED
                    THEN BEGIN
                           BONDB := BONDMATCHARRAY[CONNBONDS.BOND, BONDB];
                           IF BONDB = NOTSPECIFIED
                              THEN BEGIN
                                     FAILSTRING[1] := BONDSTRING[CONNBONDS.BOND, 1];
```

```
                                          FAILSTRING[2] := BONDSTRING[CONNBONDS.BOND, 2];
                                          FAILSTRING[3] := BONDSTRING[BONDB, 1];
                                          FAILSTRING[4] := BONDSTRING[BONDB, 2];
                                          FAILURE(42, 0, FAILSTRING)
                                         END
                                  END
                END
              END;
           BPTR := NEXT
          END;
       IF POSNSB = [] THEN PROGERROR(25)
       END;


       PROCEDURE ADDCOMBSUBS(APTR,
                             BPTR        : PPSLIST;
                             VAR LISTPTR : PPSLIST);

       { Adds a substituent combination to the PPSLIST.
         Called by Body of POINTERLIST}

       VAR WRITEPTR : PPSLIST;
           POSNSA,
           POSNSB,
           POSNSC   : INTEGSET;
           MAGSUM   : INTEGER;

       BEGIN
       NEW(WRITEPTR);
       WITH WRITEPTR^ DO
          BEGIN
           PARSTRUCT := APTR^.PARSTRUCT;
           FURTHERSUB := NIL;
           WITH CONNBONDS DO
              BEGIN
               CONNECTIONS := 2;
               CASE APTR^.CONNBONDS.CONNECTIONS OF
                NOTSET : BONDA := NOTSPECIFIED;
                0, 2   : PROGERROR(23); {connectivity must be 1 for combined substituents}
                1      : BONDA := APTR^.CONNBONDS.BOND
```

```
              END;
             GETBINFO(BONDB, POSNSB, BPTR, PARSTRUCT)
            END;
        WITH APTR^.PRNTPOSNS^ DO
           IF COMBINED
              THEN PROGERROR(24) {COMBINED position set with combined substituents}
              ELSE POSNSA := MEMBERS;
        IF POSNSA * POSNSB = []
           THEN POSNSC := []
           ELSE BEGIN
                 WITH CONNBONDS DO MAGSUM := MAGNITUDE(BONDA) + MAGNITUDE(BONDB);
                 IF MAGSUM <= 3
                    THEN GETAVAILABLEPOSITIONS(PARSTRUCT, POSNSC, MAGSUM)
                    ELSE POSNSC := [];
                 POSNSC := POSNSA * POSNSB * POSNSC
                END;
        NEW(PRNTPOSNS);
        ECTRSIZE := ECTRSIZE + 9;
        WITH PRNTPOSNS^ DO
           BEGIN
            COMBINED := TRUE;
            COMBMEMS := NIL;
            LISTPOSNS(COMBMEMS, POSNSA, POSNSB, POSNSC)
           END;
        NEW(COMBINS, TRUE);
        ECTRSIZE := ECTRSIZE + 11;
        WITH COMBINS^ DO
           BEGIN
            PARENTPOSITIONS := PRNTPOSNS;
            FREQUENCY.TOPRANGE := NOTSET;
            FREQUENCY.SUBRANGES := ESSENTFREQ;
            BOTTOMBAR := FALSE;
            ALTERNATIVES := NIL;
            NEXT := PARSTRUCT^.CHILDGATE
           END;
        PARSTRUCT^.CHILDGATE := WRITEPTR^.COMBINS;
        COPYCHILDPS := FALSE;
        NEXT := LISTPTR
       END;
    LISTPTR := WRITEPTR
```

```
   END;



   PROCEDURE ADDDEFNTABLE(VAR LISTPTR : PPSLIST;
                              SUBST       : SUBSTITUENT);

   { Adds the RDEFINITIONTABLE for SUBST to the PPSLIST in LISTPTR. If the SUBST
     has already been defined then the COMBINS is taken from RDEFINITIONTABLE,
     otherwise a new one is created and entered into RDEFINITIONTABLE.
     Called by Body of POINTERLIST}

   VAR WRITEPTR : PPSLIST;

   BEGIN
   NEW(WRITEPTR);
   WITH WRITEPTR^ DO
      BEGIN
       PARSTRUCT := NIL;
       FURTHERSUB := NIL;
       NEW(PRNTPOSNS);
       ECTRSIZE := ECTRSIZE + 9;
       PRNTPOSNS^.COMBINED := FALSE;
       PRNTPOSNS^.MEMBERS := [1..MAXCT];
       SETCONNBONDS(CONNBONDS, LISTPTR^.CONNBONDS.CONNECTIONS);
       COPYCHILDPS := FALSE;
       NEXT := LISTPTR;
       IF RDEFINITIONTABLE[SUBST] = NIL
          THEN BEGIN
                NEW(COMBINS, TRUE);
                WITH COMBINS^ DO
                   BEGIN
                    BOTTOMBAR := FALSE;
                    ALTERNATIVES := NIL;
                    NEXT := NIL;
                    PARENTPOSITIONS := PRNTPOSNS;
                    FREQUENCY.TOPRANGE := NOTSET;
                    FREQUENCY.SUBRANGES := NIL
                   END;
                RDEFINITIONTABLE[SUBST] := COMBINS
```

```
                END
            ELSE COMBINS := RDEFINITIONTABLE[SUBST]
         END;
      LISTPTR := WRITEPTR
      END;



      BEGIN {Body of POINTERLIST}
      LISTPTR := NIL;
      IF GROUPMEMS.COMBINED
         THEN WHILE GROUPMEMS.COMBMEMS <> NIL DO WITH GROUPMEMS DO
                BEGIN
                 READPTR := RDECLARATIONTABLE[COMBMEMS^.FIRST];
                 WHILE READPTR <> NIL DO
                    BEGIN
                     ADDCOMBSUBS(READPTR, RDECLARATIONTABLE[COMBMEMS^.SECOND], LISTPTR);
                     READPTR := READPTR^.NEXT
                    END;
                 COMBMEMS := COMBMEMS^.NEXT
                END
         ELSE FOR SUBST := 1 TO MAXVARS DO IF SUBST IN GROUPMEMS.MEMBERS
                THEN BEGIN
                      READPTR := RDECLARATIONTABLE[SUBST];
                      WHILE READPTR <> NIL DO
                         BEGIN
                          NEW(WRITEPTR);
                          WRITEPTR^ := READPTR^;
                          IF WRITEPTR^.COMBINS = NIL
                             THEN PROGERROR(21);  {Declaration without combination bar}
                          WRITEPTR^.NEXT := LISTPTR;
                          LISTPTR := WRITEPTR;
                          READPTR := READPTR^.NEXT
                         END;
                      ADDDEFNTABLE(LISTPTR, SUBST)
                     END;
      POINTERLIST := LISTPTR
      END;
            {of FUNCTION POINTERLIST
      --------------------------------------------------------------------------------}
```

```
PROCEDURE SUBSTASSIGNMENT;

{ Analyses a substituent assignment.
  Called by body of ASSIGNMENTSTMNT}

VAR GROUPMEMS      : TGROUPMEMS;
    PARENTPSLIST   : PPSLIST;
    DELPTR         : PDOUBLIST;

BEGIN
SUBSTGROUP(GROUPMEMS, DECLSUBS, 1);
IF ASSGNTOP <> INDEPENDENT THEN
   WRITELN('Non-independent assignment not yet implemented');
PARENTPSLIST := POINTERLIST(GROUPMEMS);
ALTNVLIST(PARENTPSLIST, FALSE);
WITH GROUPMEMS DO IF COMBINED
   THEN WHILE COMBMEMS <> NIL DO
          BEGIN
           DEFNSUBS := DEFNSUBS + [COMBMEMS^.FIRST, COMBMEMS^.SECOND];
           DELPTR := COMBMEMS;
           COMBMEMS := COMBMEMS^.NEXT;
           DISPOSE(DELPTR)
          END
   ELSE DEFNSUBS := DEFNSUBS + MEMBERS
END;



{---------------------------------------------------------------------------}
PROCEDURE MULTASSIGNMENT;

{ Analyses a multiplier assignment.
  Called by Body of ASSIGNMENTSTMNT}

VAR MULT         : MULTIPLIER;
    DEFINEDMULTS : INTEGSET;
    MULTVALUES,
```

```
    MULTVALCOPY  : INTRECORD;



PROCEDURE COPYLIST(MULTVALUES      : INTRECORD;
                   VAR MULTVALCOPY : INTRECORD);

{ Copies a list of values for a multiplier.
  Called by body of MULTASSIGNMENT}

VAR NEWITEM,
    LASTITEM : PDOUBLIST;

BEGIN
MULTVALCOPY.TOPRANGE := MULTVALUES.TOPRANGE;
MULTVALCOPY.SUBRANGES := NIL;
WHILE MULTVALUES.SUBRANGES <> NIL DO
   BEGIN
    NEW(NEWITEM);
    ECTRSIZE := ECTRSIZE + 6;
    WITH NEWITEM^ DO
       BEGIN
        FIRST := MULTVALUES.SUBRANGES^.FIRST;
        SECOND := MULTVALUES.SUBRANGES^.SECOND;
        NEXT := NIL
       END;
    IF MULTVALCOPY.SUBRANGES = NIL
       THEN MULTVALCOPY.SUBRANGES := NEWITEM
       ELSE LASTITEM^.NEXT := NEWITEM;
    LASTITEM := NEWITEM;
    MULTVALUES.SUBRANGES := MULTVALUES.SUBRANGES^.NEXT
   END
END;



PROCEDURE ADDITEM(VAR NEWITEM,
                      NEWLIST  : PDOUBLIST);

{ Inserts NEWITEM into NEWLIST.
```

```
  Called by COMBINEVALUES}

VAR NEWLISTITEM : PDOUBLIST;

BEGIN
NEWLISTITEM := NEWITEM;
NEWITEM := NEWITEM^.NEXT;
NEWLISTITEM^.NEXT := NEWLIST;
NEWLIST := NEWLISTITEM
END;



PROCEDURE COMBINEVALUES(VAR TABLEVALUES : INTRECORD;
                            NEWVALUES   : INTRECORD);

{ Combines the NEWVALUES just obtained with those already in TABLEVALUES.
  Called by Body of MULTASSIGNMENT }

VAR NEWLIST,
    NEWITEM  : PDOUBLIST;
    FINISHED : BOOLEAN;

BEGIN
IF (TABLEVALUES.TOPRANGE) = NOTSET
   THEN TABLEVALUES.TOPRANGE := NEWVALUES.TOPRANGE
   ELSE IF NEWVALUES.TOPRANGE = NOTSET
           THEN { leave TABLEVALUES.TOPRANGE as it is }
           ELSE IF NEWVALUES.TOPRANGE < TABLEVALUES.TOPRANGE
                   THEN TABLEVALUES.TOPRANGE := NEWVALUES.TOPRANGE;
NEWLIST := NIL;
WHILE NOT ((TABLEVALUES.SUBRANGES = NIL) AND (NEWVALUES.SUBRANGES = NIL)) DO
   IF TABLEVALUES.SUBRANGES = NIL
      THEN ADDITEM(NEWVALUES.SUBRANGES, NEWLIST)
      ELSE IF NEWVALUES.SUBRANGES = NIL
              THEN ADDITEM(TABLEVALUES.SUBRANGES, NEWLIST)
              ELSE IF TABLEVALUES.SUBRANGES^.FIRST > NEWVALUES.SUBRANGES^.FIRST
                      THEN ADDITEM(TABLEVALUES.SUBRANGES, NEWLIST)
                      ELSE ADDITEM(NEWVALUES.SUBRANGES, NEWLIST);
IF NEWLIST <> NIL
```

```
         THEN BEGIN
                TABLEVALUES.SUBRANGES := NEWLIST;
                NEWLIST := NEWLIST^.NEXT;
                TABLEVALUES.SUBRANGES^.NEXT := NIL
               END
         ELSE TABLEVALUES.SUBRANGES := NIL;
      WHILE NEWLIST <> NIL DO
         BEGIN
          NEWITEM := NEWLIST;
          NEWLIST := NEWLIST^.NEXT;
          IF NEWITEM^.FIRST > TABLEVALUES.SUBRANGES^.SECOND + 1
             THEN BEGIN
                   NEWITEM^.NEXT := TABLEVALUES.SUBRANGES;
                   TABLEVALUES.SUBRANGES := NEWITEM
                  END
             ELSE BEGIN
                   IF NEWITEM^.SECOND > TABLEVALUES.SUBRANGES^.SECOND
                      THEN TABLEVALUES.SUBRANGES^.SECOND := NEWITEM^.SECOND;
                   DISPOSE(NEWITEM);
                  ECTRSIZE := ECTRSIZE - 6
                  END
         END;
      FINISHED := (TABLEVALUES.SUBRANGES = NIL) OR (TABLEVALUES.TOPRANGE = NOTSET);
      WHILE NOT FINISHED DO WITH TABLEVALUES DO
         IF SUBRANGES^.SECOND >= TOPRANGE - 1
            THEN BEGIN
                  IF SUBRANGES^.FIRST < TOPRANGE
                     THEN TOPRANGE := SUBRANGES^.FIRST;
                  NEWITEM := SUBRANGES;
                  SUBRANGES := SUBRANGES^.NEXT;
                  DISPOSE(NEWITEM);
                  ECTRSIZE := ECTRSIZE - 6;
                  FINISHED := (SUBRANGES = NIL)
                 END
            ELSE FINISHED := TRUE
      END;



