tached to a ring system. The attached ring system may be any of the aromatic ring systems recognized above. Substituents attached to the chain are named by using Greek characters as locants. These need to be written out in full because of the lack of Greek characters in the ASCII character set and on terminal keyboards. An example of a conjunctive name recognized by the software is beta,delta-dimethyl-1-naphthalenevaleric acid.

Radicofunctional Nomenclature. In general, substitutive nomenclature is preferred to radicofunctional nomenclature, but the latter is included to demonstrate the versatility of the grammar-based technique in recognizing different forms of nomenclature. The Blue Book describes the use of radicofunctional nomenclature in Rules C-21-24. Both monovalent and divalent functional groups are recognized, with the radical part being an aliphatic chain radical or a phenyl group. The monovalent functional groups recognized include halides, alcohol, and thiol; the divalent functional groups include ketone, sulfone, and ether.

CONCLUSION

A context-free, phase structure grammar has been developed for the IUPAC systematic organic chemical nomenclature. While the grammar is incomplete, a substantial part of organic nomenclature is covered, and the description of how the grammar was developed should enable others to devise rules to cover more parent structures, principal groups, and substituents.

The capability of the grammar-based approach to cover a variety of nomenclatural forms is demonstrated by the inclusion of not only substitutive nomenclature but also examples of conjunctive and radicofunctional nomenclature and some semisystematic and trivial nomenclature.

The benefits of a formal grammar for any language are unambiguity and facilitation of machine processing. It is hoped that the presentation of this grammar for IUPAC systematic nomenclature and the description of the substantial quantity of work that this has entailed may lead to the publication in the future of nomenclature rules in a more amenable form for computer processing.

Further papers in this series develop the use of the grammar described here for the automatic recognition and translation of chemical nomenclature.

ACKNOWLEDGMENT

We gratefully acknowledge funding of this research by the Laboratory of the Government Chemist and advice on the use of IUPAC systematic nomenclature received from E. W. Godly and I. Cohen.

Supplementary Material Available: Current grammar for the IUPAC systematic organic chemical nomenclature (14 pages). Ordering information is given on any current masthead page.

REFERENCES AND NOTES

(1) Cooke-Fox, D. I.; Kirby, G. H.; Rayner, J. D. Computer Translation of IUPAC Systematic Organic Chemical Nomenclature. 1. Introduction and Background to a Grammar-Based Approach. *J. Chem. Inf. Comput. Sci.* (first of three papers in this issue).
(2) International Union of Pure and Applied Chemistry. *Nomenclature of Organic Chemistry, Sections A-F and H*; Pergamon: Oxford, U.K., 1979.
(3) Cooke-Fox, D. I.; Kirby, G. H.; Rayner, J. D. Computer Translation of IUPAC Systematic Organic Chemical Nomenclature. 3. Syntax Analysis and Semantic Processing. *J. Chem. Inf. Comput. Sci.* (third of three papers in this issue).
(4) Garfield, E. An Algorithm for Translating Chemical Names to Molecular Formulas. In *The Awards of Science and Other Essays*; ISI Press: Philadelphia, 1985.
(5) Thomas, M. I. *A Basic SLR Parser-Generator*; Report No. 80/3; Department of Computer Science, University of Hull: Hull, England, 1980.
(6) Aho, A. V.; Ullman, J. D. *Principles of Compiler Design*; Addison-Wesley: Reading, MA, 1978; pp 204-214.
(7) Rayner, J. D. Grammar Based Analysis by Computer of the IUPAC Systematic Chemical Nomenclature. Ph.D. Thesis, University of Hull, Hull, England, 1983.
(8) Cooke-Fox, D. I. Computer Translation of IUPAC Organic Chemical Nomenclature to Structure Diagrams. Ph.D. Thesis, University of Hull, Hull, England, 1987.
(9) The full grammar is given as an Appendix to this paper in the microfilm edition of the Journal.

Computer Translation of IUPAC Systematic Organic Chemical Nomenclature. 3. Syntax Analysis and Semantic Processing

D. I. COOKE-FOX, G. H. KIRBY,* and J. D. RAYNER
Department of Computer Science, University of Hull, Hull HU6 7RX, England

Received May 6, 1988

The construction of computer software to translate from IUPAC systematic organic chemical nomenclature into concise connection tables (CCTs) is described. A parser to perform the syntax analysis phase was produced by application of a modified SLR parser generator to the context-free grammar developed in the second paper of this series. Semantic processing of names accepted by the parser involves a second pass through the name by the established path. Semantic information associated with the morphemes, in the form of CCT fragments, is extracted from the dictionary and used to build the CCT. Data structures are described that are used to check the chemical validity of the CCT, and hence the name, and to ensure alphabetic ordering of substituent prefixes.

