107	Aerospace facilities and techniques B76
108	Earth sciences B77
109	Sonics and ultrasonics B78
110	Electrical. Power Systems and Applications B80
111	Electrical. Power networks and systems B81
112	Electrical. Generating stations and plants B82
113	Power apparatus and electric machines B83
114	Direct energy conversion and energy storage B84
115	Electrical. Power utilisation B85
116	Electrical. Industrial application of power B86
117	Computer and Control. General and Management Topics C00
118	General control topics C01
119	General computer topics C02
120	Management topics C03
121	Computer and Control. Systems and Control Theory C10
122	Systems and control theory. Mathematical techniques C11
123	Systems theory and cybernetics C12
124	Control theory C13
125	Computer and Control. Control Technology C30
126	Control and measurement of specific variables C31
127	Control equipment and instrumentation C32
128	Control application C33
129	Numerical Analysis and Theoretical Computer Topics C40
130	Numerical analysis C41
131	Computer metatheory and switching theory C42
132	Computer Hardware C50
133	Computer Hardware. Circuits and devices C51
134	Computer Hardware. Logic design and digital techniques C52
135	Computer Hardware. Storage devices and techniques C53
136	Analogue and digital computers and systems C54
137	Computer peripheral equipment C55
138	Computer Software C60
139	Software techniques and systems C61
140	Computer Applications C70
141	Computer applications. Administrative data processing C71
142	Computer applications. Information science and documentation C72
143	Computer applications. Natural sciences C73
144	Computer applications. Engineering C74
145	Other computer applications C75

BIBLIOGRAPHY

(1) Dickman, J. T.; O'Hara, M. P.; Ramsay, O. B. *Chemical Abstracts. An Introduction to Its Effective Use*; ACS Audio Course No. 52; American Chemical Society: Washington, DC.
(2) Subject coverage and arrangement of abstracts by sections in Chemical Abstracts. Editor American Chemical Society, 1982.
(3) Shinichiro, K. Changes in sections of Chemical Abstracts. *Shikoku Kokenkaiho* **1982**, *33*, 2–6.
(4) Orbit Information Technologies is a Pergamon Infoline Co., which offers online most of the largest scientific databases and patent databases.
(5) Inge Berg, H. Subject compatibility between Chemical Abstracts Subject sections and search profiles used for computerized information retrieval. *J. Chem. Doc.* **1972**, *12*, 110–113.
(6) Peterson, J. S. Replacement of an in-house current awareness bulletin by Chemical Abstracts Section Groupings. *J. Chem. Inf. Comput. Sci.* **1975**, *15*, 169–172.
(7) Dou, H.; Hassanaly, P. Mapping the Scientific network of patent and non-patent documents from chemical abstracts for a fast scientometric analysis. *World Pat. Inf.* **1988**, *2*.
(8) Ganz, C. Bibliometric models for international Science and Technology. *Rev. Fr. Bibl.* **1987**, *2*, 2.
(9) Callon, M. C.; Courtial, J.; Turner, W.; Bauin, S. Problematic networks: an introduction to C-word analysis. *Inf. Sci. Soc.* **1983**, 191.
(10) Turner, W. A.; Chartron, B.; Laville, F.; Michelet, B. Packaging information for peer review: new co-word analysis techniques. *Scientometrics* (submitted for publication).
(11) Jakobiak, F. Maitriser l'information critique. *Les editions d'organisation*; 1988; ISBN 2708108743.

Representation and Matching of Chemical Structures by a Prolog Program

JOSEPH L. ARMSTRONG[‡] and D. BRYNN HIBBERT*

Department of Analytical Chemistry, University of New South Wales, P.O. Box 1, Kensington, NSW 2033, Australia

Received September 8, 1988

A notation for describing chemical structures based on the connectivity of entities in the structure is presented in a Prolog program. Following simple rules the chemical structure is encoded as a Prolog data structure that is called the symbolic name (s-name) of the structure. An algorithm to decide if two different encodings of the same structure represent the same chemical can be used as the basis of a decision procedure to determine if an unknown chemical already exists in a large data base of s-name encoded chemical structures. A transformation on the s-name, the generic or g-name, is described that makes searching in a large data base of unknown chemicals very efficient. The s-name notation has the property that the level of abstraction used in the description may be changed, with common substructures named and descriptions nested to any level. The 35 isomers of C_9H_{20} are used as an example of this method.

INTRODUCTION

The traditional method of finding out about an unknown structure is to transform the structure into a unique name and look up this name in a chemical data base such as *Chemical Abstracts*. The fact that a structure can be converted into a unique name (which is an alphabetic string) means that the abstracts can be ordered with respect to this name. The method breaks down if the name is not unique (i.e., we assume that applying the standard naming rules to a given structure results in a unique name—expressed another way we assume that a given name has exactly one derivation using the rules of naming). There are no correctness proofs for the naming rules. Conventional methods of naming chemicals are complex and error prone. As structures become large and more complex, the corresponding systematic names become more and more difficult to understand. This problem led Silk[1] in 1981

* Author to whom correspondence should be addressed.
[‡] Present address: Computer Science Laboratory, Ericsson Telecom, Stockholm, Sweden S-126 25.

to question further attempts to fully systematize nomenclature when the exact nature of, for example, a Chemical Abstracts Registry Number uniquely determines the chemical and, at the other extreme, a popular name may be widely found in common usage. However, machine-readable line notations are still being developed with a view to packing as much unique chemical information as possible into a name while maintaining a high degree of natural chemical language. The Wiswesser line notation (WLN) is the doyen of these methods.[2] The application of graph theory to chemical notation[3] has revived interest in such methods with, for example, the SMILES language of Weininger.[4] To obtain an absolute configurational description, however, requires an approach such as Wirth's[5] that produces a code representing atoms, bonds, dihedral angles, and orientations of and between all atoms in a molecule. Redundant information is pruned to yield the final name of the molecule. To aid users in describing a structure that can be coded into a canonical form, Abe et al.[6] have shown that 35 codes span many commonly used structures. A different approach, found in GENSAL,[7] is to define a formal language for chemicals with a series of grammatical rules for naming and combining substructures such that the user need not have a detailed understanding of the internal representation of the structure.