      BEGIN  {Body of MULTASSIGNMENT}
```

```
GROUPRANGE(DEFINEDMULTS, DECLMULT, 2);
WHILE CHECKDELIM([GDEQ,GSEQ,GHASHEQ,GDOLEQ,GNOTEQ,GEQUALS,GPERIOD]) = INVALIDTOKEN DO ERROR(24,0);
IF ASSGNTOP <> INDEPENDENT
   THEN WRITELN('Non-independent assignment not yet implemented');
NEXTTOKEN;
SELECTOR(MULTVALUES,[0..MAXVARS],10);
NEXTTOKEN;
FOR MULT := 1 TO MAXVARS DO IF MULT IN DEFINEDMULTS THEN
   BEGIN
    COPYLIST(MULTVALUES, MULTVALCOPY);
    IF MULT IN DEFNMULT
       THEN COMBINEVALUES(MDEFINITIONTABLE[MULT], MULTVALCOPY)
       ELSE MDEFINITIONTABLE[MULT] := MULTVALCOPY
   END;
REDUCEECTR(MULTVALUES.SUBRANGES);
DESTROY(MULTVALUES.SUBRANGES);
DEFNMULT := DEFNMULT + DEFINEDMULTS
END;             {of MULTASSIGNMENT
----------------------------------------------------------------------}



BEGIN { Body of ASSIGNMENTSTMNT }
IF CHECKDELIM([GR,GM])=INVALIDTOKEN THEN SELECTOR(SELECTEDFREQ, [0..MAXVARS],9);
WHILE CHECKDELIM([GR,GM]) = INVALIDTOKEN DO ERROR(20,0);
IF TOKEN.DELIMVAL = GR
   THEN SUBSTASSIGNMENT
   ELSE MULTASSIGNMENT
END;
     {of PROCEDURE ASSIGNMENTSTMNT
**********************************************************************}




PROCEDURE CONDITION;

{ Analyses a condition.
  Called by IFSTATEMENT
            RESTRICTSTMNT}
```

```
         BEGIN
         CONDITIONSPRESENT := TRUE;
         REPEAT NEXTTOKEN
         UNTIL CHECKDELIM([GTHEN,GEND,GELSE,GSEMI,GPERIOD]) <> INVALIDTOKEN
         END;



         PROCEDURE RESTRICTSTMNT;

         { Analyses a RESTRICT statement.
           Called by STATEMENT}

         BEGIN
         CONDITION;
         WHILE CHECKDELIM([GELSE,GEND,GSEMI,GPERIOD]) = INVALIDTOKEN DO ERROR(24,0)
         END;



         PROCEDURE STATEMENT;
         FORWARD;



         PROCEDURE IFSTATEMENT;

         { Analyses an IF statement.
           Called by STATEMENT}

         BEGIN
         CONDITION;
         WHILE CHECKDELIM ([GTHEN]) <> GTHEN DO ERROR(17,0);
         STATEMENT;
         WHILE CHECKDELIM ([GELSE,GEND,GSEMI,GPERIOD]) = INVALIDTOKEN DO ERROR(24,0);
         IF CHECKDELIM([GELSE]) = GELSE THEN
            BEGIN
             STATEMENT;
             WHILE CHECKDELIM ([GELSE,GEND,GSEMI,GPERIOD]) = INVALIDTOKEN DO ERROR(24,0)
```

```
   END
 END;



 PROCEDURE CMPDSTMNT;

 { Analyses a compound statement.
   Called by STATEMENT}

 BEGIN
 REPEAT
  STATEMENT;
  WHILE CHECKDELIM([GSEMI,GEND]) = INVALIDTOKEN DO ERROR(15,0)
 UNTIL TOKEN.DELIMVAL = GEND;
 NEXTTOKEN
 END;



 PROCEDURE STATEMENT;

 { Analyses a STATEMENT, calling the appropriate procedure.
   Called by IFSTATEMENT
             RESTRICTSTMNT
             CMPDSTMNT
             Body of INTERPRET}

 VAR DELIMCHECK : DELIMTYPE;

 BEGIN
 NEXTTOKEN;
 REPEAT
  DELIMCHECK := CHECKDELIM([GBEGIN,GIF,GRESTRICT,GR,GM,GOPENANG,GEND,GSEMI,GPERIOD]);
  IF DELIMCHECK=INVALIDTOKEN THEN ERROR(19,0)
 UNTIL DELIMCHECK<>INVALIDTOKEN;
 CASE DELIMCHECK OF
   GR, GM,
   GOPENANG,
   INVALIDTOKEN  : ASSIGNMENTSTMNT;
```

```
    GBEGIN          : CMPDSTMNT;
    GIF             : IFSTATEMENT;
    GRESTRICT       : RESTRICTSTMNT;
    GPERIOD,
    GEND, GSEMI     : (* empty statement *)
   END
  END;



  PROCEDURE CHECKALLDONE;

  { Checks that all declared substituents and multipliers have been defined
    Called by Body of INTERPRET}

  VAR M          : 1..MAXVARS;

  BEGIN
  IF (DECLSUBS - DEFNSUBS) <> [] THEN
     BEGIN
      WRITELN('The following substituents',' remain undefined:');
      FOR M := 1 TO MAXVARS DO
          IF M IN (DECLSUBS-DEFNSUBS) THEN WRITE('    R', M:1);
      WRITELN;
      TOKEN.DELIMVAL := INVALIDTOKEN
     END;
  IF (DECLMULT - DEFNMULT) <> [] THEN
     BEGIN
      WRITELN('The following multipliers',' remain undefined:');
      FOR M := 1 TO MAXVARS DO
          IF M IN (DECLMULT-DEFNMULT) THEN WRITE('    M', M:1);
      WRITELN;
      TOKEN.DELIMVAL := INVALIDTOKEN
     END;
  IF TOKEN.DELIMVAL = INVALIDTOKEN THEN
     WHILE CHECKDELIM([GSEMI])=INVALIDTOKEN DO ERROR(13,0)
  END;
```

```
PROCEDURE RECORDMULTS;

{ Adds the values for multipliers to the appropriate FREQUENCY fields in the ECTR.
  Called by Body of INTERPRET}

VAR MULT  : MULTIPLIER;
    PMPTR : PMDECLIST;

BEGIN
FOR MULT := 1 TO MAXVARS DO IF MULT IN DEFNMULT THEN
   BEGIN
    PMPTR := MDECLARATIONTABLE[MULT];
    REPEAT
     PMPTR^.SUBSTDECN^.COMBINS^.FREQUENCY := MDEFINITIONTABLE[MULT];
     PMPTR := PMPTR^.NEXT
    UNTIL PMPTR = NIL
   END
END;



{****************************************************************************}
PROCEDURE OUTINTREP;

{ Outputs a reprsentation of the ECTR to a diagnostics file }

VAR SUBST   : SUBSTITUENT;
    MULT    : MULTIPLIER;
    READPTR : PIRLIST;



FUNCTION PSNO(PTRPS : PTRPSTYPE) : INTEGER;

VAR PTR   : PIRLIST;
    NUM   : INTEGER;
    FOUND : BOOLEAN;

BEGIN
NUM := 0;
```

```
PTR := INTERNALREP.PSLIST;
FOUND := FALSE;
WHILE (PTR<>NIL) AND NOT FOUND DO
   BEGIN
    NUM := NUM + 1;
    IF PTR^.PARSTRUCT = PTRPS
       THEN FOUND := TRUE
       ELSE PTR := PTR^.NEXT
   END;
IF FOUND THEN PSNO := NUM
         ELSE PSNO := 0
END;



PROCEDURE WRITEFREQ(FREQUENCY : INTRECORD);

VAR PTR : PDOUBLIST;

BEGIN
WRITE(DIAGFILE, '<');
WITH  FREQUENCY DO
   BEGIN
    IF TOPRANGE <> NOTSET THEN WRITE(DIAGFILE, TOPRANGE:1, '-,');
    PTR := SUBRANGES;
    WHILE PTR <> NIL DO WITH PTR^ DO
       BEGIN
        WRITE(DIAGFILE, FIRST:1, '-', SECOND:1, ',');
        PTR := NEXT
       END
   END;
WRITE(DIAGFILE, '>')
END;



PROCEDURE WRITEPOSNS(POSNSET : TGROUPMEMS);

VAR POSN : ATOMNUMBER;
    PTR  : PDOUBLIST;
```

```

BEGIN
WRITE(DIAGFILE, '[');
IF POSNSET.COMBINED
   THEN BEGIN
         PTR := POSNSET.COMBMEMS;
         WHILE PTR <> NIL DO WITH PTR^ DO
            BEGIN
             WRITE(DIAGFILE, FIRST:1, '/', SECOND:1, ',');
             PTR := NEXT;
            END
        END
   ELSE FOR POSN := 1 TO MAXCT DO IF POSN IN POSNSET.MEMBERS
            THEN WRITE(DIAGFILE, POSN:1, ',');
WRITE(DIAGFILE, ']')
END;



PROCEDURE WRITECONNS(CONNBONDS : TCONNBONDS);

{ Writes out the bond orders in CONNBONDS.
  Called by WRITEPGS
            WRITECOMBIN
            WRITEDECN}

BEGIN
WITH CONNBONDS DO
 CASE CONNECTIONS OF
  NOTSET : WRITELN(DIAGFILE, '--');
  0      : ;
  1      : WRITELN(DIAGFILE, BONDSTRING[BOND]);
  2      : BEGIN
            WRITE(DIAGFILE, BONDSTRING[BONDA]);
            WRITELN(DIAGFILE, BONDSTRING[BONDB])
           END
 END
END;


```

```
PROCEDURE WRITEPGS(PARENTS : PPARENTLIST);

{ Writes a series of Parent Gates, headed by PARENTS, to DIAGFILE }


BEGIN
WRITELN(DIAGFILE);
WRITELN(DIAGFILE, 'PARENT GATES: ');
WHILE PARENTS <> NIL DO WITH PARENTS^ DO
   BEGIN
    WRITEPOSNS(CHILDPOSITIONS);
    WRITE(DIAGFILE, ' PS:', PSNO(PARENTPS) : 1, ' ');
    WRITEPOSNS(PARENTPOSITIONS);
    WRITECONNS(CONNBONDS);
    PARENTS := NEXT
   END
END;



PROCEDURE WRITECG(COMBINLIST : PCOMBINLIST;
                  INDENT     : INTEGER);
FORWARD;



PROCEDURE WRITEALTERNS(ALTERNATIVES : PALTERNLIST;
                       INDENT       : INTEGER);

VAR ALTNO,
    I     : INTEGER;

BEGIN
ALTNO := 0;
WHILE ALTERNATIVES <> NIL DO
   BEGIN
    FOR I := 1 TO INDENT DO WRITE(DIAGFILE, ' ');
    ALTNO := ALTNO + 1;
    WRITELN(DIAGFILE, ' ALT ', ALTNO : 2);
```

```
          WRITECG(ALTERNATIVES^.COMBINATION, INDENT + 4);
          ALTERNATIVES := ALTERNATIVES^.NEXT
         END
      END;



      PROCEDURE WRITECOMBIN(COMBINPTR : PCOMBINLIST;
                            ITEMNO,
                            INDENT    : INTEGER);

      { Outputs the information in the single combination gate pointed to by COMBINPTR. }

      VAR I : INTEGER;

      BEGIN
      WITH COMBINPTR^ DO
         BEGIN
          FOR I := 1 TO INDENT DO WRITE(DIAGFILE, ' ');
          WRITE(DIAGFILE,'Item. No.', ITEMNO : 3 );
          IF COMBINPTR^.PARENTPOSITIONS = NIL
             THEN WRITE(DIAGFILE, '[NIL]')
             ELSE WRITEPOSNS(PARENTPOSITIONS^);
         WRITEFREQ(FREQUENCY);
         IF BOTTOMBAR
            THEN BEGIN
                   WRITE(DIAGFILE, 'PS:', PSNO(COMBINPTR^.CHILDPS) : 1);
                   WRITEPOSNS(CHILDPOSITIONS);
                   WRITECONNS(CONNBONDS)
                 END
            ELSE BEGIN
                   WRITELN(DIAGFILE, 'ALTERNATIVES:');
                   WRITEALTERNS(ALTERNATIVES, INDENT);
                   FOR I := 1 TO INDENT DO WRITE(DIAGFILE, ' ');
                   WRITELN(DIAGFILE,'End of Item ', ITEMNO : 1, ' alternatives')
                 END
        END
      END;
```