INTRODUCTION

Part 1 in this series[1] described the background to this project in which computer techniques and software have been applied and developed to translate IUPAC systematic organic chemical nomenclature into concise connection tables and hence to displays of the corresponding structure diagrams. The grammar-based approach that was adopted required first the development of a formal grammar for IUPAC systematic organic nomenclature, and such a context-free phrase structure grammar for a number of classes of compounds was reported in part 2.[2]

Having a context-free grammar allows application of the process commonly used for the translation, or compilation, of computer programs into machine or similar instructions. The process of compilation is complex but can be considered a series of phases, namely, lexical analysis, syntax analysis or parsing,

0095-2338/89/1629-0112$01.50/0

semantics and code generation.

In lexical analysis, the input string (a program in the case of compilation, but here a chemical name) is split into valid fragments and corresponding syntactic tokens are fed to the syntax analysis phase. The possible tokens that appear in a chemical name are the fragments defined in the nomenclature rules, and they can be associated with the syntactic tokens by dictionary look-up.

In syntax analysis not only are valid sentences of a language recognized but also the underlying structure of the sentence can be determined. An efficient parser can be automatically constructed from a context-free grammar by a software tool called a parser generator.[3] A number of such tools are in existence for the various parsing techniques that have been used to write compilers, including operator precedence, recursive descent, and LR, LL, and LALR parsing. We use a simple LR parser generator, SLR, based on an algorithm by Aho and Ullman[3] that we have extended from its initial implementation[4] to support various requirements of the present work.

A sentence accepted by a parser is one that belongs to the language described by its underlying grammar. That is, in the case of nomenclature, acceptance depends at this stage solely on syntactic form, and no meaning can as yet be ascribed to the name. Thus, no statement can be made by the parser as to the chemical validity of a name it accepts. This is the concern of the semantic phase where the validity of a name is determined by the structural information implied by the name and its correctness according to the laws of chemistry. A systematic name is composed of fragments specially selected to describe the structural features of the name. So, if the rules that govern the assembly of this structural information are known, then a structure diagram can be produced from any systematic name.

In any given language, fragments of a sentence that may be ascribed some individual meaning are termed "morphemes". However, the structural information associated with each systematic name morpheme must be represented within the computer system and assembled into a complete representation before the structure diagram can be drawn from it. This corresponds to the code generation phase of a compiler. While fragment codes[5] are able to describe the structural meaning of morphemes, they do not give the required information on the association of these structural fragments in the molecule. We preferred to use the concise connection table (CCT) that was originated by Rayner[6] specifically as a solution to this problem. Each entry or group of entries in the CCT depicts the structure for a morpheme and also describes the way in which this structural fragment is connected to others to form the whole structure.

IUPAC systematic organic nomenclature includes some trivial names that are still in common usage. However, the structure diagrams for these names cannot be derived in the manner outlined above since a trivial name is merely a label and contains no inherent structural information. Nevertheless, the structure diagram of a semisystematic name can be derived if the structure of the entire trivial part is represented within the grammar as for a systematic fragment.

The details in the next section explain the capabilities that we have incorporated in the parser to cope with the variety of nomenclatural constructions. An appreciation of these is necessary for the discussion, in the following section, of structure elucidation by semantic processing and, in a future paper, of error detection and correction.

SYNTAX ANALYSIS

SLR Parsing Process. LR parsers, of which SLR is a derivative, are so named because they scan the input from left to right and form a rightmost derivation in reverse. In contrast to typical computer-programming languages, for which the LR method was originally devised, the IUPAC nomenclature is a right-rooted language—that is, the main structure-determining parts of a name occur toward the end of the name, whereas in a typical computer program each construct is introduced by a unique keyword. Since chemical names are relatively short compared to programs (a few hundred characters at most, as against many thousands), an input name may be absorbed entirely before analysis and processed from right to left. Thus, the grammars developed and illustrated in this paper appear to represent names written "backward". However, actual input is presented in the normal way.

```
TERMINALS

     A-MARK,  EN-MARK,  ANE-MARK,  ENE-MARK,  YNE-MARK,
     ROOT  ;  LOCANT ;  HYPHEN  ;  COMMA   ;  MULT  ;


RULES

     NAME = ANE-MARK  ROOT  ,
            SAT-SEQ  ROOT   LOC-PART  ;

     SAT-SEQ = ENE-MARK  ENE-EXT  ,
               YNE-MARK  YNE-EXT  ;

     ENE-EXT = MULT  A-MARK , $ ;

     YNE-EXT = MULT  YNE-EN ,
               LOC-PART  HYPHEN  EN-MARK  ENE-EXT ,
               $ ;

     YNE-EN = A-MARK ,
              LOC-PART  HYPHEN  EN-MARK  ENE-EXT ;

     LOC-PART = HYPHEN  LOCANT  LOC-SEQ ;

     LOC-SEQ = COMMA  LOCANT  LOC-SEQ , $ ;


ROOTSYMBOL   NAME
```

Figure 1. Simple SLR grammar for hydrocarbons.