We propose a method that uses the power of a fifth-generation computer language to name chemicals and to match chemical structures on the basis of the topology of the structures concerned. In our method we encode a given structure by assigning variables to each of the entities in the structure (we will describe what is meant by entities later in the paper) and by describing the connectivity of the entities. In what follows we will refer to such an encoding of the structure as the s-name (s for symbolic) of the structure.

The s-name of a structure is unambiguous in the sense that it describes a unique chemical structure, but it is not unique according to the rules of naming, though there are no proofs as to the correctness of this assumption. To decide if two different s-names refer to the same structure, we use a simple algorithm that decides if the structures represented by the two different s-names are isomorphic. While the s-name is not unique, a unique name can be formed by transforming the s-name in such a manner that all s-name encodings of the same structure yield the same transformed name. This transformed name we call the g-name (generic name) of the structure. The generic name is not unique (two chemically different structures could have the same g-name) but serves as a key by which the s-names can be ordered. Thus, while a total ordering of s-names does not exist, they can be partially ordered by ordering their g-names. Use of g-names makes storing and accessing all the chemicals in a large chemical data base possible. If they were totally unordered, matching unknown structures would take a prohibitive time. The g-name is thus analogous to the molecular formula of a structure.

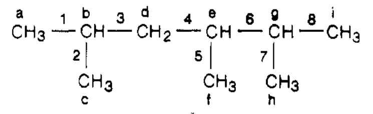
The s-name encoding of a structure has several other advantages, namely, that it becomes an easy matter to define substructures (which are themselves s-names). Transformations can be defined that expand structures or search for the presence of substructures within other structures.

The concept of using graph isomorphic tests for matching chemical structures was propounded by Sussenguth,[8] who determined several tests that could be encoded in some form of machine code for an IBM 7090. It is instructive to compare his algorithm, expressed in a procedural manner, with the Prolog program presented here, in which the algorithm as such consists of a declaration of the nature of similarity with Prolog taking care of the computational details.

We have used the 35 acyclic isomers of C_9H_{20} to illustrate our method. Illustrations of the Prolog algorithms developed in the paper are given as a series of queries (**query1** to **query10**) that may be found in full in Appendix 1.[9,12]

S-NAME OF A STRUCTURE

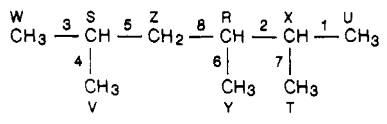
We start by considering the s-name of 2,3,5-trimethylhexane. The first step is to assign to each entity in the structure a unique identifier (here we have used the letters a–g), and then each bond in the structure is labeled (1–8 in the diagram). Next we write down a representation of each



of the bonds in the structure. For example, bond 1 is written ch3(a)/ch(b). The representation of the entire structure we express as the Prolog term[9–12]

```
example_chem(chem1,'2,3,5-trimethylhexane',[
  /* bond 1 is represented as */  ch3(a)/ch(b),
  /* bond 2 is represented as */  ch(b)/ch3(c),
  /* bond 3 is represented as */  ch(b)/ch2(d),
  /* bond 4 is represented as */  ch2(d)/ch(e),
  /* bond 5 is represented as */  ch3(f)/ch(e),
  /* bond 6 is represented as */  ch(e)/ch(g),
  /* bond 7 is represented as */  ch(g)/ch3(h),
  /* bond 8 is represented as */  ch(g)/ch3(i)]).
```

In the program we develop to match structures, the ordering of the entities representing a bond is irrelevant, i.e., A/B means the same as B/A and the order of writing down the bonds in the list of bond descriptors is also irrelevant. The s-name of the structure is the entire list of items represented by the third parameter of **example_chem/3.** Another chemist could have encoded the same structure with the labelings



which would be represented in Prolog thus:

```
example_chem(chem2,'funny_stuff',[
   /* bond 1 */  ch3(U)/ch(X),
   /* bond 2 */  ch(R)/ch(X),
   /* bond 3 */  ch3(W)/ch(S),
   /* bond 4 */  ch3(V)/ch(S),
   /* bond 5 */  ch(S)/ch2(Z),
   /* bond 6 */  ch(R)/ch3(Y),
   /* bond 7 */  ch(X)/ch3(T),
   /* bond 8 */  ch2(Z)/ch(R)]).
```

In the next section we present an algorithm that can be used to decide if these two structures are isomorphic.