```
PROCEDURE WRITECG; {FORWARD declaration above WRITEALTERNS}

VAR ITEMNO : INTEGER;

BEGIN
ITEMNO := 0;
WHILE COMBINLIST <> NIL DO
   BEGIN
    ITEMNO := ITEMNO + 1;
    WRITECOMBIN(COMBINLIST, ITEMNO, INDENT);
    COMBINLIST := COMBINLIST^.NEXT
   END
END;



PROCEDURE WRITECT(VAR CT : CTTYPE);

VAR ROWNO   : ATOMNUMBER;
    CNGNR   : 1..MAXCONGENERS;

BEGIN
WRITELN(DIAGFILE, 'SPECIFIC': 10);
FOR ROWNO := 1 TO MAXCT DO IF CT[ROWNO] <> NIL THEN WITH CT[ROWNO]^ DO
   BEGIN
    WRITE(DIAGFILE, ROWNO :2);
    IF ATOMICROW THEN WRITE(DIAGFILE, ATOM : 3)
                 ELSE WRITE(DIAGFILE, 'R', NAME:1, '  ');
    WRITE(DIAGFILE, CHARGE : 3, HYDROGENS : 2);
    FOR CNGNR := 1 TO MAXCONGENERS DO WITH CONGENERS[CNGNR] DO
        IF RELATIONSHIP = FRATERNAL THEN
           BEGIN
            WRITE(DIAGFILE, ORD(BOND) : 3);
            WRITE(DIAGFILE, ROWNUM : 3);
           END;
    IF NOT ATOMICROW AND (VALUES <> NIL)
       THEN BEGIN
             WRITELN(DIAGFILE, ' VALUES:');
             WRITECG(VALUES, 0);
```

```
              END
         ELSE WRITELN(DIAGFILE)
     END
  END;



  PROCEDURE WRITEPS(PTRPS    : PTRPSTYPE);


  BEGIN
  WRITE(DIAGFILE, PSNO(PTRPS):2 ,':');
  CASE PTRPS^.PSVARIETY OF
   DUMMY    : WRITELN(DIAGFILE,'   DUMMY   R', PTRPS^.SUBSTNAME: 1);
   UNKNOWN  : WRITELN(DIAGFILE,'   UNKNOWN'  );
   SPECIFIC : WRITECT(PTRPS^.CT);
   GENERIC  : BEGIN
               WRITELN(DIAGFILE,'   GENERIC');
               LISTPARAMS(DIAGFILE,PTRPS^.PARAMLIST)
              END;
   OTHER    : BEGIN
               WRITELN(DIAGFILE,'OTHER':7);
               WRITELN(DIAGFILE,PTRPS^.TERM)
              END
   END;
  IF PTRPS^.PARENTGATE <> NIL THEN WRITEPGS(PTRPS^.PARENTGATE);
  WRITELN(DIAGFILE);
  IF PTRPS^.CHILDGATE <> NIL THEN
     BEGIN
      WRITELN(DIAGFILE, 'CHILD GATES:');
      WRITECG(PTRPS^.CHILDGATE, 0)
     END;
  WRITELN(DIAGFILE, '-----------------------------------------');
  WRITELN(DIAGFILE)
  END;



  PROCEDURE WRITEDECN(DECLPTR : PPSLIST);

```

```
{ Writes out the information in DECLPTR. }

BEGIN
WHILE DECLPTR <> NIL DO WITH DECLPTR^ DO
   BEGIN
    WRITELN(DIAGFILE);
    WRITELN(DIAGFILE, 'Declared in ', PSNO(PARSTRUCT) : 2);
    IF PRNTPOSNS = NIL THEN WRITE(DIAGFILE, '[NIL]')
                       ELSE WRITEPOSNS(PRNTPOSNS^);
    WRITECONNS(CONNBONDS);
    IF FURTHERSUB <> NIL
       THEN WRITELN(DIAGFILE, 'Further substitution on PS ', PSNO(FURTHERSUB):2);
    IF COPYCHILDPS THEN WRITELN('COPYCHILDPS');
    DECLPTR := NEXT
   END
END;



BEGIN { Body of OUTINTREP }
WRITELN(DIAGFILE);
WRITELN(DIAGFILE, '***************** GENERIC STRUCTURE ', INTERNALREP.REFNUMBER : 3);
WRITELN(DIAGFILE);
WRITELN(DIAGFILE, 'Partial Structures: ');
READPTR := INTERNALREP.PSLIST;
WHILE READPTR <> NIL DO WITH READPTR^ DO
   BEGIN
    WRITEPS(PARSTRUCT);
    READPTR := NEXT
   END;
WRITELN(DIAGFILE);
WRITELN(DIAGFILE);
WRITELN(DIAGFILE, 'Declarations: ');
WRITELN(DIAGFILE);
FOR SUBST := 1 TO MAXVARS DO IF SUBST IN DECLSUBS THEN
   BEGIN
    WRITELN(DIAGFILE, '****   R', SUBST : 1, '  ****');
    WRITEDECN(RDECLARATIONTABLE[SUBST]);
    WRITELN(DIAGFILE)
   END;
```

```
WRITELN(DIAGFILE);
WRITELN(DIAGFILE);
WRITELN(DIAGFILE, 'Definitions: ');
WRITELN(DIAGFILE);
FOR SUBST := 1 TO MAXVARS DO IF SUBST IN DEFNSUBS THEN
   BEGIN
    WRITELN(DIAGFILE);
    WRITELN(DIAGFILE, '++++ R', SUBST : 1, '  ++++');
    WRITECG(RDEFINITIONTABLE[SUBST], 0)
   END;
WRITELN(DIAGFILE);
WRITELN(DIAGFILE);
WRITELN(DIAGFILE, 'Multipliers:');
FOR MULT := 1 TO MAXVARS DO IF MULT IN DECLMULT THEN
   BEGIN
    WRITELN(DIAGFILE);
    WRITELN(DIAGFILE, ':::: M', MULT:1, ' ::::');
    WRITE(DIAGFILE, 'Values :');
    WRITEFREQ(MDEFINITIONTABLE[MULT]);
    WRITELN(DIAGFILE);
    WRITELN(DIAGFILE)
   END;
WRITELN(DIAGFILE);
WRITELN(DIAGFILE)
END;
                  { of Procedure OUTINTREP
**********************************************************************************}



{--------------------------------------------------------------------------------}
PROCEDURE TIDYINTREP;

{ Deletes RDECLARATIONTABLE and RDEFINITIONTABLE, along with the latter's
  pendant gates. Then runs down from IRLISTTOP, deleting those PSs without
  parent gates, their child  gates. If DIAGNOSTICS is TRUE, then a list
  of PSs is output, with their PSNOs, and an indication of whether or
  not they have been deleted. }

VAR DECLPTR      : PPSLIST;
```

```
    PMPTR        : PMDECLIST;
    MULT         : MULTIPLIER;
    SUBST        : SUBSTITUENT;
    OLDECTR,
    NUMPSS,
    NUMDESTROYED : INTEGER;



PROCEDURE DESTROYCG(VAR COMBINBAR : PCOMBINLIST);

VAR COMBINPTR : PCOMBINLIST;
    ALTERNPTR : PALTERNLIST;

BEGIN
WHILE COMBINBAR <> NIL DO
   BEGIN
    COMBINPTR := COMBINBAR^.NEXT;
    IF COMBINBAR^.BOTTOMBAR
       THEN BEGIN
              DISPOSE(COMBINBAR, FALSE);
              ECTRSIZE := ECTRSIZE - 24
            END
       ELSE BEGIN
              WHILE COMBINBAR^.ALTERNATIVES <> NIL DO
                 BEGIN
                  DESTROYCG(COMBINBAR^.ALTERNATIVES^.COMBINATION);
                  ALTERNPTR := COMBINBAR^.ALTERNATIVES^.NEXT;
                  DISPOSE(COMBINBAR^.ALTERNATIVES);
                  ECTRSIZE := ECTRSIZE - 4;
                  COMBINBAR^.ALTERNATIVES := ALTERNPTR
                 END;
              DISPOSE(COMBINBAR, TRUE);
              ECTRSIZE := ECTRSIZE - 11
            END;
    COMBINBAR := COMBINPTR
   END
END;
```

```

BEGIN  {Body of TIDYINTREP}
OLDECTR := ECTRSIZE;
FOR SUBST := 1 TO MAXVARS DO IF SUBST IN DEFNSUBS THEN
   BEGIN
    REPEAT
     DECLPTR :=  RDECLARATIONTABLE[SUBST]^.NEXT;
     DISPOSE(RDECLARATIONTABLE[SUBST]);
     RDECLARATIONTABLE[SUBST] := DECLPTR
    UNTIL RDECLARATIONTABLE[SUBST] = NIL;

    DESTROYCG(RDEFINITIONTABLE[SUBST])
   END;

FOR MULT := 1 TO MAXVARS DO IF MULT IN DEFNMULT THEN
   REPEAT
    PMPTR := MDECLARATIONTABLE[MULT]^.NEXT;
    DISPOSE(MDECLARATIONTABLE[MULT]);
    MDECLARATIONTABLE[MULT] := PMPTR
   UNTIL MDECLARATIONTABLE[MULT] = NIL;

NUMPSS := 0;
NUMDESTROYED := 0;
IRLISTBOT := INTERNALREP.PSLIST;
WHILE IRLISTBOT <> NIL DO WITH IRLISTBOT^ DO
   BEGIN
    NUMPSS := NUMPSS + 1;
    IF DIAGNOSTICS
       THEN WRITE(DIAGFILE, NUMPSS:6, ADDRESSOF(PARSTRUCT^): 12);
    IF (PARSTRUCT^.PARENTGATE = NIL) AND (PARSTRUCT <> INTERNALREP.CONSTANTPART)
       THEN BEGIN
              DESTROYCG(PARSTRUCT^.CHILDGATE);
              CASE PARSTRUCT^.PSVARIETY OF
               UNKNOWN  : BEGIN
                           DISPOSE(PARSTRUCT, UNKNOWN);
                           ECTRSIZE := ECTRSIZE - 6
                          END;
               DUMMY    : BEGIN
                           DISPOSE(PARSTRUCT, DUMMY);
                           ECTRSIZE := ECTRSIZE - 8
```

```
                         END;
               SPECIFIC : BEGIN
                           DISPOSE(PARSTRUCT, SPECIFIC);
                           ECTRSIZE := ECTRSIZE - 70
                          END;
               GENERIC  : BEGIN
                           DISPOSE(PARSTRUCT, GENERIC);
                           ECTRSIZE := ECTRSIZE - 50
                          END;
               OTHER    : BEGIN
                           DISPOSE(PARSTRUCT, OTHER);
                           ECTRSIZE := ECTRSIZE - 22
                          END
              END;
              NUMDESTROYED := NUMDESTROYED + 1;
              IF DIAGNOSTICS THEN WRITELN(DIAGFILE, '   DESTROYED')
             END
         ELSE IF DIAGNOSTICS THEN WRITELN(DIAGFILE);
      IRLISTBOT := NEXT
     END;
   NUMPSS := NUMPSS - NUMDESTROYED;
   WRITELN('ECTR occupies ', ECTRSIZE : 5, ' words, in ', NUMPSS : 3, ' partial structures.');
   WRITELN(NUMDESTROYED : 2, ' partial structures  (', OLDECTR - ECTRSIZE : 5, ' words) were reclaimed.');
   IF DIAGNOSTICS THEN
      BEGIN
       WRITELN(DIAGFILE);
       WRITELN(DIAGFILE);
       WRITELN(DIAGFILE);
       WRITELN(DIAGFILE)
      END
   END;          { of TIDYINTREP
   ------------------------------------------------------------------------}




   BEGIN          (* Body of Procedure INTERPRET *)
   INITIALISE;
   NEXTTOKEN;
```

```
WHILE CHECKDELIM([GINPUT, GQUERY]) = INVALIDTOKEN DO ERROR(11,0);
INTERNALREP.QUERYSTRUCTURE := TOKEN.DELIMVAL = GQUERY;
NEXTTOKEN;
WHILE TOKEN.NATURE <> INTEGRAL DO ERROR(16,0);
INTERNALREP.REFNUMBER := TOKEN.INTEGVAL;
NEXTTOKEN;
WHILE CHECKDELIM([GSD]) <> GSD DO ERROR(12,0);
WITH INTERNALREP DO
   BEGIN
     READSD(CONSTANTPART, INPUTMODE=TERMINAL);
     NEW(PSLIST);
     ECTRSIZE := ECTRSIZE + 4;
     PSLIST^.PARSTRUCT := CONSTANTPART;
     PSLIST^.NEXT := NIL;
     IRLISTBOT := PSLIST
   END;
REPEAT
 STATEMENT;
 WHILE CHECKDELIM([GSEMI,GPERIOD]) = INVALIDTOKEN DO ERROR(24,0);
 IF TOKEN.DELIMVAL = GPERIOD THEN CHECKALLDONE
UNTIL TOKEN.DELIMVAL = GPERIOD;
RECORDMULTS;
WRITELN;
IF CONDITIONSPRESENT THEN WRITELN('(Conditions not yet implemented)');
WRITELN;
WRITELN('Generic Structure ',INTERNALREP.REFNUMBER : 6, '   accepted.');
IF DIAGNOSTICS THEN OUTINTREP;
TIDYINTREP;
WRITELN
END;
```

APPENDIX 4

Line Number Index to Routines in the

GENSAL Interpreter Program (Appendix 3)

APPENDIX 5

GLOBAL DECLARATIONS FOR GENPROG