LR parsers accept an input stream of grammar symbol tokens and use a stack and a parsing table to control their operation. The input tokens are derived from the raw source by lexical analysis, and in principle the stack contains a string

$$S_0X_1S_1X_2X_2S_2 \ldots X_nS_n$$

where S_n is at the top of the stack. Each X_i is an input token and each S_i is a state. It is not always necessary to have the input tokens on the stack, and they are not included within the nomenclature system.

Intuitively, each state corresponds to the parser being at a certain point in the processing of a rule of the grammar: as parsing proceeds, the parser moves between states. The current state value is used as one index to the parsing table, of which there are two parts, an Action table and a Goto table. A further table summarizes the grammar rule lengths and their left-side nonterminal symbol codes.

Figures 1–3 illustrate a simple grammar for unbranched hydrocarbons and its associated parser tables. It is not feasible in the space available here to demonstrate the operation of the technique on a meaningful subset of the actual grammar described in part 2.[2] Therefore, the example given here is illustrative only. Grammars accepted by the SLR parser generator are formed into three sections (see Figure 1):

(a) Terminals. These are the terminal symbols of the grammar, each representing possibly a number of different actual input morphemes (e.g., ROOT could be meth, eth, prop, etc.).

(b) Rules. The syntax rules of the grammar are presented in the form

$$L = a, b, c;$$

	A-MARK	EN-MARK	ANE-MARK	ENE-MARK	YNE-MARK	ROOT	LOCANT	HYPHEN	COMMA	MULT	E-O-I
0			S 28	S 26	S 11						
1											A
2						S 3					
3								S 5			
4											R 1
5							S 6				
6								R 13	S 8		R 13
7								R 12			R 12
8							S 9				
9								R 13	S 8		R 13
10								R 14			R 14
11						R 7		S 5		S 19	
12						R 3					
13								S 14			
14		S 15									
15						R 5				S 17	
16						R 8					
17	S 18										
18						R 6					
19	S 25							S 5			
20						R 9					
21								S 22			
22		S 23									
23						R 5				S 17	
24						R 10					
25						R 11					
26						R 5				S 17	
27						R 4					
28						S 29					
29											R 2

Figure 2. Action table for the grammar of Figure 1.

	NAME	SAT-SEQ	ENE-EXT	YNE-EXT	YNE-EN	LOC-PART	LOC-SEQ
0	1	2					
1							
2							
3						4	
4							
5							
6							7
7							
8							
9							10
10							
11				12		13	
12							
13							
14							
15			16				
16							
17							
18							
19					20	21	
20							
21							
22							
23			24				
24							
25							
26			27				
27							
28							
29							

a

Rule Number	Leftside Nonterminal	Rightside Length
1	NAME	3
2	NAME	2
3	SAT-SEQ	2
4	SAT-SEQ	2
5	ENE-EXT	0
6	ENE-EXT	2
7	YNE-EXT	0
8	YNE-EXT	4
9	YNE-EXT	2
10	YNE-EN	4
11	YNE-EN	1
12	LOC-PART	3
13	LOC-SEQ	0
14	LOC-SEQ	3

b

Figure 3. Goto table (a) and rule table (b) for the grammar of Figure 1.

where L is a single nonterminal symbol and the various sequences a–c, etc. are sequences of terminal and nonterminal symbols, which are the rule alternatives of L. The various alternatives for L are separated by commas, and the final alternative is followed by a semicolon.

(c) Rootsymbol. The rootsymbol of the grammar denotes the complete utterance of the language (sentence, name) with which the parser generator begins to analyze the grammar and construct the parser tables. It is the goal symbol of the parser when analyzing a name.

Thus, in Figure 1, there are 10 terminal symbols and 14 rule alternatives for 7 nonterminal symbols. The $ symbol as a rule alternative indicates that the rule may be null; that is, its corresponding fragments need not appear in a name at all.