Matching s-Names. We define in this section the predicate **same(S1,S2)**, which is true if S1 and S2 are s-names representing the same chemical. Two chemical structures are isomorphic if some permutation of the labeling of one structure yields the labeling of the other structure. The predicate **same** can be defined as follows:

```
same([],[]).
same(S1,S2) :-
   S1 = [A/B|Rest],
     (delete(A/B,S2,Temp);
     delete(B/A,S2,Temp)),
   same(Rest,Temp).
```

The predicate **delete/3** may be found in Appendix 2 along with other standard Prolog terms used in the paper. The predicate **same** works as follows. If the lists representing both structures are empty, then **same** is true. If S1 begins with a bond descriptor A/B, then we try to delete this descriptor from S2, forming a new structure Temp, and recursively apply **same** to what remains of the S1 structure and the structure formed by deleting A/B from the S2 structure. If we cannot delete A/B from S2, we try deleting B/A from the structure since A/B and B/A are alternative representations of the same bond. The built-in backtracking mechanism used by Prolog ensures that when a recursive call to **same** fails, the system will backtrack to the previous application of the delete rule and then continue by trying to delete the next structural pair that matches the given rule. Thus, all possible ways of matching the structures will be tried before either a match is found or it proves impossible to match the two structures.

We can now test our procedure with the following Prolog query (**query1** in Appendix 1)

```
?- example_chem(chem1,_,S1),example_chem(chem2,_
                  ,S2),same(S1,S2),write(S1),nl,write(S2).
```

Prolog replies

```
[ch3(a)/ch(b),ch(b)/ch3(c),ch(b)/ch2(d),ch2(d)/ch(e),
    ch3(f)/ch(e),ch(e)/ch(g),ch(g)/ch3(h),ch(g)/ch3(i)]
[ch3(i)/ch(g),ch(e)/ch(g),ch3(a)/ch(b),ch3(c)/ch(b),
  ch(b)/ch2(d),ch(e)/ch3(f),ch(g)/ch3(h),ch2(d)/ch(e)]
```

If we examine the lists S1 and S2 in more detail, we observe that they represent the same structures since S2 can be formed by permuting the elements of S1 with some of its elements transposed. This can be seen if we write S1 and S2 as follows:

```
S1 = [ch3(a)/ch(b),ch(b)/ch3(c),ch(b)/ch2(d),ch2(d)/ch(e),
          1            2            3             4
      ch3(f)/ch(e),ch(e)/ch(g),ch(g)/ch3(h),ch(g)/ch3(i)]
          5            6            7             8
S2 = [ch3(i)/ch(g),ch(e)/ch(g),ch3(a)/ch(b),ch3(c)/ch(b),
        flip(8)      copy(6)      copy(1)      flip(2)
      ch(b)/ch2(d),ch(e)/ch3(f),ch(g)/ch3(h),ch2(d)/ch(e)]
        copy(3)      flip(5)      copy(7)      copy(4)
```

Here we have numbered the elements of the structure S1 1–8 and numbered the elements of S2 to show where they came from in the structure S1. The annotation "copy(i)" means that the indicated element of S2 is a copy of the *i*th element of S1; "flip(i)" means the indicated element is a transposed copy of the *i*th element of S1. Observe that the initial definition of S2 was

```
S2 = [ch3(U)/ch(X),ch(R)/ch(X),ch3(W)/ch(S),
      ch3(V)/ch(S),ch(S)/ch2(Z),ch(R)/ch3(Y),
                          ch(X)/ch3(T),ch2(Z)/ch(R)]
```

and the reply to the Prolog query was

```
S2 = [ch3(i)/ch(g),ch(e)/ch(g),ch3(a)/ch(b),
  ch3(c)/ch(b),ch(b)/ch2(d),ch(e)/ch3(f),ch(g)/ch3(h),
                                           ch2(d)/ch(e)]
```

Comparing the two, we note that Prolog has satisfied the query by finding a substitution instance of the variables U...Z such that the relation **same(S1,S2)** is true. Comparing the two, we note that the appropriate substitutions are

{R = e, S = b, T = h, U = i, V = c, W = a,
X = g, Y = f, Z = d}

Suppose now that we have a large data base of chemicals and we wish to determine if our unknown chemical is in this data base. Appendix 3 contains an s-name encoding of all 35 isomers that belong to the acyclic C_9H_{20} group. We have called this data base **chem_db/3** to avoid confusion with our previously defined **example_chem/3** examples. To see if our example chemical is in the data base, we make a query of the form (**query2**, Appendix 1)

```
?- example_chem(chem1,_,S1),chem_db(Chem,_,S2),
                          same(S1,S2),write(Chem).
```

to which Prolog (correctly) replies c23, which if we check Appendix 3 is 2,3,5-trimethylhexane.

That no other chemicals in the data base satisfy the matching requirement may be checked with the query (**query3**, Appendix 1)

```
?-findall(Chem,(example_chem(chem1,_,S1),
              chem_db(Chem,_,S2),same(S1,S2)),L).
```

to which Prolog replies

```
        Chem =_57,S1 =_95,S2 =_136,
             L = [c23,c23,c23,c23]
```

findall finds all solutions to a given problem.[10] The result is a list, L, that shows all ways in which the query can be satisfied. In this case the list only contains the item c23, which constitutes a proof that no other chemical in the data base matches our unknown chemical. The fact that c23 occurs more than once implies that the relation **same/2** can be satisfied by using several different substitutions.

A careful reading of this section will reveal a slight difference in the way **chem1** and **chem2** were defined. **chem1** was defined by using terms like ch2(a)/ch(b), whereas **chem2** used terms like ch2(A)/ch2(B)—the first case uses Prolog constants to denote the different entities in the structure, the second uses Prolog variables. In what follows and in Appendix 3 all chemical expression will be encoded by using Prolog variables. The use of constants and variables in the first example was to avoid problems in explaining the use of Prolog's "anonymous" variables.