```
CONST MAXCT          = 32;      { Size of single connection table }
      TERMLENGTH     = 32;      { Length of nomenclatural terms }
      MAXVARS        = 63;      { Number of substituents or multipliers }
      MAXCONGENERS   = 6;       { Number of congeners }
      MAXPACKETS     = 32;
      MAXBITS        = 32;
      MAXSCREENS     = 1024;
      MAXLENGTH      = 100;     { Length of GENESIS command line, or Gensal line }
      CTFLAG         = '{';
      GENEXFLAG      = '|';
      HSTFLAG        = '}';
      CONTNFLAG      = '\';
      ENDGENFLAG     = '~';
      NOTSET         = -1;      { Indicator for INTRECORD.TOPRANGE }


 TYPE TCOMMAND     =(CGENSAL, CFILE, CSAVE, CLIST, CSEARCH, CPRINT, CEDIT, CDRAW,
                     CDIAGNOSE, CDICT, CNEWTERM, CSYNONYM, CSTOP, CRUN,CCURRENT,
                     CFORWARD, CBACK,CTOP,CEND,CDELETE,CINSERT,CLOCATE,CHELPEDIT,
                     CQUIT);
      STRING14     = PACKED ARRAY[1..14] OF CHAR;
      LINESTRING   = PACKED ARRAY[1..MAXLENGTH] OF CHAR;
      PLINELIST    = ^LINELIST;
      LINELIST     = RECORD
                      LINE : LINESTRING;
                      NEXT,
                      LAST : PLINELIST
                     END;

      STRING4      = PACKED ARRAY[1..4] OF CHAR;

      USERTYPE     = PACKED RECORD
                      FSAUTH,
                      UPDAUTH,
                      SWEEP    : BOOLEAN;
                      NAME     : STRING4
                     END;

      PDOUBLIST    = ^DOUBLIST;
```

```
DOUBLIST    = RECORD
               FIRST,
               SECOND : INTEGER;
               NEXT   : PDOUBLIST
              END;
INTRECORD   = RECORD
               SUBRANGES : PDOUBLIST;
               TOPRANGE  : INTEGER
              END;
PINTEGSET   = ^INTEGSET;
INTEGSET    = SET OF 0..MAXVARS;
PTGROUPMEMS = ^TGROUPMEMS;
TGROUPMEMS  = RECORD
              CASE COMBINED       : BOOLEAN OF
               TRUE  : (COMBMEMS : PDOUBLIST);
               FALSE : (MEMBERS  : INTEGSET)
              END;

BONDORDER   =(NOTSPECIFIED, ANY, CHAIN, RING, SINGLE, DOUBLE, TRIPLE,
              CHAISING, CHAIDOUB, CHAITRIP, CHAITAUT, RINGSING, RINGDOUB,
              RINGTRIP, AROMATIC, RINGTAUT);
TCONNS      = NOTSET..2;
TCONNBONDS  = RECORD
              CASE CONNECTIONS : TCONNS OF
               NOTSET,
               0               : ();
               1               : (BOND  : BONDORDER);
               2               : (BONDA,
                                  BONDB : BONDORDER)
              END;
TSELECTMODE =(INDEPENDENT, ALLSAME, ALLDIFF, NOTALLSAME, NOTALLDIFF);

PTRPSTYPE   = ^PSTYPE;

PCOMBINLIST = ^COMBINLIST;
PALTERNLIST = ^ALTERNLIST;

COMBINLIST  = RECORD
               PARENTPOSITIONS : PTGROUPMEMS;
               FREQUENCY       : INTRECORD;
```

```
                NEXT              : PCOMBINLIST;
                CASE BOTTOMBAR    : BOOLEAN OF
                 TRUE             : (CHILDPS          : PTRPSTYPE;
                                     CHILDPOSITIONS : TGROUPMEMS;
                                     CONNBONDS      : TCONNBONDS);
                 FALSE            : (ALTERNATIVES   : PALTERNLIST)
               END;

ALTERNLIST  = RECORD
                COMBINATION : PCOMBINLIST;
                NEXT        : PALTERNLIST
              END;

PPARENTLIST = ^PARENTLIST;
PARENTLIST  = RECORD
                CHILDPOSITIONS,
                PARENTPOSITIONS : TGROUPMEMS;
                PARENTPS        : PTRPSTYPE;
                CONNBONDS       : TCONNBONDS;
                NEXT            : PPARENTLIST
              END;

RELATIVES   =(NONE, FRATERNAL, PARENTAL, FILIAL);
ATOMNUMBER  = 0..MAXCT;
SUBSTITUENT = 0..MAXVARS;

CONGARRAY   = ARRAY [1..MAXCONGENERS] OF
              RECORD
                BOND : BONDORDER;
                CASE RELATIONSHIP : RELATIVES OF
                 NONE,
                 PARENTAL,
                 FILIAL    : ();
                 FRATERNAL : (ROWNUM    : ATOMNUMBER)
                END;

STRING2     = PACKED ARRAY[1..2] OF CHAR;
NUMCONGENERS=0..MAXCONGENERS;

ROW         = RECORD
```

```
             CHARGE    : -9..9;
             HYDROGENS : NUMCONGENERS;
             CONGENERS : CONGARRAY;
             CASE ATOMICROW : BOOLEAN OF
              TRUE  : (ATOM    : STRING2);
              FALSE : (NAME    : SUBSTITUENT;
                       VALUES  : PCOMBINLIST)
            END;

CTTYPE      = ARRAY[1..MAXCT] OF ^ROW;

TPARAMETERS =(ATOMCOUNT, TBRANCH, QBRANCH, EUNSATURATION, YUNSATURATION,
              RINGCOUNT, RINGATOMS, RINGSUBSTITUTION, RINGFUSIONS,
              RINGAROMATIC, HETEROATOM);

TPARAMLIST  = ARRAY[TPARAMETERS] OF INTRECORD;

TPSVARIETY  =(DUMMY, UNKNOWN, SPECIFIC, GENERIC, OTHER);
STRING32    = PACKED ARRAY[1..32] OF CHAR;

PSTYPE      = RECORD
               VISITED         : BOOLEAN;
               CHILDGATE       : PCOMBINLIST;
               PARENTGATE      : PPARENTLIST;
               CASE PSVARIETY  : TPSVARIETY OF
                DUMMY          : (SUBSTNAME : SUBSTITUENT);
                UNKNOWN        : ();
                SPECIFIC       : (CT        : CTTYPE);
                GENERIC        : (PARAMLIST : TPARAMLIST);
                OTHER          : (TERM      : STRING32)
              END;

MULTIPLIER  = 0..MAXVARS;

FELDROW     = RECORD
               CHEM : STRING4  (* the atomic symbol, R group or * *);
               CHGE : -9..9;
               MULT : MULTIPLIER;
               AR   : ARRAY [1..MAXCONGENERS] OF ATOMNUMBER  (* the congeners *)
              END;
```

```
 BONDROW      = RECORD
                  NODE1,
                  NODE2 : ATOMNUMBER;
                  BOND  : 1..16
                END;
 TFELDMODE    =(NEWDIAGRAM, OLDDIAGRAM, NUMBERDRAW, NUMBERLESSDRAW);

 PIRLIST      = ^TIRLIST;
 TIRLIST      = RECORD
                  PARSTRUCT : PTRPSTYPE;
                  NEXT      : PIRLIST
                END;

 TINTERNALREP= RECORD
                  REFNUMBER      : INTEGER;
                  QUERYSTRUCTURE : BOOLEAN;
                  CONSTANTPART   : PTRPSTYPE;
                  PSLIST         : PIRLIST;
                END;


VAR DIAGFILE            : TEXT;          { Diagnostics file variable }
    TOPOGMFILE          : TEXT;          { Grammar file variable }
    USERFILE            : FILE OF USERTYPE;
    USER                : USERTYPE;
    DIAGFIL             : ALFA;          { Diagnostics file name }
    DIAGNOSTICS,                         { Diagnostics file indicator }
    STRUCTURECOMPLETED  : BOOLEAN;

    BUFFER              : LINESTRING;    { GENESIS or EDITOR command line or Gensal line }
    N                   : 0..MAXLENGTH;  { Character counter for BUFFER }
    WORKSPACE           : PLINELIST;     { Static pointer to held GENSAL }
    INTERNALREP         : TINTERNALREP;
    INSERTGENEX         : PLINELIST;                   { Lines of SPSfile Gensal expression }
    SPSPARAMLIST        : TPARAMLIST;                  { SPSfile parameter list }
    FELDCT              : ARRAY [1..MAXCT] OF FELDROW; { The Feldmann connection table }
    FELDBD              : ARRAY [1..MAXCT] OF BONDROW; { The Feldmann bonding table }
    FELDMODE            : TFELDMODE;                   { Calling mode for FELDMN }
    NUMOFNODES,                                        { Number of nodes in the Feldmann connection table }
```

```
   NUMOFBONDS          : ATOMNUMBER;                  { Number of bonds in the Feldmann bonding table }
   FELDFIL             : ALFA;                        { Feldmann transfer file }


{-----------------------------------------------------------------------------

                    PRIME APPLICATIONS LIBRARY ROUTINES                       }


FUNCTION CLOS$A(UNIT : SHORTINT): BOOLEAN;
EXTERN;


FUNCTION OPNV$A(OPNKEY : SHORTINT;
                NAME   : STRING14;
                NAMLEN,
                UNIT,
                VERKEY,
                WTIME,
                RETRYS : SHORTINT): BOOLEAN;
EXTERN;


FUNCTION POSN$A(POSKEY,             { 1=A$ABS }
                UNIT   : SHORTINT;
                POS    : INTEGER) : BOOLEAN;
EXTERN;


{-----------------------------------------------------------------------------
                 EXTERNAL FORTRAN SUBROUTINES
                     loaded in INOUTSUBS}
```

```
PROCEDURE GETLIN(VAR LINE : LINESTRING);
EXTERN;
{ FORTRAN subroutine to obtain a single line from the file already open on
  unit 1 and positioned at the correct place. }



 PROCEDURE FELDMN(VAR FELDMODE : TFELDMODE;
                  VAR FELDFIL  : ALFA);
 EXTERN;
 { Displays a structure diagram, for which the connection
   table is in the file FELDFIL }



 PROCEDURE GETNOM(VAR TERM : STRING32;
                  VAR ADDR : INTEGER);
 EXTERN;
 { FORTRAN subroutine to obtain the next TERM and ADDR from the file
   SPSDICT, which is already open on unit 1. }



 {*****************************************************************************}



 PROCEDURE ADDINTS (VAR PTR1     : PDOUBLIST;
                     LOWER, UPPER : INTEGER);

 VAR PTR2 : PDOUBLIST;

 BEGIN
 IF PTR1 = NIL
    THEN PTR2 := NIL
    ELSE WITH PTR1^ DO
           IF (SECOND = LOWER-1) OR (SECOND = LOWER)
              THEN SECOND := UPPER
              ELSE BEGIN
```

```
                PTR2 := PTR1;
                PTR1 := NIL
               END;
IF PTR1 = NIL
   THEN BEGIN
         NEW(PTR1);
         WITH PTR1^ DO
         BEGIN
          FIRST  := LOWER;
          SECOND := UPPER;
          NEXT   := PTR2
         END
        END
END;


PROCEDURE PRINTNOM(NOMENVAL : STRING32);

VAR M : 1..32;

BEGIN
FOR M := 1 TO 32 DO
   IF NOMENVAL[M] <> ' ' THEN WRITE(NOMENVAL[M])
END;


PROCEDURE DELETEGENSAL(VAR LINE1 : PLINELIST);

VAR LINE2 : PLINELIST;

BEGIN
WHILE LINE1 <> NIL DO
  BEGIN
   LINE2 := LINE1^.NEXT;
   DISPOSE(LINE1);
   LINE1 := LINE2
  END
 END;
```

```
{-------------------------------------------------------------------------------}
PROCEDURE DECODECT (VAR CTLINE : PLINELIST;
                    DISPLAYING : BOOLEAN);

VAR CHPOSN      : 0..MAXLENGTH;    { Character position in LINE }
    NODE        : ATOMNUMBER;      { Loop counter }
    M,                             { Miscellaneous counter }
    SPACE       : INTEGER;         { Ordinal value offset }



FUNCTION NEXTCH : CHAR;