Figure 2 presents the Action table for the grammar of Figure 1, showing what the parser must do for each state and input terminal symbol [including end-of-input (E-O-I)]. Where no action is specified, an input symbol is not valid in the given state; otherwise, A indicates that a complete input sequence may be accepted as valid, S n denotes adding the symbol to the Stack and moving to State n (technically called Shifting), R n denotes Removing symbols from the stack according to Rule n (Figure 3b) and then adding the left-side symbol of that rule to the stack and moving to a state given by the Goto table (Figure 3a), as if that left-side symbol had been directly input for the state in force when the first actual symbol of the rule was received (technically called Reducing). This process is illustrated in Figure 4 for the input sequence "2-butene".

Lexical Analysis. The parsing process depends upon the ability to determine correctly which terminal symbol corresponds to each morpheme of the input string and to identify those morphemes correctly as well. For the typical modern programming language, an SLR-based parser is able to process an input program in a single left to right pass, since each terminal symbol can be determined uniquely from well-

Stage	Stack	State	Input	Action	Leftside	Length	Old State	Goto
0	<empty>	0	ENE-MARK (ene)	S 26				
1	ENE-MARK 0	26	ROOT (but)	R 5	ENE-EXT	0	26	27
2	ENE-MARK 0 ENE-EXT 26	27	ROOT (but)	R 4	SAT-SEQ	2	0	2
3	SAT-SEQ 0	2	ROOT (but)	S 3				
4	SAT-SEQ 0 ROOT 2	3	HYPHEN (-)	S 5				
5	SAT-SEQ 0 ROOT 2 HYPHEN 3	5	LOCANT (2)	S 6				
6	SAT-SEQ 0 ROOT 2 HYPHEN 3 LOCANT 5	6	E-O-I (end)	R 13	LOC-SEQ	0	6	7
7	SAT-SEQ 0 ROOT 2 HYPHEN 3 LOCANT 5 LOC-SEQ 6	7	E-O-I (end)	R 12	LOC-PART	3	3	4
8	SAT-SEQ 0 ROOT 2 LOC-PART 3	4	E-O-I (end)	R 1	NAME	3	0	1
9	NAME 0	1	E-O-I (end)	A				

Figure 4. Example SLR parse for the name 2-butene.

delimited units of the program text. However, chemical nomenclatures differ from programming languages in this respect, since they generally contain few delimiting characters, in particular between letter-based morphemes. Whereas the space is used frequently in programs to separate adjacent symbols, chemical names contain strings of directly concatenated letters that represent a number of functionally different morphemes.

This lack of delimiters necessitates a method for recognizing morphemes within the input character string and associating each morpheme with a corresponding terminal symbol. To this end, the SLR parser generator has been extended to allow relevant morphemes to be listed against each terminal symbol in the overall grammar. These morphemes are then processed to construct a dictionary table that associates each morpheme with one or more terminal symbols as specified. The parser then reads this table to build a tree-structured dictionary before accepting names for analysis and translation.

For each valid morpheme within the tree dictionary, there is a pointer to a list of appropriate terminal symbol codes. On input of a name to the parser, the lexical analysis phase matches input characters against the tree to find the largest available morpheme. Terminal symbol codes for this morpheme are then checked against the Action table to find one that is valid. If no symbol is valid, a shorter morpheme may be chosen instead.

Backtracking. The complexity of the nomenclature is such that the parser may reach a point at which no valid morphemes may be available, although alternative shorter morphemes could have been chosen earlier. Since the entire name is absorbed before analysis begins, it is possible to backtrack the parser to such an earlier choice and then advance on a different path.

As a simple example, the morphemes in the name iododecane are "iodo", "dec", and "ane". "dodec" is also a valid morpheme in the dictionary and will be found at the same time as "dec" (we are parsing the name from right to left), but as "dodec" is the larger morpheme, it will be chosen in preference to "dec". The parser will then require further input from the lexical analyzer that will find "io" in the input stream. There is no entry for "io" in the dictionary, and so the parse fails and backtracks to find an alternative path. "dec" is an alternative morpheme that leads to the valid morpheme "iodo", and thus the name parses correctly.

Normally an SLR parser is deterministic; that is, it does not backtrack. For chemical nomenclature, due to the established nature of the morphemes, there may be a number of alternative terminal symbols that could be used as input to the parser. It has proved possible to augment the SLR method to allow backtracking and so overcome the lack of delimiters within chemical nomenclature.

If a name is input that is incorrect according to the IUPAC syntax, or that is correct but not known in the language described by a partially developed grammar, then the parser will enter an error state. The parser will attempt to backtrack and to retry alternatives but will ultimately fail, leaving the parser at the right-hand end of the name with no further alternatives to try. The name will be rejected at this point, and it has been found empirically that by marking out the leftmost position reached that led to an error state, an indication is given as to the type of error in the name. Therefore, a record is kept of the maximum extent of the parser through the input string, and this is used to suggest the position and extent within the name of the fragment that led to the error.