Improving the Efficiency of the Search. The efficiency of the search procedure defined by the predicate **same/2** can be improved by normalizing the s-name of the structure. The first step in normalizing the structure is achieved by lexically ordering the entities in the bond descriptors; i.e., a term of the form A(...)/B(...) is considered to be in normal form if A @< B, where @< represents the lexical "less than" operator. This relation can be expressed

```
    order(A/B,A/B) :-
        functor(A,Af,_),
        functor(B,Bf,_),
      Af @< Bf.
    order(A/B,B/A).
```

If all the bond descriptors have been ordered, then we can omit the "or" test in the original version of the predicate **same**. A new version **same1** can be defined as

```
    same1([],[]).
    same1([H|T],S2) :-
      delete(H,S2,Temp),
      same1(T,Temp).
```

which is more efficient than **same/2**, since there is no doubt in which order the A/B pairs occur in both structures.

At this point it is interesting to note that the predicate **same1** is nothing other than a slightly disguised version of a permutation generator. See, for example, the predicate **permutation2** described in section 3.2.6 of Bratko.[10] This could be illustrated by the query

```
        ?- findall(X,same1(X,[a,b,c]),L)
```

and the reply

```
  X =_61,
  L = [[a,b,c],[a,c,b],[b,a,c],[b,c,a],[c,a,b],[c,b,a]]
```

In general, to prove that two structures are isomorphic, we have to compute all possible permutations of the labelings of one of the structures and compare them with the labelings of the other structure. Since there are factorial *N* ways of permuting a structure having N labels, we have to find ways of reducing the number of possible legal permutations that could possibly yield a correct relabeling of the structure concerned.

The search length can be considerably shortened by rearranging the elements in the s-name. In a normalized s-name the order of elements is formed by sorting the elements in the descriptor list with respect to a key that is formed from the normalized bond descriptor name. For example, the structure [d(i)/b(a),a(i)/h(k),c(i)/b(o)] becomes [a(i)/h(k),b(o)/c (i),b(a)/d(i)] because a/h @< b/c @< b/d and a @< h and b @< c and b @< d.

The procedure **normalize_structure** performs this normalization:

```
normalize_structure(X,Y) :-
    maplist(order,X,T),
    maplist(add_tag_key,T,T1),
    keysort(T1,T2),
    maplist(remove_tag_key,T2,Y).
add_tag_key(X,Fa/Fb-X) :-
    X = A/B,
    functor(A,Fa,_),
    functor(B,Fb,_).
remove_tag_key(X-Y,Y).
```

Applied to our original example (2,3,5-trimethylhexane), we make the following query (**query4**, Appendix 1):

```
?- normalize_structure(
    [ch3(a)/ch(b),ch(b)/ch3(c),ch(b)/ch2(d),
    ch2(d)/ch(e),ch3(f)/ch(e),ch(e)/ch(g),
    ch(g)/ch3(h),ch(g)/ch3(i)],S),
write(S).
```

Prolog replies with the normalized structure

```
[ch(g)/ch(e),ch(b)/ch2(d),ch(e)/ch2(d),ch(b)/ch3(a),
    ch(b)/ch3(c),ch(e)/ch3(f),ch(g)/ch3(h),ch(g)/ch3(i)]
```

If all structures are normalized before they are added to the data base, the possible search depths required to establish if two structures are equal will be shorter than if the structures were not normalized. This is because the matching process will "fail early" in the case where the structures are not isomorphic.

The ordering implied in the construction of the s-name is essentially the principle used in Sussenguth's[8] algorithm—namely, that if two graphs are isomorphic, then some property of some subset of the nodes in one graph must correspond to the same property of a subset of the nodes in the other graph.

GENERIC NAMES

The generic name of a structure is formed as follows. First, the variable names are removed from the normalized name. Thus

```
[ch(g)/ch(e),ch(b)/ch2(d),ch(e)/ch2(d),ch(b)/ch3(a),
    ch(b)/ch3(c),ch(e)/ch3(f),ch(g)/ch3(h),ch(g)/ch3(i)]
```

becomes

```
[ch/ch,ch/ch2,ch/ch2,ch/ch3,ch/ch3,ch/ch3,ch/ch3,
                                              ch/ch3]
```

The number of times identical terms occur in the list are counted [1:ch/ch,2:ch/ch2,5:ch/ch3], and all elements in the list are formed into a single name, which in this case is 1:ch/ch-2:ch/ch2-5:ch/ch3.

The code for this is

```
gname(Structure,G_name) :-
    normalize_structure(Structure,Normalized_structure),
    make_name(Normalized_structure,G_name),!.
```

(**make_name/2** is defined in Appendix 2). We can test **gname** with the following query (**query5**, Appendix 1):

```
?- gname(
    [ch3(a)/ch(b),ch(b)/ch3(c),ch(b)/ch2(d),
    ch2(d)/ch(e),ch3(f)/ch(e),ch(e)/ch(g),
    ch(g)/ch3(h),ch(g)/ch3(i)],S),
write(S).
```