{ Returns the next character in the string, taking new lines when necessary }

BEGIN
IF CHPOSN=MAXLENGTH
   THEN BEGIN
          CTLINE := CTLINE^.NEXT;
          CHPOSN := 2  { First character in each line is omitted (CONTNFLAG) }
        END
   ELSE CHPOSN := CHPOSN + 1;
NEXTCH := CTLINE^.LINE [CHPOSN]
END;



BEGIN { Body of DECODECT }
SPACE := ORD(' ');
CHPOSN := 1;     { first character in string is omitted }
NUMOFNODES := ORD(NEXTCH) - SPACE;
FOR NODE := 1 TO NUMOFNODES DO WITH FELDCT[NODE] DO
   BEGIN
    FOR M := 1 TO 4 DO CHEM[M] := NEXTCH;
    CHGE := ORD(NEXTCH) - SPACE - 9;
    MULT := ORD(NEXTCH) - SPACE;
    FOR M := 1 TO MAXCONGENERS DO AR[M] := ORD(NEXTCH) - SPACE
```

```
   END;
NUMOFBONDS := ORD(NEXTCH) - SPACE;
FOR M := 1 TO NUMOFBONDS DO WITH FELDBD[M] DO
   BEGIN
    NODE1 := ORD(NEXTCH) - SPACE;
    NODE2 := ORD(NEXTCH) - SPACE;
    BOND  := ORD(NEXTCH) - SPACE
   END;
IF DISPLAYING THEN
   BEGIN
    REWRITE(OUTPUT,FELDFIL);
    WRITELN(NUMOFNODES : 3);
    FOR NODE := 1 TO NUMOFNODES DO WITH FELDCT[NODE] DO
       BEGIN
        WRITE(CHEM, CHGE:2, '  ');
        IF MULT=0 THEN WRITE('    ')
                  ELSE WRITE('M',MULT:3);
        FOR M := 1 TO MAXCONGENERS DO IF AR[M] <> 0 THEN WRITE(AR[M] : 3);
        WRITELN
       END;
    WRITELN(NUMOFBONDS : 3);
    FOR M := 1 TO NUMOFBONDS DO WITH FELDBD[M] DO
        WRITELN(NODE1 : 3, NODE2 : 3, BOND : 3);
    REWRITE(OUTPUT,'@TTY');
    FELDMODE := NUMBERDRAW;
    FELDMN(FELDMODE,FELDFIL)
   END
END;
{------------------------------------------------------------------------}


{------------------------------------------------------------------------}
PROCEDURE ENCODECT(VAR CTLINE : PLINELIST);

VAR CHPOSN  : 0..MAXLENGTH;   { Charcter position in line }
    NODE    : ATOMNUMBER;     { Loop counter }
    M,                        { Miscellaneous counter }
    SPACE   : INTEGER;        { Ordinal value offset }
```

```
PROCEDURE STORECHAR (CH : CHAR);

{ Stores CH in the next position in the charcter string, taking new lines when
  necessary }

VAR NEWLINE : PLINELIST;

BEGIN
IF CHPOSN=MAXLENGTH
   THEN BEGIN
          NEW(NEWLINE);
          NEWLINE^.NEXT := NIL;
          NEWLINE^.LAST := CTLINE;
          CTLINE^.NEXT := NEWLINE;
          CTLINE := NEWLINE;
          CTLINE^.LINE[1] := CONTNFLAG;
          CHPOSN := 2
         END
   ELSE CHPOSN := CHPOSN+1;
CTLINE^.LINE [CHPOSN] := CH
END;



BEGIN {Body of ENCODECT}
SPACE := ORD(' ');
CHPOSN := 0;
STORECHAR(CTFLAG);   { Connection table indicator flag }
STORECHAR(CHR(NUMOFNODES + SPACE));
FOR NODE := 1 TO NUMOFNODES DO WITH FELDCT[NODE] DO
   BEGIN
    FOR M := 1 TO 4 DO STORECHAR(CHEM[M]);
    STORECHAR(CHR(CHGE+9+SPACE));
    STORECHAR(CHR(MULT+SPACE));
    FOR M := 1 TO MAXCONGENERS DO STORECHAR(CHR(AR[M]+SPACE))
   END;
STORECHAR(CHR(NUMOFBONDS + SPACE));
FOR M := 1 TO NUMOFBONDS DO WITH FELDBD[M] DO
```

```
  BEGIN
   STORECHAR(CHR(NODE1+SPACE));
   STORECHAR(CHR(NODE2+SPACE));
   STORECHAR(CHR(BOND+SPACE))
  END;
WHILE CHPOSN < MAXLENGTH DO STORECHAR(' ')
END;
{-----------------------------------------------------------------------}



{-----------------------------------------------------------------------}
PROCEDURE READSPSPARAMS(SPSTRING : PLINELIST);

VAR CH        : CHAR;
    PTR       : PDOUBLIST;
    CHPOSN    : 0..MAXLENGTH;
    PARAMETER : TPARAMETERS;



FUNCTION NEXTCH : CHAR;

{ Returns the next character in the string, taking new lines when necessary }

BEGIN
IF CHPOSN=MAXLENGTH
   THEN BEGIN
          SPSTRING := SPSTRING^.NEXT;
          CHPOSN := 1
        END
   ELSE CHPOSN := CHPOSN + 1;
NEXTCH := SPSTRING^.LINE [CHPOSN]
END;



BEGIN {Body of READSPSPARAMS}
CHPOSN := 1;    {first character in string is ommitted (HSTFLAG) }
FOR PARAMETER:= ATOMCOUNT TO HETEROATOM DO WITH SPSPARAMLIST[PARAMETER] DO
```

```
   BEGIN
    SUBRANGES := NIL;
    TOPRANGE := ORD(NEXTCH) - ORD('0');
    CH := NEXTCH;
    WHILE CH <> ' ' DO
       BEGIN
        NEW(PTR);
        PTR^.NEXT := SUBRANGES;
        PTR^.FIRST := ORD(CH)- ORD('0');
        PTR^.SECOND := ORD(NEXTCH) - ORD('0');
        SUBRANGES := PTR;
        CH := NEXTCH
       END
   END
END;
{--------------------------------------------------------------------------}


FUNCTION NORECORD(NOMEN       : STRING32;
                  VAR ADDRESS : INTEGER): BOOLEAN;

VAR SPSNOM : STRING32;

BEGIN
IF NOT OPNV$A(SHORT(1), 'LI2GEN>SPSDICT', SHORT(14), SHORT(1), SHORT(1), SHORT(1), SHORT(100))
   THEN PROGERROR(101); {File error  - cannot open SPSDICT}
REPEAT GETNOM(SPSNOM, ADDRESS)
UNTIL (SPSNOM=NOMEN) OR (SPSNOM[1]=' ');
IF NOT CLOS$A(SHORT(1)) THEN PROGERROR(102); {cannot close SPSDICT}
NORECORD := SPSNOM[1]=' '
END;



FUNCTION TERMREAD(VAR TERM : STRING32) : BOOLEAN;

VAR M, MM : 0..TERMLENGTH;

BEGIN
```

```
READLN(TERM : M);
FOR MM := 1 TO M DO IF TERM[MM] IN ['a'..'z']
    THEN TERM[MM] := CHR(ORD(TERM[MM]) - ORD('a') + ORD('A'));
TERMREAD := M>0
END;



PROCEDURE LISTPARAMS(VAR OUTFILE : TEXT;
                     PARAMLIST   : TPARAMLIST);

VAR PARAMETER : TPARAMETERS;
    PTR       : PDOUBLIST;

BEGIN
FOR PARAMETER := ATOMCOUNT TO HETEROATOM DO WITH PARAMLIST[PARAMETER] DO
   IF TOPRANGE <> 0 THEN
      BEGIN
       CASE PARAMETER OF
        ATOMCOUNT         : WRITE(OUTFILE,'C');
        TBRANCH           : WRITE(OUTFILE,'T');
        QBRANCH           : WRITE(OUTFILE,'Q');
        EUNSATURATION     : WRITE(OUTFILE,'E');
        YUNSATURATION     : WRITE(OUTFILE,'Y');
        RINGCOUNT         : WRITE(OUTFILE,'RC');
        RINGATOMS         : WRITE(OUTFILE,'RN');
        RINGSUBSTITUTION  : WRITE(OUTFILE,'RS');
        RINGFUSIONS       : WRITE(OUTFILE,'RF');
        RINGAROMATIC      : WRITE(OUTFILE,'RA');
        HETEROATOM        : WRITE(OUTFILE,'Z')
       END;
       PTR := SUBRANGES;
       WRITE(OUTFILE, '<');
       WHILE PTR <> NIL DO WITH PTR^ DO
          BEGIN
           WRITE(OUTFILE, FIRST : 1);
           IF FIRST <> SECOND THEN WRITE(OUTFILE, '-', SECOND : 1);
           PTR := NEXT;
           IF (PTR <> NIL) OR (TOPRANGE <> NOTSET) THEN WRITE(OUTFILE, ',')
          END;
```

```
       IF TOPRANGE <> NOTSET
          THEN WRITE(OUTFILE, TOPRANGE:1, '-> ')
          ELSE WRITE(OUTFILE, '> ')
      END;
WRITELN(OUTFILE);
WRITELN(OUTFILE)
END;



FUNCTION SPSVARIETY(ADDRESS    : INTEGER;
                    DISPLAYING : BOOLEAN) : TPSVARIETY;

{ Returns the variety of partial structure, whose record begins at ADDRESS in
  SPSFILE. The lines of the record in SPSFILE are in reverse order, and as they
  are read into a linked list of lines, the order is automatically put right.
  The first character of the first (in correct order) line indicates the nature
  of the partial structure. DECODECT is called to deal with connection tables
  (with DISPLAYING as its parameter); homologous series terms are handled by
  READSPSPARAMS, and listed by LISPARAMS if DISPLAYING is TRUE; Gensal
  expressions are stored in INSETGENEX, and listed by LISTGENEX if DISPLAYING
  is TRUE.  }

VAR SPSTRING  : PLINELIST;    { Lines of partial structure record }
    PARAMETER : TPARAMETERS;

BEGIN
IF NOT (OPNV$A(SHORT(1), 'LI2GEN>SPSFILE', SHORT(14), SHORT(1), SHORT(1), SHORT(1), SHORT(100))
       AND POSN$A(SHORT(1), SHORT(1), ADDRESS)) THEN
   PROGERROR(103); {File error - opening/positioning SPSFILE}
NEW(SPSTRING);
SPSTRING^.LAST := NIL;
SPSTRING^.NEXT := NIL;
GETLIN(SPSTRING^.LINE);
WHILE NOT (SPSTRING^.LINE[1] IN [CTFLAG, HSTFLAG, GENEXFLAG]) DO
   BEGIN
    NEW(SPSTRING^.LAST);
    SPSTRING^.LAST^.NEXT := SPSTRING;
    SPSTRING := SPSTRING^.LAST;
    SPSTRING^.LAST := NIL;
```

```
   GETLIN(SPSTRING^.LINE)
  END;
IF NOT CLOS$A(SHORT(1)) THEN PROGERROR(104); {File error (SPSVARIETY) - closing SPSFILE}
CASE SPSTRING^.LINE[1] OF
 CTFLAG:
   BEGIN
    SPSVARIETY := SPECIFIC;
    DECODECT(SPSTRING, DISPLAYING);
    DELETEGENSAL(SPSTRING)
   END;
 GENEXFLAG:
   BEGIN
    SPSVARIETY := OTHER;
    INSERTGENEX := SPSTRING^.NEXT;
    IF DISPLAYING THEN LISTGENEX(SPSTRING^.NEXT)
   END;
 HSTFLAG:
   BEGIN
    SPSVARIETY := GENERIC;
    READSPSPARAMS(SPSTRING);
    DELETEGENSAL(SPSTRING);
    IF DISPLAYING
       THEN BEGIN
              LISTPARAMS(OUTPUT, SPSPARAMLIST);
              FOR PARAMETER := ATOMCOUNT TO HETEROATOM DO
                  DESTROY(SPSPARAMLIST[PARAMETER].SUBRANGES)
            END
   END
 END { of case }
END;



PROCEDURE READFELDMANN;

{ Reads the Feldmann table from FELDFIL. }

VAR CH    : CHAR;
    NODE  : ATOMNUMBER;
    M     : INTEGER;
```

```
BEGIN
RESET(INPUT,FELDFIL);
READLN(NUMOFNODES);
FOR NODE := 1 TO NUMOFNODES DO WITH FELDCT[NODE] DO
   BEGIN
    FOR M := 1 TO 4 DO READ(CHEM[M]);
    READ(CHGE,CH,CH,CH);
    IF CH = 'M' THEN READ(MULT)
                ELSE MULT := 0;
    FOR M := 1 TO MAXCONGENERS DO
       IF EOLN(INPUT) THEN AR[M] := 0
                      ELSE READ(AR[M]);
    READLN
   END;

READLN(NUMOFBONDS);
FOR M := 1 TO NUMOFBONDS DO WITH FELDBD[M] DO
   READLN(NODE1,NODE2,BOND);
RESET(INPUT,'@TTY')
END;
```

APPENDIX 6

SAMPLE INTERPRETER SESSION

In this sample interpreter session the structure shown in Figure 3.3 is entered, with various errors being indicated by the program, and corrected by the user. After a "failure", a session using the editor corrects an erroneous structure diagram, and the whole structure is then reprocessed in non-interactive mode, before the user continues to input GENSAL statements.