Where the parser successfully reaches the end of the input name, and the Action table indicates the name is acceptable, the sequence of terminal symbols identified for the name is recorded as a (syntactically) valid interpretation, and semantic processing can begin.

SEMANTICS

Semantic Information and the Grammar. Semantic information is held within the grammar alongside each appropriate morpheme. A morpheme in the dictionary has one of four types of information associated with it: (1) information relating to structure and given by one or more CCT records; (2) an integer that may be the representation of the length of a chain, a multiplication factor for a multiplying prefix, or the locant value for a Greek locant, depending on the interpretation of the morpheme; (3) an integer that specifies the atomic number of an atom entry for which a CCT record must be constructed when needed by using a copy of the integer value in the SIZE field of the record; (4) a null entry, the fragment having no structural meaning.

For example, the morpheme "but" that is used to construct butane has an integer value 4 associated with it. At a later stage the value is extracted and expanded by the appropriate semantic code to the CCT fragment "1 1 4 0". Similarly, the morpheme "chlor" is associated with the value 17, which is incorporated by different semantic actions into the CCT fragment "1 2 17 0". Some morphemes, for example, hyphens, commas, and parentheses, have no structural meaning in

themselves and so have no semantic information associated with them.

The semantic information associated with each of the morphemes only becomes available for semantic processing if the terminal symbol that represents the morpheme is shifted onto the parser's stack during the parsing of a name. As parsing may involve a number of backtracks and retries, the construction of the CCT is held back until the final path has been found. This path is followed again without the need to backtrack, and the semantic information is extracted to enable the building of the CCT. This semantic information is extracted from the dictionary during the Shift actions and then manipulated during the Reduce actions of the parser. Thus, the construction of the CCT has a significant influence on the way the syntax rules are written in that the Shift and Reduce actions take place in an order which ensures that information is not lost.

The Pascal code associated with each production rule alternative was placed into the grammar alongside the appropriate rule alternative and executed when the rule is processed. The rules are numbered by the SLR parser generator, and these numbers are used as Pascal case selector values. The SLR parser generator was amended to extract the Pascal code and put it into a file with the correct rule numbering ready for automatic inclusion into the main parse program on compilation. On any subsequent change to the grammar, the code relating to rules already present does not need to be changed, and on generating a new parsing table, the semantic rules are automatically renumbered and the case limbs relabeled.

Constructing the CCT. A chemical name is hierarchical with its substructures preceding the parent term. Each substructure in turn may have its own substructures that precede it. For example

6-(1-chlorobutyl)-8-(2-iodopropyl)tridecane

has the parent structure tridecane, with two substructures attached to it, butyl and propyl. Each of these substructures has a single substructure or substituent, chloro on the butyl and iodo on the propyl. This hierarchy extends from the rear of the name forward.

The same hierarchy is seen in the CCT for the above compound:

LOCT	TIPE	SIZE	SUBS
1	1	13	2
6	1	4	1
1	2	17	0
8	1	3	1
2	2	53	0

The first entry represents the aliphatic chain (TIPE = 1) tridecane (SIZE = 13) and has two substituents (SUBS = 2). The first substituent, the second entry, on carbon 6 of the parent (LOCT = 6) is the aliphatic chain butyl with a single substituent given by the following entry. Thus, the chloro, an atom of atomic number 17 (TIPE = 2, SIZE = 17), is on position 1 of its parent and has no substituents. Similarly, the final two entries represent the second complex substituent of the parent.

The hierarchy can be nested to many levels and may result in any number of CCT entries. It is far easier to construct the CCT from the top down, by parsing the name from right to left, than from the bottom up. However, the hierarchy in the CCT refers to parent and its immediate substructures rather than the right to left order in the name. Substructures appear in the CCT in locant order so, in the example given above, the chlorobutyl substructure with locant 6 appears in the CCT before the iodopropyl substructure (see Sorting the CCT).

As the CCT is being constructed, some checks can be made as to the structural validity of the name that has been input. It is not feasible to attempt to trap all the errors by syntactic means. The final grammar[7] as given in part 2 will accept the name pent-1,3-triene as a syntactically correct name, although it is not valid on two counts. First, the number of locants does not match the multiplying term and, second, an "a" is not present after the aliphatic stem.

The first error can be uncovered in the building of the CCT. Having shifted the terminal symbol that represents the morpheme "tri-mark", the numerical value of the multiplying term is recorded. If no multiplying term is present, the value 0 is recorded. On reading the locants, a count is kept of the number of locant values, and at the end of a locant sequence this total is compared with the original multiplying value. If they are not equal, an error flag is raised.