Prolog replies 1:ch/ch-2:ch/ch2-5:ch/ch3. To test the effectiveness of the g-name as a selection key, we have taken all 35 acyclic C_9H_{20} compounds and computed their g-names. Table I is computed by **query6**, Appendix 1. Examination of Table I reveals that 28 different g-names have been generated. This figure of 28 can be subdivided as follows: number of chemicals with unique g-name, 21; number of instances where two chemicals had the same g-name, 7; number of instances where more than two chemicals had the same g-name, 0. If the g-name is used as a screening factor to determine which of the C_9H_{20} homologues could possibly match an unknown C_9H_{20} structure, in the worse case we may have to perform two similarity tests; 60% of the time a single test would suffice and on average 1.4 tests (0.6 chance of one test plus 0.4 chance of two tests) would be needed. Since the g-name for a given structure can be easily computed, and since it is an alphabetic string, then the g-name itself can be used as a sort key to order the names of all chemicals that belong to a given family. In choosing a generic name, whether it be a molecular formula or the g-name computed above, a balance must be made between short, simple names that cover many different compounds and more complex names that narrow the search to only a few.

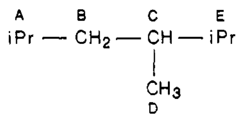
Table I. Generic Names of the Acyclic Isomers C_9H_{20} Showing the 28 Different g-Name Categories ?- query6. no_of_chemicals(35), no_of_gnames(28)

g-name order	s-name order[a]	g-name
1	c33	1:c/c-1:c/ch2-6:c/ch3
2	c20	1:c/ch-1:c/ch2-2:c/ch3-1:ch/ch2-1:ch/ch3-2:ch2/ch3
3	c32	1:c/ch-3:c/ch3-1:ch/ch-3:ch/ch3
4	c16	1:c/ch-3:c/ch3-1:ch/ch2-1:ch/ch3-1:ch2/ch2-1:ch2/ch3
5	c30	1:c/ch-3:c/ch3-2:ch/ch2-2:ch2/ch3
6	c18	1:c/ch2-3:c/ch3-1:ch/ch2-2:ch/ch3-1:ch2/ch2
7	c17	1:c/ch2-3:c/ch3-2:ch/ch2-1:ch/ch3-1:ch2/ch3
8	c5	1:c/ch2-3:c/ch3-3:ch2/ch2-1:ch2/ch3
9	c6	1:ch/ch-1:ch/ch2-3:ch/ch3-2:ch2/ch2-1:ch2/ch3
10	c12	1:c/ch-2:ch/ch2-2:ch/ch3-1:ch2/ch2-2:ch2/ch3
10	c24	1:ch/ch-2:ch/ch2-2:ch/ch3-1:ch2/ch2-2:ch2/ch3
11	c23	1:ch/ch-2:ch/ch2-5:ch/ch3
12	c26	1:ch/ch-3:ch/ch2-1:ch/ch3-3:ch2/ch3
13	c2	1:ch/ch2-2:ch/ch3-4:ch2/ch2-1:ch2/ch3
14	c31	2:c/ch-2:c/ch3-4:ch/ch3
15	c21	2:c/ch2-2:c/ch3-1:ch/ch2-2:ch/ch3-1:ch2/ch3
16	c19	2:c/ch2-2:c/ch3-1:ch/ch3-1:ch2/ch2-2:ch2/ch3
17	c10	2:c/ch2-2:c/ch3-2:ch2/ch2-2:ch2/ch3
17	c11	2:c/ch2-2:c/ch3-2:ch2/ch2-2:ch2/ch3
18	c34	2:c/ch2-6:c/ch3
19	c22	2:ch/ch-1:ch/ch2-4:ch/ch3-1:ch2/ch3
19	c28	2:ch/ch-1:ch/ch2-4:ch/ch3-1:ch2/ch3
20	c3	2:ch/ch2-1:ch/ch3-3:ch2/ch2-2:ch2/ch3
20	c4	2:ch/ch2-1:ch/ch3-3:ch2/ch2-2:ch2/ch3
21	c9	2:ch/ch2-4:ch/ch3-2:ch2/ch2
22	c29	3:c/ch2-1:c/ch3-1:ch/ch3-3:ch2/ch3
23	c25	3:c/ch2-1:c/ch3-1:ch2/ch2-3:ch2/ch3
24	c14	3:ch/ch2-2:ch2/ch2-3:ch2/ch3
24	c15	3:ch/ch2-2:ch2/ch2-3:ch2/ch3
25	c7	3:ch/ch2-3:ch/ch3-1:ch2/ch2-1:ch2/ch3
25	c8	3:ch/ch2-3:ch/ch3-1:ch2/ch2-1:ch2/ch3
26	c35	4:c/ch2-4:ch2/ch3
27	c13	4:ch/ch2-2:ch/ch3-2:ch2/ch3
27	c27	4:ch/ch2-2:ch/ch3-2:ch2/ch3
28	c1	6:ch2/ch2-2:ch2/ch3

[a] See Appendix 3.

NAMING OF SUBSTRUCTURES

We start by considering the s-name of 2,3,5-trimethylhexane generated from a structure written with isopropyl radicals (iPr).