```
 Enter Command : GENSAL

   1      18  GENSAL: INPUT 4163058
   2      18  GENSAL: SD

FELDMANN graphics system for structure diagram input and display:

#
RING 5

#
ABRAN 1 1 2 1 3 1 4 1

#
SATOM 1 3
ATOM TYPE =
N

#
SATOM 6 8
ATOM TYPE =
R5

#
SATOM 7 9
ATOM TYPE =
O
```

```
#
SBOND 2 7 4 9
BOND TYPE=
CD

#
RING 6

#
RING 6

#
ABOND 5 10

#
ABOND 5 16

#
ALTBD 10 11

#
ALTBD 16 17

#
D
              19.20                12.13
              .   .               .   .
             .    .             .    .
             18    21            11     14
               .    .           .     .
                .   .           .    .
                17.16          10.15
                     \         /
                      \       /
                       \     /
                        \   /
                         \ /
                    90    5
                     +   / \
                      + /   \
                       4     1N-6R5
                       !     !
                       !     !
                       3N----2
                      /       +
                     /         +
                    8R5         70


#
END
 3    702  GENSAL: R = H / SD
                     ^
**** ERROR 23
Integer expected.
```

```
Remainder of input line ignored

  4    702  GENSAL:  5 = H / SD

FELDMANN graphics system for structure diagram input and display:

#
CHAIN 3

#
ABRAN 1 1 1 1

#
SATOM 2
ATOM TYPE =
R3

#
SATOM 3
ATOM TYPE =
R2

 #
SATOM 4
ATOM TYPE =
R1

#
SATOM 5
ATOM TYPE =
*

#
D
              5*
               \
                \
                 1--2R3-3R2
                /
               /
              4R1


#
END
  5   1299  GENSAL: R1 = H / alkyl <1-7> ;
                     ^

**** ERROR 24
Unexpected symbol.

Remainder of input line ignored

  6   1299  GENSAL: ;
  7   1299  GENSAL: R1 = H / alkyl <1-7> ;
```

```
  8   1648  GENSAL: R2 = SD
```

FELDMANN graphics system for structure diagram input and display:

```
#
CHAIN 2

#
ABRAN 1 1 1 1

#
SATOM 2
ATOM TYPE =
O

#
SATOM 3
ATOM TYPE =
R4

#
SATOM 4
ATOM TYPE =
*

#
SBOND 1 4
BOND TYPE=
 CD

#
D
              4*
               +
                +
                 1--2O
                /
               /
              3R4


#
END

**** FAILURE 42
Bond types CS and CD are incompatible.

Edit existing GENSAL or start again!


Enter Command : EDIT
```

[The editor session is not shown here. It involves the replacement of the erroneous double bond in the last

```
structure diagram by a single bond]

> RUN


  1     18  GENSAL: INPUT 4163058
  2     18  GENSAL: SD

              19.20                12.13
              .   .               .   .
             .     .             .    .
             18     21           11    14
               .    .            .    .
                .   .            .   .
                17.16          10.15
                     \         /
                      \       /
                       \     /
                        \   /
                         \ /
                    90    5
                     +   / \
                      + /   \
                       4     1N-6R5
                       !     !
                       !     !
                       3N----2
                      /       +
                     /         +
                     8R5         70

  3    702  GENSAL: R
  4    702  GENSAL:  5 = H / SD

               5*
               \
                \
                 1--2R33R2
                /
               /
              4R1

  5   1299  GENSAL:
  6   1299  GENSAL: ;
  7   1299  GENSAL: R1 = H / alkyl <1-7> ;
  8   1648  GENSAL: R2 = SD

               4*
               \
                \
                 1++20
                /
               /
              3R4
```

```
End of stored GENSAL.
Input at the terminal:

  9   1887  GENSAL: ;
 10   1887  GENSAL: R3 = O / S;
 11   2292  GENSAL: RESTRICT <1> R5 <> H.
The following substituents remain undefined:
     R4

**** ERROR13
";" expected.

 Remainder of input line ignored

 12   2292  GENSAL: ;
 13   2292  GENSAL: R4 = 'acyl residue of naturally-
 14   2325  GENSAL:        occurring protein amino acid'.

(Conditions not yet implemented)

Generic Structure 4163058   accepted.
ECTR occupies  2124 words, in   9 partial structures.
 0 partial structures  (  279 words) were reclaimed.