The nonpresence of the "a" in the name is allowed by the rule alternative

a-part = a-mark, $;

The semantic code for the rule "a-part = a-mark" requires explicitly that an "a" is present within the name and will give a warning message if the recorded multiplying value is 0. Alternatively, the code associated with the rule "a-part = $" will give a warning if this value is not 0.

Checking the Validity of the CCT. Two essential checks must be done on the CCT to confirm that the structure represented is chemically valid. First, the locants of substituents and modifications should be within the bounds of their parent structure and, second, the maximum valency of each atom within the structure should not be exceeded.

The above two checks require the examination of each of the parent structure CCT main entries with their respective substituent and modification entries. While as discussed above there is an implied hierarchy in the CCT, there is a problem in finding the position of the CCT main entries for substituents at the same hierarchical level, since their position is dependent upon entries for substituents and modifications of preceding substituents of the parent.

A means of simplifying semantic checks is to state the hierarchy of the chemical name explicity, so that each main entry is directly referenced from the main entry immediately above it in the hierarchy, rather than the implied hierarchy of the CCT. A semantic tree has been developed that allows the explicit description of the hierarchy of a chemical name using CCT fragments. It is the semantic tree, rather than the CCT directly, that is constructed by the second pass of the parser through the name.

The semantic tree consists of nodes arranged in the appropriate hierarchy, each of which points to CCT records to describe the actual structure. Each semantic tree node has seven fields, three of which are concerned with bidirectional linkage within the tree. The remaining four fields are pointers to groups of CCT entries concerned with the parent and any unsaturations, heteroatoms, and functional groups. The root of the semantic tree is a node that contains a pointer to a CCT main entry record for the parent compound. In the simplest of all semantic trees, the parent pointer points to the CCT main entry record of the parent compound with all the other pointers being NIL. Figure 5 shows the semantic tree for a name with several levels of nesting of substituents and containing unsaturations, namely

4-bromo-7-[1-(1-chloroethyl)-2-methylprop-1-enyl]undeca-1,5,9-triene

With the semantic information organized in this tree, the checking of locant values and correct valencies is straightforward. All the CCT main entries are in known positions, as are the substituent and modification entries that belong to them.

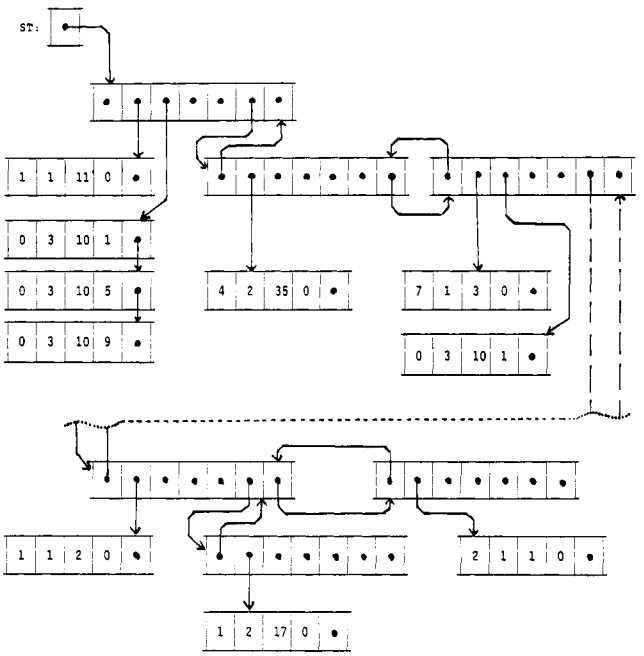

Figure 5. Example semantic tree for the name 4-bromo-7-[1-(1-chloroethyl)-2-methylprop-1-enyl]undeca-1,5,9-triene. Semantic tree node fields are, from left to right, PREV, PARENT, UNSATURATED, HETERO, SEFG, SUBSTITUTED, and NEXT, as outlined in the text. CCT records have fields LOCT, TIPE, SIZE, SUBS, and NEXT, from left to right.

In most cases the maximum locant value allowed by a parent structure or substructure is given by the CCT entry for that structure, but there are some cases where the size of the structure cannot be deduced from the normal CCT entry alone. For example, the size of a bridged alicyclic system is given by the addition of the SIZE fields of the main entry record with the SIZE fields of the subsequent chain–ring segment entries. These CCT entries will be clearly grouped together below a single semantic tree node for the parent structure.