The structure can be described as chem_with_macro (c1,'2,3,5-trimethylhexane', [iPr(A)/ch2(B),ch2(B)/ch(C), ch(C)/ch3(D),ch(C)/iPr(E)]). The bond iPr(X)/Y we write as a macro definition as follows: **macro**(iPr(X)/Y,[ch(X)/ch3(P),ch(X)/ch3(Q),ch(X)/Y]). The structure iPr(X) joined to node Y is by this definition the same as the node ch(X) joined to ch3(P) and ch(X) joined to ch3(Q) and ch(X) joined to Y, where P and Q are dummy variables. With such a definition it is an easy matter to define a predicate **expand/2** that takes a definition containing macros and expands them to form a definition containing only atomic terms. This can be defined as

```
expand(S1,S2) :-
    delete(A/B,S1,Temp1),
    (macro(A/B,L);macro(B/A,L)),
    append(Temp1,L,Temp2),
    expand(Temp2,S2).
expand(S,S).
```

We can test **expand/2** as follows (**query7**, Appendix 1):

```
?- chem_with_macro(c1,_,T1),expand(T1,S1).

T1 = [iPr(_345)/ch2(_347),ch2(_347)/ch(_356),
     ch(_356)/ch3(_365),ch(_356)/iPr(_374)]

S1 = [ch2(_347)/ch(_356),ch(_356)/ch3(_365),
     ch(_345)/ch3(_389),ch(_345)/ch3(_398),
     ch(_345)/ch2(_347),ch(_374)/ch3(_430),
     ch(_374)/ch3(_439),ch(_374)/ch(_356)]
```

The expanded form of the chemical structure can then be matched with a template describing the full structure. For example, we can match this structure with the data base of Appendix 3. This is demonstrated by the query (**query8**, Appendix 1)

```
?- chem_with_macro(c1,_,T1),expand(T1,S1),
   chem_db(_,Chem,S2),same(S1,S2).
       Chem = 2,3,5-trimethylhexane
```

which demonstrates the usage of both the expansion and matching mechanisms. Macros can be used to considerably shorten the description of an unknown structure and to make common substructure names that are appropriate for the descriptive job at hand. This property is one of the desired attributes that any new nomenclature system should have and echoes the argument of Silk[1] for "descriptiveness at an appropriate cognitive level". Classic chemical notation achieves this by providing an extensive vocabulary of predefined terms. In our approach the only fixed terms are the names of the atoms and the bonds that join them. What we do provide is a powerful macro facility, the use of which allows us to glue together collections of atoms to form interesting structures and to glue these structures together into yet more complex structures. This approach and its intimate connection to a programming language leads us to view a chemical structure as a formal language where the terminal symbols in the language are the atoms and bonds themselves, and the grammatical production rules define how to assemble complex parts by assembling simpler parts.

Note that macros can be arbitrary nested in a Prolog-based representation. The possibility of such an approach was put by Read,[13] who suggested iterating up from the atomic connectivity adding layers of common substructures at each cycle.

Searching for Common Substructures. Having defined the predicate **same/2** that tests if two structures are identical, it is an easy matter to test if one structure is a substructure of another structure:

```
sub_str(S,[ ]).
sub_str(S,S1) :-
    S1 = [A/B|Rest],
    (delete(A/B,S,Temp);
    delete(B/A,S,Temp)),
    sub_str(Temp,Rest).
```

The predicate **sub_str(S,Sub)** is true if Sub is a substructure of S. This definition is easy to understand if we compare it to the definition of **same/2**. The only difference is that instead of both lists being simultaneously reduced to the empty list it suffices to reduce the substructure list to zero. We could, for example, use this to test if the isopropyl radical is a substructure of 2,3,5-trimethylhexane (**query9**, Appendix 1):

```
?- example_chem(chem1,_,S1),
   sub_str(S1,[ch(X)/ch3(P),ch(X)/ch3(Q),ch(X)/Y]).
```

Prolog answers

```
yes
```

Another possible query is to find all chemicals in the data base that contain the isopropyl radical (**query10**, Appendix 1)

```
?- query10.
```

to which the answer is

```
2-methyloctane
2,3-dimethylheptane
2,4-dimethylheptane
2,5-dimethylheptane
2,6-dimethylheptane
3,4-dimethylheptane
3,5-dimethylheptane
2,2,5-trimethylhexane
3,3,5-trimethylhexane
2,3,4-trimethylhexane
2,3,5-trimethylhexane
3-ethyl-2-methylhexane
4-ethyl-2-methylhexane
3-ethyl-2,4-dimethylpentane
2,3,3,4-tetramethylpentane
2,2,3,4-tetramethylpentane
yes
```

which shows that 16 of the 35 chemicals of the C_9H_{20} homologues contain the isopropyl radical. The output of the query also serves as proof that 16 and only 16 of the chemicals of the C_9H_{20} homologues contain an isopropyl radical.

EFFICIENCY OF THE SEARCH

While we have only studied in detail the isomers of C_9H_{20}, we can make some estimates as to the efficiency of searching a database of a larger number of structures. Using a 40 kLip (1 Lip = 1 logical inference per second) Prolog interpreter running on a Sun 3/60 workstation, we measured the time taken for 10 000 insertions of the same s-name coded structure into the Prolog assert/retract data base to be 3.12 s. The timings for inserting the same structure 10 000 times should not be significantly different from the time to insert 10 000 different structures. Using this 10 000-structure data base, we found that an exhaustive match of an unknown structure against every structure in the data base took 37.9 s. This gives an average match time of about 40 ms. When matching smaller groups (e.g., **query3** in Appendix 1), we observed that the match took only 20 ms. The difference in time arises from virtual memory paging and garbage collection overheads in the larger problem.

When using g-name encoding, we expect to perform a very small number of matches to test if a given structure exists in the data base. If we imagine a compound program in which the g-name is used as a data-base key in a conventional data-base search and Prolog is used to search for the s-name equivalent of a g-name, we can envisage the data-base lookup and the Prolog matching each taking a few tens of milliseconds, leading to a total access time of under 100 ms for most structures that is independent of the size of the external data base.