Used  5.734 seconds.            Fragment generation begins.
```

APPENDIX 7

INTERPRETER ERROR MESSAGES

In these error messages the symbol # is replaced by an integer, and the symbol $ by a character (normally part of a bond type abbreviation).

1) Substituent R# has not been declared.

2) Multiplier M# has not been declared.

3) Position # which is implicit in nomenclatural term is not available.

4) No positions available for the further substitution implicit in nomenclatural term.

5) # is not a valid value for this parameter.

6) Position # is not available for attachment in all child structures.

7) Substitution is not possible in position #.

8) Insufficient substitutable positions for # substitutions.

9) Not enough substituent declations for # selective definitions.

10) # is too big a value for a multiplier.

11) "INPUT" or "QUERY" expected.

12) "SD" expected.

13) ";" expected.

14) ")" expected.

15) ";" or "END" expected.

16) Patent number expected.

17) "THEN" expected.

18) Substituent definition element expected.

19) Statement expected.

20) Substituent or multiplier group expected.

21) "<" expected.

22) ">" expected.

23) Integer expected.

24) Unexpected symbol.

25) Nomenclatural term or "'" expected.

26) Substituent expected.

27) Integer range must have increasing values left to right.

28) Substituent values must be in the range 1 to 63.

29) "'" expected.

30) Combination of doubly-connected substituents not permitted.

31) Connectivity incompatible with substituent(s) being defined.

32) Assignment operator expected.

33) "/" expected.

34) Position combination not permitted for singly-connected substitution.

35) No appropriate declaration for R#.

36) "+" expected.

37) Substituents in combination not declared in same partial structures.

38)

39) Positions specified previously are not available.

40) Error in stored GENSAL.

41) Structure Diagram rejected.

42) Bond types $$ and $$ are incompatible.

43) No available position in child structure for $$ bond.

44) Doubly-connected value for singly-connected substituent.

45) Bond size cannot be accomodated in parent structure at specified position #.

46) Bond size cannot be accomodated in parent structure at any position.

47) No positions available in parent structure for combined substitution.

48) Bond sizes cannot be accomodated at any pair of positions in parent structure.

49) Positions specified for further substitution on R# are not available.

50) No available positions for further substitution on R#.

51) Incompatible connectivities for substituents in this group.

52) No available positions in child structure for $$ and $$ bonds.

53) Number of bonds on substituent at node # does not agree with previous declaration.

54) More than two bonds on substituent at node #.

55) Illegal valency on atom at node #.

56) Variable-position label applied to atom at node #.

57) Multiplier appplied to node #, which is not a substituent.

58) Multivalent label or hydrogen at node #.

59) Maximum of two apical bond labels exceeded.

60) Illegal pattern of AROMATIC bonds at node #.

61) Illegal pattern of TAUTOMERIC bonds at node #.

BIBLIOGRAPHY

"When you steal from one author, it's plagiarism

if you steal from many, it's research"

William Mizner (1876-1933)

1. Anon., "No growth in chemical literature in 1981", CAS Report (12) 10 (1982)

2. Rowlett, R.J., "Gleaning patents with Chemical Abstracts", ChemTech 9 (6) 348-9 (1979)

3. Oppenheim, C., "The patents coverage of Chemical Abstracts", Information Scientist 8 (3) 133-8 (1974)

4. Oppenheim, C., "Recent changes in patent law and their implications for information services and information scientists", Journal of Documentation 34 (3) 217-229 (1978)

5. Dodd, V.S., "Developments in patent documentation", Aslib Proceedings 31 (4) 180-190 (1979)

6. Kaback, S.M., "Chemical structure searching in Derwent's World Patent Index", Journal of Chemical Information and Computer Sciences 20 (1) 1-6 (1980)

7. Markush, E.A., "Pyrazolone dyes", U.S. Patent No. 1506316, Aug 26th 1924

8. Rosa, M.C., "Outline of practice relative to 'Markush' claims", Journal of the Patent Office Society 34 (5) 324-345 (1952)

9. Coulter, R.I., "Comments on 'alternativeness' in claims (The Rule Against Or) Journal of the Patent Office Society 33 (11) 819-831 (1951)

10. Silk, J.A., Personal Communication to M.F. Lynch, letter dated 19 October 1979

11. Bouman, H., "Too prolific Markush", Journal of Documentation 26 (2) 161-163 (1970)

12. Beton, J.L., "The nature and grant of chemical patents", Chemistry and Industry 298-301 (5 April 1975)

13. Valance, E.H., "Understanding the Markush claim in chemical patents", Journal of Chemical Documentation **1** (2) 87-92 (1961)

14. Sneed, H.M.S., Turnipseed, J.H., Turpin, R.A., "A line-formula notation system for Markush structures", Journal of Chemical Documentation **8** (3) 173-178 (1968)

15. Krishnamurthy, E.V., Lynch, M.F., "Analysis and coding of generic chemical formulae in chemical patents", Journal of Information Science **3** 75-79 (1981)

16. Krishnamurthy, E.V., Lynch, M.F., "Formal description, coding and computer handling of generic formulae in chemical patents. Report on tenure of a British Library Senior Visiting Fellowship by Prof. E.V. Krishnamurthy at the Postgraduate School of Librarianship and Information Studies, University of Sheffield, February 1979." BLR&D Report No 5490, British Library (1979)

17. Geivandov, E.A., ["Language for notation of generalised structures of organic compounds containing alternative delocalised fragments (Markush Structures)"], Nauchno-tekhnicheskiya Informatsiya Seriya 2 (10) 21-24, 46 (1972) [IN RUSSIAN]

18. Warr, W.A., "Software for storage and retrieval of chemical structures", presented at CNA(UK) Seminar on the Future of Chemical Documentation, Exeter, September 1982.

19. Eakin, D.R., "The ICI CROSSBOW system", in "Chemical Information Systems", eds. Ash, J.E., Hyde, E. Ellis Horwood (1975)

20. Smith, E.G., Baker, P.A., "The Wiswesser Line-Formula Chemical Notation", 3rd edition, Chemical Information Management Inc. (1972)

21. Polton, D., "Installation and experience with MACCS (Molecular Access System)", Online Review, **6** (3) 235-242 (1982)

22. Farmer, N.A., O'Hara, M.P., "CAS ONLINE: a new source of substance information from Chemical Abstracts Service", Database **3** 10-25 (1980)

23. Dubois, J.E., "French national policy for chemical information and the DARC system as a potential tool of this policy", Journal of Chemical Documentation, **13** (1) 8-13 (1973)

24. Dubois, J.E., "The DARC system in chemistry", in "Computer Representation and Manipulation of Chemical Information", eds. Wipke, W.T., Heller, S.R., Feldmann,

R.J., Hyde, E. Wiley (1974)

25. Dubois, J.E., "Ordered chromatic graph and limited environment concept", in "Chemical Applications of Graph Theory", ed. Balaban, A.T. Academic Press (1976)

26. Jackson, F.T., "Markush Structures", Proceedings of the CNA(UK) Seminar on Integrated Databases for Chemical Systems, 9-11 April 1979, University of Kent at Canterbury, pp. 134-157. CNA(UK) (1981).

27. Japan Patent Association, "Patent Information Study Team in Europe and America. Report, 31 August 1975", Derwent Publications (1976)

28. Smith, R.G., Anderson, L.P., Jackson, S.K., "Online retrieval of chemical patent information. An overview and brief comparison of three major files", Journal of Chemical Information and Computer Sciences 17 (3) 148-157 (1977)

29. Silk, J.A., "Present and future prospects for structural searching of the journal and patent literature", Journal of Chemical Information and Computer Sciences 19 (4) 195-198 (1979)

30. Oppenheim, C., "Patent Information Online - a Review", Proceedings of the 5th International Online Information Meet Cunard Hotel, London, December 1981, pp. 91-99. Learned Information, Oxford (1981).

31. Rowland, J.F.B., "Information Transfer and Use in Chemistry. Final Report of the Chemical Information Review Committee", British Library R&D Department Report 5385 (1978)

32. Rowland, F., "How will chemical information develop?", Chemistry in Britain 14 (7) 342-4 (1978)

33. Kaback, S.M., "A user's experience with the Derwent patent files", Journal of Chemical Information and Computer Sciences 17 (3) 143-148 (1977)

34. Kaback, S.M., "Derwent Search Aids", Database, 5 (3) 19-21 (1982)

35. Hyams, M., "Chemical patents information", Chemistry in Britain 6 416-420 (1968)

36. Deforeit, H., Caric, A., Combe, H., Leveque, S., Malka, A., Valls, J., "CORA - a semiautomatic coding system. Application to the coding of Markush formulas", Journal of Chemical Documentation 12 (4) 230-232 (1972)

37. Ramussen, L.E., Van Oot, J.G., "Operation of Du Pont's central patent index", Journal of Chemical Documentation 9 (4) 201-206 (1969)

38. Balent, M.Z., Emberger, J.M., "A unique chemical fragmentation system for indexing patent literature", Journal of Chemical Information and Computer Sciences 15 (2) 100-104 (1975)

39. Balent, M.Z., Lotz, J.W., "Polymers and patents don't mix - easily", Journal of Chemical Information and Computer Sciences 19 (2) 80-83 (1979)

40. Cattley, J.M., Rief, T.A., Moore, J.E., Banks, D.G., O'Leary, P.T., "Retrieving patents by weighted term searching", Chemical Engineering Progress 62 (10) 91-96 (1966)

41. Rössler, S., Kolb, A., "The GREMAS system, an integral part of the IDC system for chemical documentation", Journal of Chemical Documentation 10 (2) 128-134 (1970)

42. Internationale Dokumentationsgesellschaft fur Chemie m.b.H. Fachinformationszentrum Chemie, "IDC System Overview. 2. IDC and its methods of operation", IDC (Undated)

43. Fugmann, R., "The IDC System", in "Chemical Information Systems", eds. Ash, J.E., Hyde, E., pp. 195-226, Ellis Horwood (1975)

44. Pötzcher, G., "Dokumentation vom Markush-Structuren in Organischen Patenten" ["Documentation of Markush structures in organic patents"], in "Kleincomputer in Information und Dokumentation", Deutcher Dokumentartag 1981, Mainz, 5-8 Oktober 1981, ed. H. Strohl-Goebel, K.G. Saur, Munich (1982)

45. Mullen, A., "Informationswesen in der Chemie. Teil II. Neue Tendenzen" [Means of information in chemistry. Part II. New trends], Praxis der Naturwissenschaften Chemie 29 (8) 243-247 (1980) [IN GERMAN]

46. Oppenheim, C., "The performance of the Chemical Abstracts subject and formula indexes in retrieving compounds disclosed in chemical patent Information Scientist 9 (3) 107-111 (1975)

47. Kaindl, H., "The function of the internal database - burial ground or intelligence service", Presented at CNA(UK) Seminar on "The Future of Chemical Documentation", University of Exeter, September 1982

48. Schwartz, J.H., "Some observations concerning Chemical Abstracts' formula indexes", Journal of Chemical

Documentation 9 (3) 169-171 (1969)

49. Silk, J.A., Howarth, K.E., Waterman, J.R., Wilkins, M., "Substructure searching of the CA Registry file. Some implications of a publicly-available system", ICI Chemical Structures Task Force Private Circulation Document. Unpublished (1980)

50. Bois, R., Chaumier, J., "A comparative analysis of the DARC system and the information and documentation system of the IDC", World Patent Information 2 (2) 61-66 (1980)

51. Dubois, J.E., Veillard, H., Panaye, A., "Systeme DARC. XV. Theorie de generation-description VI. Les structures ambigues: description et chainage par un graphe fictif documentaire", Bulletin de la Societe Chimique de France. Part 2 1996-2002 (1973)

52. Berte, M., Delaet, F., "Recherche documentaires automatiques dans le domaine des Prostaglandines a l'aide du Systeme DARC" [Automatic document search in the Prostaglandin field by means of the DARC system], Bulletin des Societes Chimiques Belges 88 (3) 175-191 (1979). [In French, English Abstract].

53. Gay, J.-P., "The DARC system", Presented at CNA(UK) Seminar on the "Future of Chemical Documentation", University of Exeter, September 1982

54. Dyson, G.M., "Generic (or Markush) groups in notation and search programs, with particular reference to patents", Information Storage and Retrieval 2 59-71 (1964).

55. Hayward, H.W., Tauber, S.J., "The HAYSTAQ experiment", in Proceedings of the Fifth Annual Meeting of the Commit for International Cooperation in Information Retrieval Among Examining Patent Offices (ICIREPAT), held London, 31 August 10 September 1965, pp 337-50. Published Thompson, Washington D.C. (1966)

56. Tauber, S.J., "Digital handling of chemical structures and associated information", in Proceedings of 20th National Conference of the Association for Computing Machinery, Ohio, August 1965, pp. 206-16 ACM Publication P-65, Lewis Winner, New York (1965)

57. Kirby, C.E.L., Anderson, R.K., Tauber, S.J., "Tentative connection table word format for Markush-type generic chemical structures", Appendix C (Working Paper 66-3 (27 January 1966)) to "Progress in techniques for manipulating and organising chemical information", National Bureau of Standards Report 9587 (January 1967)

58. Tauber, S.J. et al., "Developing computer programs for searching specific and generic structures, including the formatting of Markush structures", in "Progress in techniques for manipulating and organising chemical information", pp. 23-34 National Bureau of Standards Report **9587** (January 1967)

59. Fraser Williams (Scientific Systems) Ltd., "Proposals for the use of CROSSBOW techniques in the computer handling of Markush structures", Unpublished internal document (1978)

60. Krishnamurthy, E.V., Sankar, P.V., Krishnan, S., "ALWIN - Algorithmic Wiswesser Notation system for organic compounds", Journal of Chemical Documentation **14** (3) 130-141 (1974)

61. Krishnan, S., Krishnamurthy, E.V., "Compact grammar for Algorithmic Wiswesser Notation using Morgan name", Information Processing and Management **12** 19-34 (1976)

62. Krishnan, S., "Compact ALWIN grammar and algorithms for computer handling of organic chemical structures", Ph.D. Thesis, Molecular Biophysics Unit, Indian Institute of Science, Bangalore, India (1975)

63. Howe, W.J., Hagadone, T.R., "Progress toward an online chemical and biological information system at the Upjohn company", in "Retrieval of Medicinal Chemical Information", eds. Howe, W.J., Milne, M.M., Pennel, A.F. ACS Symposium Series **84** 107-131 (1978)

64. Howe, W.J., Hagadone, T.R., "Molecular substructure searching: computer graphics and query entry methodology", Journal of Chemical Information and Computer Sciences in press (1982)

65. Lynch, M.F., "Screening large chemical files", in "Chemical Information Systems", eds. Ash, J.E., Hyde, E., pp. 177-194 Ellis Horwood (1974)

66. Ash, J.E., "Connection Tables and their role in a system", in "Chemical Information Systems", ed. Ash, J.E., Hyde, pp. 156-176, Ellis Horwood (1975)

67. Welford, S.M., "Topological Chemical Grammars and the Generation of Limited Environment Fragments from Generic Chemical Structures", Unpublished Ph.D. Thesis, University of Sheffield (1982)

68. Hopcroft, J.E., Ullman, J.D., "Formal languages and their relation to automata", Addison-Wesley (1969)

69. Ginsburg, S., "The Mathematical Theory of Context-Free Languages", McGraw Hill, New York (1966)

70. Hopcroft, J.E., Ullman, J.D., "Introduction to Automata Theory, Languages and Computation" Addison-Wesley, 1979

71. Backhouse, R.C., "Syntax of Programming Languages. Theory and Practice", Prentice Hall (International Series in Computer Science) (1979)

72. Cleaveland, J.C., Uzgalis, R.C., "Grammars for Programming Languages", Elsevier North-Holland (1977)

73. Salomaa, A., "Formal Languages", Academic Press (1973)

74. Chomsky, N., "On certain formal properties of grammars", Information and Control 2 137-167 (1959)

75. Chomsky, N., "Syntactic Structures", Mouton & Co., 's-Gravenhage (1957)

76. Chomsky, N., Miller, G.A., "Introduction to the formal analysis of natural languages", in "Handbook of Mathematical Psychology" eds Luce, R.D., Bush, R.R. and Galanter, E., Volume 2, pp. 269-321, Wiley (1963)

77. Chomsky, N., "Formal properties of grammars", in "Handbook of Mathematical Psychology", eds Luce, R.D. Bush, R.R. and Galanter E., Volume 2, pp. 323-418, Wiley (1963)

78. Chomsky, N., "Aspects of the theory of syntax", MIT Press (1965)

79. Ginsburg, S., Greibach, S.A., "Deterministic context-free languages", Information and Control 9 (6) 620-648 (1966)