The locants of the substituents may be found in the LOCT field of the main entry records. For bond modifications, the locant values are found in the SUBS field except for a double-bond modification on a bridge head where both the start and end locants are given in the following CCT entry. Hence, it is a simple task to check actual locant values against the range permitted by the parent structure.

The semantic tree allows the value of the SUBS field of each main entry to be calculated from the number of substituents and modifications on that entry. Before this structure was adopted, a count had to be kept of the number of substituents and modifications being added to each of the main entries as the CCT was being constructed.

By knowing the maximum valency of each atom type, the validity of substitutions may be checked. For each of the parents and substructures that may be substituted, assumptions have been made about the number of substituents allowed at each locant position. The actual number of substituents present can easily be checked within the semantic tree.

Alphabetic Ordering of Substituent Prefixes. The ordering of the substituent prefixes within a chemical name is important. If a chemical compound is to have a single systematic name, then there must a methodology to order the prefixes. In the first edition of the IUPAC organic chemical nomenclature rules this was governed by the complexity of the substituents. This has been replaced by the alphabetic ordering of the substituents, the rules governing the ordering being Rules C-16.1–3 in the Blue Book.[8]

While alphabetic ordering is a simple matter, the question of what characters of the substituent prefixes it is applied to is not. Simple substituent prefixes are arranged alphabetically, with multiplying affixes (if any) being added later. These do not therefore alter the alphabetic order of the prefix. Prefixes for subsubstituents are arranged in alphabetic order among themselves in the substituted substituent in the same way as are the substituents of a parent compound; then the complete names of the substituted substituents are treated as single, complex prefixes to the name of the parent compound. However, complex substituents are considered to begin with the first letter of the complete name of the prefix. If that begins with a multiplying term, then it is alphabetically ordered according to the first letter of that term. For example, the correct alphabetic ordering is

3,5,9-trichloro-6-(2,2-dibromobutyl)hexadecane

where the "c" in trichloro comes before the "d" of dibromo.

A check for the correct alphabetic ordering of substituent prefixes in an input name is done in the semantic phase. A warning is given if the order is incorrect, though the order of the entries in the eventual CCT and the meaning of the name are unaffected.

The name is stored in an array on input to the nomenclature translator. During the second pass through the parser, the indices to the prefixes within this array are extracted. Since the correct order of the prefixes is dependent upon the hierarchy within the name, alphabetic checking relies upon similar information as is used in constructing the semantic tree. In practice, the checking is facilitated by the separate construction of a substituent tree with two types of node to represent simple and complex substituents. Fields in each node hold indices to the start of the locant sequence and the first and last characters of the substituent in the name array, a pointer to the next node at the same hierarchical level, and, in the case of complex substituents, a pointer to a node for the first of any subsubstituents.

During the second pass through the name by the parser, a root node for the substituent tree is created when the parent structure is processed. Substituents and their locants, which may be made up of a number of morphemes, are recognized and represented as appropriate by simple and complex nodes. When the tree is complete, comparisons can be made on nodes at the same hierarchical level, starting from the deepest level and working upward until the root node is encountered.

When two simple nodes are compared, it is possible to detect two identical substituents and indicate to the user that the name should be rearranged with insertion of a multiplying term and removal of the duplicated substituent. If incorrect ordering is found, a warning message is given with identification of all terms involved. A comparison of two complex substituents checks first the alphabetic ordering, giving a warning message if the order is incorrect. If the two are identical, then the locants are checked and the substituent with the lower locant appears first in the name.

For the comparison of a simple and a complex substituent, the alphabetic characters are considered. If they are the same up to the point where the simple substituent ends and the complex one continues, then the simple one has precedence. Otherwise, the substituent with the lower character at the first point of difference has precedence. For example, chloro should come before chloromethyl, which precedes chlorosyl.

Sorting the CCT. The final step before a CCT is delivered by the semantic phase is to impose a canonical order. While there is already an order arising from the hierarchy of the table, the specific order of locants and of substituents at the same level depends upon the order in which they appear in the name. However, locants not in numeric sequence and substituents not in alphabetic order do not affect the meaning of the name. It is desirable, though, to generate only one CCT for variations on the same name in order to draw one corre-

sponding structure and to compare different structures by using the CCT.

The CCT entries are ordered by using the semantic tree. The groups of entries corresponding to unsaturations, heteroatoms, and functional groups on one parent structure are sorted by locant order and output directly after the CCT entries for the parent structure represented by that semantic tree node. Each substituent, which may have subsubstituents as indicated by the semantic tree hierarchy, is then sorted first by locant order and then, for entries with the same locant, on the TIPE field. This brings substituents with the same locant and type (chain, rings, atom) together, and these are finally sorted on the SIZE field to order them from smallest first to largest last.