DISCUSSION

Uniqueness of Naming Conventions. In this paper we have referred to the fact that our s-names and g-names are not unique. This may be proved trivially by showing the falsity of the assertion that s-names and g-names *are* unique by writing down two different encodings of the same structure. The implication of this observation is that it is not necessary to have a unique nomenclature system if an efficient isomorphism algorithm exists.

Extension to Stereochemical and Other Descriptors. Further notation may be introduced to allow more complex connections (including stereochemical information). We give two examples here and will expand on this approach in a later paper.

Throughout the paper we have not differentiated between different types of bonds in our atoms and macros. The scheme we have presented can easily be extended to handle different bond types. For example, a generalization of the entity representation aa(A)/bb(B) is **bond(Type,aa(A),bb(B))**, where Type is the bond type (single, double, etc.). Alternatively, different bond types may be introduced directly, e.g., ch2(A)//ch2(B) or ch(X)///ch(Y). Appropriate changes can be made to the predicates **same/2**, **expand/2**, etc. to reflect the generalized bond types.

One possible descriptor of rings in our system would be ring(A,B,C...), where A, B, C, etc. are the nodes in the ring. Thus, 1,2-dimethylbenzene could be represented as [aromatic(A,B,C,D,E,F),ch3(X)/A,ch3(Y)/B].

Translation of WLN. The expansion mechanism could also, for example, be used to expand Wiswesser line notation to s-name notation and thereby allow processing of WLN structures by the algorithms of this paper. In principle, any system that embodies connectivity with information on the connected entities can be translated to our system. We give the simple example of propanone, for which the WLN description 1V1 could become [1(X)/V(X,Y),V(X,Y)/1(Y)] with the definitions macro(1(A),[ch3(B)/A]) and macro(V(A,B),[c(X)/A,c(X)//o(Y),c(X)/B]). A full translation scheme for WLN will be given in another paper.

Comparison with Other Systems. A large number of systems (systematic nomenclature, WLN, etc.) are based on the following idea: (1) draw the structure and then (2) apply naming rules to yield a unique name. Two structures are considered the same if they have the same name. This follows from the assumed uniqueness of the final name. These methods break down if the naming rules do not yield a unique name, for example, if applying the rules in a different order (where this is permitted) gives a different name. Moreover, the rules themselves may be complex and their application error prone. A more direct approach is to write down a description of the structure and then apply a graph isomorphism test to see if two descriptions represent the same structure. This method, while being less liable to error, is combinatorially expensive. Our method is of this latter type, but the transformation to a normal form (the s-name) avoids the inefficiencies of unconstrained searching, and the g-name representation further reduces the search area to a point at which a large data base may be quickly searched.

The fact that the algorithms we have presented are short and easy to understand is of some interest. Many authors have presented algorithms in conventional imperative languages for performing similar operations; these algorithms are usually difficult to understand and even more difficult to prove correct.

The current trends toward declarative languages and the ease with which symbolic computations can be expressed in these languages lead the authors to believe that techniques such as those described in this paper could be used for the unambiguous description of chemical structures and that such descriptions could become the basis of communication between chemists, instead of the more descriptive, though difficult to understand, systematic nomenclature. Much current research in computer science is directed toward the production of extremely efficient Prolog systems capable of handling millions of facts. The work described in this paper was developed by using a 20-kLip interpreter running on an IBM PC/AT clone and a 40-kLip interpreter on a Sun 3/60. By the mid-1990s megalip machines should be available that will have the symbolic capacity to manage very large data bases of chemical structures. Prolog is not only well suited to the problem of describing chemical structures, it is also well suited for any tasks involved with the symbolic manipulation of structures—for example, deriving the systematic name of a structure from the s-name or vice versa. Such tasks, while possible, represent formidable problems in conventional languages.