80. Aho, A.V., Ullman, J.D., "The Theory of Parsing, Translation and Compiling", 2 Vols, Prentice Hall (1972)

81. Aho, A.V., Ullman, J.D., "Principles of Compiler Design", Addison-Wesley (1977)

82. Knuth, D.E., "On the translation of languages from left to right", Information and Control, 8 607-639 (1965)

83. Lewis, P.M., Stearns, R.E., "Syntax-directed transduction", Journal of the Association for Computing Machinery 15 (3) 465-488 (1968)

84. Rosenkrantz, D.J., Stearns, R.E., "Properties of deterministic top-down grammars", Information and Control 17 226-256 (1970)

85. Greibach, S., "A new Normal-Form theorem for context-free phrase structure grammars", Journal of the Association for Computing Machinery 12 42-52 (1965)

86. Koranjak, A.J., Hopcroft, J.E., "Simple deterministic languages", Proceedings 7th Annual Symposium on Switching and Automata Theory, IEEE Conference Record, Publication No. 16-C-40 pp. 36-46 (1966)

87. Beatty, J.C., "On the relationship between LL(1) and LR(1) grammars", Journal of the Association for Computing Machinery 29 (4) 1007-1022 (1982)

88. Knuth, D.E., "Top-down syntax analysis", Acta Informatica 1 79-110 (1971)

89. Foster, J.M., "Automatic Syntax Analysis", Macdonald (1970)

90. Sammet, J.E., "Programming Languages. History and Fundamentals", Prentice Hall (1969)

91. Burkhardt, W.H., "Metalanguage and syntax specification", Communications of the Association for Computing Machinery 8 (5) 304-5 (1965)

92. Naur, P., "Report on the algorithmic language ALGOL60", Communications of the Association for Computing Machinery 3 (5) 299-314 (1960)

93. Naur, P. (ed.), "Revised report on the algorithmic language ALGOL60", Communications of the Association for Machinery 6 (1) 1-20 (1963); Computer Journal 5 349-367 (1963); Numerische Mathematik 4 420-452 (1963)

94. Woodward, P.M., Bond, S.G., "User's Guide to Algol68R", Her Majesty's Stationery Office, London (1974)

95. Jensen, K., Wirth, N., "Pascal User Manual and Report", Springer Verlag, New York (1975)

96. Pyle, K., "The Ada Programming Language", Prentice Hall (1981)

97. Knuth, D.E., "Backus Normal Form vs. Backus Naur Form" (letter to the editor), Communications of the Association for Computing Machinery 7 (12) 735-6 (1964)

98. Ginsburg, S., Rice, H.G., "Two families of languages related to ALGOL", Journal of the Association for Computing Machinery 9 350-371 (1962)

99. Iverson, K.E., "A method of syntax specification", Communications of the Association for Computing

Machinery 7 (10) 588-9 (1964)

100. Taylor, W., Turner, L., Waychoff, R., "A syntactical chart of ALGOL 60", Communications of the Association for Computing Machinery 4 393 (1961)

101. Wirth, N., "What can we do about the unnecessary diversity of notations for syntactic definitions?", Communications of the Association for Computing Machinery 20 (11) 822-3 (1977)

102. Irons, E.T., "A syntax directed compiler for ALGOL 60", Communications of the Association for Computing Machinery 4 51-55 (1961)

103. Conway, M.E., "Design of a separable transition-diagram compiler", Communications of the Association for Computing Machinery 6 (7) 396-408 (1963)

104. Pratt, T.W., "Programming languages: design and implementation", Prentice Hall, Englewood Cliffs, N.J. (1975)

105. Brown, P.J., "Writing Interactive Compilers and Interpreters", Wiley (1979)

106. Nicholls, J.E., "The Structure and Design of Programming Languages", Addison-Wesley (1975)

107. Floyd, R.W., "The syntax of programming languages - a survey", IEEE Transactions on Electronic Computers 13 (4) 346-53 (1964)

108. Tennent, R.D., "Principles of Programming Languages", Prentice Hall (International Series in Computer Science) (1981)

109. Wirth, N., "Algorithms + Data Structures = Programs", Prentice Hall (1976)

110. Welsh, J., McKeag, M., "Structured System Programming", Prentice Hall (International Series in Computer Science) (1980)

111. Wirth, N., "The programming language Pascal", Acta Informatica 1 35-63 (1971)

112. Addyman, A.M., "A draft proposal for Pascal", SIGPLAN Notices 15 (4) 1-69 (1980)

113. Addyman, A.M., et al. "A draft description of Pascal", Software Practice and Experience 9 (5) 381-424 (1979)

114. Wilson, I.R., Addyman, A.M., "A Practical

Introduction to Pascal", Macmillan, London (1978)

115. Findlay, W., Watt, D.A., "Pascal: an Introduction to Methodical Programming", Pitman (1978)

116. Atkinson, L.V., "Pascal Programming", Wiley (1980)

117. McGregor, J.J., Watt, A.H., "Simple Pascal", Pitman Books (1981)

118. Wirth, N., "The design of a Pascal compiler", Software Practice and Experience 1 309-333 (1971)

119. Stevens, R., Graham, I., "Matching the abilities of man and computer", Practical Computing pp. 63-7 (August 1979)

120. Fletcher, D., "Pascal Power", Datamation 25 (8) 142-5 (1979)

121. Tenenbaum, A.M., Augenstein, M.J., "Data Structures using Pascal", Prentice Hall (1981)

122. Lecarme, O., Desjardins P., "More comments on the programming language Pascal", Acta Informatica 4 231-243 (1975)

123. Habermann, A.N., "Critical comments on the programming language Pascal", Acta Informatica 3 47-57 (1973)

124. Conradi R., "Further critical comments on Pascal, particularly as a systems programming language", SIGPLAN Notices 11 (11) 8-25 (1976)

125. Main, A., "Pascal creates some bad habits", Computer Weekly (770), p.11 (13 Aug 1981)

126. Welsh, J., Sneeringer, W.J., Hoare, C.A.R., "Ambiguities and insecurities in Pascal", Software Practice and Experience 7 685-696 (1977)

127. Cole, J., Morrison, R., "Linguistic Disfigurement", Computing 8 (4) 45 (30 Oct 1980)

128. Tanenbaum, A.S., "A comparison of Pascal and Algol 68", Computer Journal 21 (4) 316-323 (1978)

129. Tennent, R.D., "A note on files in Pascal", BIT 17 362-6 (1977)

130. Doty, K.L., "A top-down evaluation of Pascal", Computer Design 19 (5) 167-177 (1980)

131. Wirth, N., "An assessment of the programming language

Pascal", IEEE Transactions on Software Engineering SE-1 (2) 192-198 (1975)

132. Glass, R.L., "From Pascal to Pebbleman... and beyond", Datamation 25 (8) 146-50 (1979)

133. Pyle, I., "Ada proves it is equal to 'almost impossible' feats", Computing 9 (44) 26-27 (29 October 1981)

134. Hoare, C.A.R., "The Emperor's Old Clothes" (1981 ACM Turing Award Lecture), Communications of the Association for Computing Machinery 24 75-83 (1981)

135. Skelly, P.G., "The ACM position on standardisation of the Ada language", Communications of the Association for Computing Machinery 25 (2) 118-120 (1982)

136. Ledgard, H.F., Singer, A., "Scaling down Ada (or, Towards a standard Ada subset)", Communications of the Association for Computing Machinery 25 (2) 121-125 (1982)

137. Ince, D., "Why ADA shouldn't go to war", The Guardian p.13 (25th June 1981)

138. Gilbert, J.R., Martin, C.W., "Sheffield University V-mode Pascal System for PR1ME Computers", Unpublished Information File, Computing Services Department, University of Sheffield (1981)

139. de Bakker, J.W., "Semantics of programming languages", Advances in Information Systems Science 2 173-227 (1969)

140. Pagan, F.G., "Formal Specification of Programming Languages", Prentice Hall (1981)

141. Milne, R., Strachey, C., "A Theory of Programming Language Semantics", Chapman and Hall, London (1976)

142. Hoare, C.A.R., Wirth, N., "An axiomatic definition of the programming language Pascal", Acta Informatica 2 335-355 (1973)

143. Hoare, C.A.R., "An axiomatic basis for computer programming", Communications of the Association for Computing Machinery 12 (10) 576-80, 583 (1969)

144. Kupka, I., Wilsing N., "Conversational Languages", Wiley (1980)

145. "The Interpretive Basic Programmer's Guide", Prime Computer Inc., IDR 1813 (1978)

146. Spencer, D.D., "A Guide to BASIC Programming; a Time-Sharing Language", Addison-Wesley (1970)

147. Barron, D.W., "Approaches to conversational Fortran", Computer Journal **14** (2) 123-127 (1971)

148. Atkinson, L.V., McGregor, J.J., "CONA - a conversational Algol system", Software - Practice and Experience **8** 699-708 (1978)

149. Atkinson, L.V., North, S.D., "COPAS - a conversational Pascal system", Software: Practice and Experience **11** 819-829 (1981)

150. Anon., "Abacus makes Pascal as easy as Basic", Computing 8 (17) 6 (24 Apr 1980)

151. Atkinson, L.V., McGregor, J.J., North, S.D., "Context sensitive editing as an approach to incremental compilation", Computer Journal 24 (3) 222-229 (1981)

152. Henry, W.M., Leigh, J.A., Tedd, L.A., Williams, P.W., "Online Searching. An Introduction", Butterworth (1980)

153. Hall, J.L., "Online Information Retrieval Sourcebook", Aslib (1977)

154. Morgan, M.M., Beaman, P.D., Shusman, D.J., Hupp, J.A., Zielstorff, R.D., Barnett, G.O., "Medical query language", Proceedings of the Annual Symposium on Computer Applications in Medical Care, Fifth Symposium, Washington D.C., 1-4 November 1981, 5 322-325 (1981)

155. Orf, H.V., "CHMTRN - LHASA Data Table Language", Unpublished Document, LHASA Project (July 1978)

156. Corey, E.J., Wipke, W.T., Cramer, R.D., Howe, W.J., "Computer-assisted synthetic analysis. Facile man-machine communication of chemical structure by interactive computer graphics", Journal of the American Chemical Society 94 421-430 (1972)

157. Corey, E.J., Cramer, R.D., Howe, W.J., "Computer-assisted synthetic analysis for complex molecules. Methods and procedures for machine generation of synthetic intermediates", Journal of the American Chemical Society 94 440-459 (1972)

158. Dyott, T.M., "Utilization of Stereochemistry and Other Aspects of Computer-Assisted Synthetic Design", Ph.D. Thesis, Princeton University (1973). University Microfilms Order No. 74-9677. Abstracted in Dissertation Abstracts 34 5382-B

159. Wipke, W.T., Braun, H., Smith, G., Choplin, F.,

Seiber, W., "SECS - Simulation and Evaluation of Chemical Synthesis: strategy and planning", in Computer Assisted Organic Synthesis, eds. Wipke, W.T. Howe, W.J., ACS Symposium Series 61 97-127 (1977)

160. Wipke, W.T., Ouchi, G.I., Krishnan, S., "Simulation and evaluation of chemical synthesis - SECS: an application of artificial intelligence techniques", Artificial Intelligence 11 173-193 (1978)

161. Elder, M., Unpublished work (1981)

162. Lin, C.H., Lee, P.L., Lin, J.L., "A SEFLIN compiler - automatic syntactic analysis of separate feature linear notations of chemical compounds", Proceedings of the National Science Council of the Republic of China. Part A 5 (3) 165-172 (1981)

163. Lin, C.-H., "SEFLIN - separate feature linear notation system for chemical compounds", Journal of Chemical Information and Computer Sciences 18 (1) 41-7 (1978)

164. Fehder, P.L., Burnett, M.P., "Syntactic scanning of chemical information", Journal of Chemical Documentation 5 8-13 (1965)

165. Carpenter, N., "Syntax-directed translation of organic chemical files into their two-dimensional representation", Computers in Chemistry 1 25-28 (1976)

166. Barker, P.G., "Syntactic definition and parsing of molecular formulas", Computer Journal 18 355-359 (1978)

167. Kirby, G.H., Morgan, C.H., Rayner, J.D., "Microcomputer Formulae", Education in Chemistry 15-16 (1981)

168. Garfield, E., "An algorithm for translating chemical names to molecular formulas", Journal of Chemical Documentation 2 177-179 (1962)

169. Dyson, G.M., "A cluster of algorithms relating the nomenclature of organic compounds to their structure matrices and ciphers", Information Storage and Retrieval 2 159-199 (1964)

170. Vander Stouw, G.G., Elliott, P.M., Isenberg, A.C., "Automated conversion of chemical substance names to atom-bond connection tables", Journal of Chemical Documentation 14 185-193 (1974)

171. Tauber, S.J., Rankin, K., "Valid structure diagrams and chemical gibberish", Journal of Chemical Documentation 12 (1) 30-34 (1972)

172. Rankin, K., Tauber, S.J., "Linguistics as a basis for analysing chemical structure diagrams", Journal of Chemical Documentation 11 (3) 139-141 (1971)

173. Whitlock, H.W., "An organic chemist's view of formal languages", ACS Symposium No 61 pp. 60-80 (American Chemical Society) (1976)

174. Welford, S.M., Lynch, M.F., Barnard, J.M., "Computer storage and retrieval of generic chemical structures in patents. 3. Chemical grammars and their role in the manipulation of chemical structures", Journal of Chemical Information and Computer Sciences 21 (3) 161-168 (1981)

175. Lynch, M.F., Barnard, J.M., Welford, S.M., "Computer storage and retrieval of generic chemical structures in patents. 1. Introduction and general strategy", Journal of Chemical Information and Computer Sciences 21 (3) 148-150 (1981)

176. Barnard, J.M., Lynch, M.F., Welford, S.M., "Computer storage and retrieval of generic chemical structures in patents. 2. GENSAL, a formal language for the description of generic chemical structures", Journal of Chemical Information and Computer Sciences 21 (3) 151-161 (1981)

177. Barnard, J.M., Lynch, M.F., Welford, S.M., "Computer storage and retrieval of generic chemical structures in patents. 4. An extended connection table representation for generic structures", Journal of Chemical Information and Computer Sciences 22 (3) 160-164 (1982)

178. Hill, S.J., "Coding Manuals for Chemical Structure Systems", Unpublished M.Sc. Dissertation, Sheffield University (1981)

179. Feldmann, R.J., Milne, G.W.A., Heller, S.R., Fein, A., Miller, J.A., Koch, B., "An interactive substructure search system", Journal of Chemical Information and Computer Sciences 17 (3) 157-163 (1977)

180. Elder, M., Hull, S.E., Machin, P.A., Mills, O.S., "CSSR. Crystal Structure Search Retrieval. User Manual", 2nd Edition, Daresbury Laboratory, Science and Engineering Research Council, (1981)

181. Milne, G.W.A., Heller, S.R., "NIH/EPA Chemical Information System", Journal of Chemical Information and Computer Sciences 20 (4) 204-211 (1980)

182. Gluck, D.J., "A chemical structure storage and search system developed at Du Pont", Journal of Chemical Documentation 5 43-51 (1965)

183. Roos-Kozel, B.L., Jorgensen, W.L., "Computer-assisted mechanistic evaluation of organic reactions. 2. Perception of rings, aromaticity and tautomers", Journal of Chemical Information and Computer Sciences **21** (2) 101-111 (1981)

184. Corey, E.J., Wipke, W.T., Cramer, R.D., Howe, W.J., "Techniques for perception by a computer of synthetically significant structural features in complex molecules", Journal of the American Chemical Society **94** 431-439 (1972)

185. Corey, E.J., Petersson, G.A., "An algorithm for machine perception of synthetically significant rings in complex cyclic organic structures", Journal of the American Chemical Society **94** 460-465 (1972)

186. Fugmann, R., Nickelsen, H., Nickelsen, I., Winter, J.H., "Representation of concept relations using the TOSAR system of the IDC. Treatise III on Information Retrieval Theory", Journal of the American Society for Information Science **25** (5) 287-307 (1974)

187. Fugmann, R., Nickelsen, H., Nickelsen, I., Winter, J.H., "TOSAR - a topological method for the representation of synthetic and analytical relations of concepts", Angewante Chemie International Edition **9** (8) 589-95 (1970)

188. Dittmar, P.G., Stobaugh, R.E., Watson, C.E., "The Chemical Abstracts registry system. 1. General design", Journal of Chemical Information and Computer Sciences **16** (2) 111-121 (1976)

189. Mockus, J., Isenberg, A.C., Vander Stouw, G.G., "Algorithmic generation of Chemical Abstracts index names. 1. General design", Journal of Chemical Information and Computer Sciences **21** (4) 183-95 (1981)

190. Hoare, C.A.R., "Recursive data structures", International Journal of Computer and Information Studies, **4** (2) 105-132 (1975)

191. Burton, W., "Generalised recursive data structures", Acta Informatica **12** 95-108 (1979)

192. Kinsella, J.E., "An Interactive Test Editor for GENESIS, a Generic Chemical Structure Information System", Unpublished M.A. Dissertation, University of Sheffield (1982)

193. Nishida, F., Takamatsu, S., "Structured-information extraction from patent-claim sentences", Information Processing and Management **18** (1) 1-13 (1982)

194. Frome, E., "Effects of information storage and retrieval techniques and computers on problems of patentability", Journal of Chemical Documentation **6** (2) 66-71 (1966)

195. Blick, A.R., "Computer-assisted chemical synthesis packages: is this a new problem in patentability?", Journal of Information Science **1** 227-229 (1979)