DISCUSSION AND CONCLUSIONS

This paper has described the processing carried out by the parser software in analyzing the syntax and semantics of IUPAC systematic nomenclature for some classes of organic compounds. The software has been successfully implemented in Turbo-Pascal and used within the nomenclature to structure diagram translator, which runs on an IBM PC-XT or compatible microcomputer. It confirms that the grammar-based approach using techniques developed for processing computer-programming languages can be applied to other less artificial languages. Two modifications were necessary, though, to these techniques. Most importantly, backtracking had to be introduced in the syntax analysis phase. Second, reference to a dictionary of valid morphemes was essential in the lexical analysis phase because of the lack of delimiters in chemical nomenclature.

In part 2,[2] a comment was made about the difficulty of developing a formal grammar from nomenclature rules as described in the Blue Book.[8] In this paper a good illustration of the complicated processing necessary to implement the current rules has been given. The rules for the alphabetic ordering of substituents are not only complex but also inconsistent as to which characters alphabetic ordering is applied. This depends upon whether the substituent is simple or complex. It necessitated the implementation of a data structure specifically for this one task, with code to detect simple and complex substituents and perform different processing on the two types.

ACKNOWLEDGMENT

We gratefully acknowledge funding of this research by the Laboratory of the Government Chemist.

REFERENCES AND NOTES

(1) Cooke-Fox, D. I.; Kirby, G. H.; Rayner, J. D. Computer Translation of IUPAC Systematic Organic Chemical Nomenclature. 1. Background and Introduction. *J. Chem. Inf. Comput. Sci.* (first of three papers in this issue).
(2) Cooke-Fox, D. I.; Kirby, G. H.; Rayner, J. D. Computer Translation of IUPAC Systematic Organic Chemical Nomenclature. 2. Development of a Formal Grammar. *J. Chem. Inf. Comput. Sci.* (second of three papers in this issue).
(3) Aho, A. V.; Ullman, J. D. *Principles of Compiler Design*; Addison-Wesley: Reading, MA, 1977.
(4) Thomas, M. I. *A Basic SLR Parser Generator*; Report No. 80/1; Department of Computer Studies, University of Hull: Hull, England, 1980.
(5) Lynch, M. F.; Harrison, J. M.; Town, W. G.; Ash, J. E. *Computer Handling of Chemical Structural Information*; MacDonald-American Elsevier: London, 1971; Chapter 6.
(6) Rayner, J. D. A Concise Connection Table Based on Systematic Nomenclatural Terms. *J. Chem. Inf. Comput. Sci.* **1985**, *25*, 108-111.
(7) See ref 2 Appendix.
(8) International Union of Pure and Applied Chemistry. *Nomenclature of Organic Chemistry, Sections A-F and H*; Pergamon, Oxford, U.K., 1979.

Canadian Scientific Numeric Database Service†

GORDON H. WOOD,* JOHN R. RODGERS, and S. ROGER GOUGH

Canada Institute for Scientific and Technical Information, National Research Council of Canada, Montreal Road, Ottawa, Canada K1A 0S2

Received November 14, 1988

Modern computer systems and telecommunications networks are being harnessed in increasingly innovative ways to deliver evaluated scientific/technical numeric data to the desk or laboratory bench. As an example of this development, the Canadian Scientific Numeric Database Service (CAN/SND) is described. CAN/SND provides international online access to factual databases in crystallography, molecular biology, spectroscopy, and chemical thermodynamics. In addition, CAN/SND carries out research in data storage, retrieval, and analysis techniques. The paper gives a description of the databases currently available, examples showing the variety of scientific questions that can be answered, and an outline of plans for virtually linking related databases for interdisciplinary searching.

I. INTRODUCTION

This paper describes the Canadian Scientific Numeric Database Service (CAN/SND), what it is, what it offers, and where it plans to go. All readers are almost certainly familiar with the computer as a tool for performing calculations and automating measurements; many are aware that computers may be used to search large bibliographic databases. Probably relatively few think of the computer as a means of accessing evaluated scientific data from the international literature. It is this latter use that will be highlighted here.

Immediately following, section II defines some basic terminology and gives some perspective on scientific numeric database systems. Section III reviews the background to CAN/SND and the databases currently offered. Section IV

† Presented at the Herman Skolnik Award Symposium on Scientific Numerical Databases—Present and Future, sponsored by the Division of Chemical Information of the American Chemical Society at the Third Chemical Congress of North America, Toronto, Canada, June 7, 1988.

* Author to whom correspondence should be addressed.

0095-2338/89/1629-0118$01.50/0 Published 1989 by the American Chemical Society