APPENDIX 1: QUERIES

query1	test if two chemicals are the same
query2	locate unknown chemical in data base
query3	find all chemicals in data base that match unknown chemical
query4	demonstrate normalization of a structure
query5	compute the g-name of a structure
query6	make a table of descrete g-names in data base
query7	expand macros in chemical name
query8	expand macros in chemical name and locate chemical name in data base
query9	find if one chemical is a substructure of another
query10	find all isopropyl radicals in the C_9H_{20} data base

```
query1 :-
  example_chem(chem1,_,S1),
  example_chem(chem2,_,S2),
  same(S1,S2),
  write(S1),nl,
  write(S2).
query2 :-
  example_chem(chem1,_,S1),chem_db(Chem,_,S2),
  same(S1,S2),
  write(Chem),nl.
query3 :-
  findall(Chem,
  (example_chem(chem1,_,S1),chem_db(Chem,_,S2),
same(S1,S2)),L),
  write(L),nl.
query4 :-
  normalize_structure(
    [ch3(a)/ch(b),ch(b)/ch3(c),ch(b)/ch2(d),
    ch2(d)/ch(e),ch3(f)/ch(e),ch(e)/ch(g),
    ch(g)/ch3(h),ch(g)/ch3(i)],S),
  write(S).
query5 :-
  gname(
    [ch3(a)/ch(b),ch(b)/ch3(c),ch(b)/ch2(d),
    ch2(d)/ch(e),ch3(f)/ch(e),ch(e)/ch(g),
    ch(g)/ch3(h),ch(g)/ch3(i)],S),
  write(S).
```

```
query6 :-
  findall(X,chem_db(X,_,_),L),
  length(L,L1),
  write(no_of_chemicals(L1)),nl,
  findall(N,(chem_db(X,_,S1),gname(S1,N)),L3),
  sort(L3,L4),
  length(L4,L5),
  write(no_of_gnames(L5)),nl,!,
  findall((G-X),(chem_db(X,_,S2),gname(S2,G)),L6),
  keysort(L6,L7),
  query6_write_table(L7,xx,0).
query6_write_table([],_,_).
query6_write_table([Gname-Name|T],Gname,N) :-
  write(' '),write(N),write(' '),
  write(Name),write(' '),write(Gname),nl,
  query6_write_table(T,Gname,N).
query6_write_table([Gname-Name|T],_,N) :-
  N1 is N + 1,
  write(''),write(N1),write(''),
  write(Name),write(' '),write(Gname),nl,
  query6_write_table(T,Gname,N1).
query7 :-
  chem_with_macro(c1,_,T1),expand(T1,S1),
  write(T1),nl,
  write(S1).
query8 :-
  chem_with_macro(c1,_,T1),expand(T1,S1),
  chem_db(_,Chem,S2),same(S1,S2),
  write(Chem),nl.
query9 :-
  example_chem(chem1,_,S1),
  sub_str(S1,[ch(X)/ch3(P),ch(X)/ch3(Q),ch(X)/Y]).
query10 :-
  findall(N,chem_db(N,_,_),L),
  query10_filter(L).
query10_filter([]).
query10_filter([H|T]) :-
  chem_db(H,Name,S),
  sub_str(S,[ch(X)/ch3(P),ch(X)/ch3(Q),ch(X)/Y]),
  write(' '),write(Name),nl,
  query10_filter(T).
query10_filter([H|T]) :-
  query10_filter(T).
all_queries :-
  write(query1),nl,query1,nl,
  write(query2),nl,query2,nl,
  write(query3),nl,query3,nl,
  write(query4),nl,query4,nl,
  write(query5),nl,query5,nl,
  write(query6),nl,query6,nl,
  write(query7),nl,query7,nl,
  write(query8),nl,query8,nl,
  write(query9),nl,query9,nl,
  write(query10),nl,query10,nl.
```

APPENDIX 2: AUXILIARY DEFINITIONS

```
make_name(Norm,Name) :-
  maplist(key_name,Norm,T1),
  sort(T1,T2),
  count(T2,T1,T3),
  compress(T3,Name).
compress([H|T],Y) :-
  nname(H,Name1),
  compress1(T,Temp),
  append(Name1,Temp,T2),
  name(Y,T2).
compress([],[]).
compress1([H|T],Y) :-
  nname(H,Temp1),
  compress1(T,Temp2),
  append([45|Temp1],Temp2,Y). % 45 is asci magic for '-'
nname(Count-X/Y,Z) :-
  name(Count,T1),
  name(X,T2),
  name(Y,T3),
  append(T1,[58|T2],T4), % 58 is magic for ':'
  append(T4,[47|T3],Z). % 47 is asci magic number for '/'
count([],_,[]).
count([H|T],L,[N-H|T1]) :-
  count_occurrences(H,L,N),count(T,L,T1).
count_occurrences(H,L,N) :-
  findall(H,member(H,L),T1),
  length(T1,N).
key_name(X,Y) :-
  add_tag_key(X,Y-X).
```

Standard Prolog Algorithms[10,11]

```
delete(H,[H|T],T).
delete(H,[X|T],[X|T1]) :- delete(H,T,T1).
append([],L,L).
append([A|B],L,[A|C]) :- append (B,L,C).
member(X,[X|_]).
member(X,[_|Y]) :- member(X,Y).
maplist(_,[],[]).
maplist(Func,[H|T],[H1|T1]) :-
  Goal =.. [Func,H,H1],
  call(Goal),
  maplist(Func,T,T1).
findall(X,G,_) :-
  asserta(found(mark)),
  call(G),
  asserta(found(X)),
  fail.
findall(_,_,L) :- collect_found([],M),!,L=M.
collect_found(S,L) :- getnext(X),!,collect_found([X|S],L).
collect_found(L,L).
getnext(X) :- retract(found(X)),!, X \== mark.
```

APPENDIX 3

```
chem_db(c1,'nonane', [ch3(A)/ch2(B),
      ch2(B)/ch2(C),
      ch2(C)/ch2(D),
      ch2(D)/ch2(E),
      ch2(E)/ch2(F),
      ch2(F)/ch2(G),
      ch2(G)/ch2(H),
      ch2(H)/ch3(I)]).
chem_db(c2,'2-methyloctane',
      [ch3(A)/ch(B),
      ch(B)/ch3(C),
      ch(B)/ch2(D),
      ch2(D)/ch2(E),
      ch2(E)/ch2(F),
      ch2(F)/ch2(G),
      ch2(G)/ch2(H),
      ch2(H)/ch3(I)]).
chem_db(c3,'3-methyloctane',
      [ch3(A)/ch2(B),
      ch2(B)/ch(C),
      ch(C)/ch3(D),
      ch(C)/ch2(E),
      ch2(E)/ch2(F),
      ch2(F)/ch2(G),
      ch2(G)/ch2(H),
      ch2(H)/ch3(I)]).
chem_db(c4,'4-methyloctane',
      [ch3(A)/ch2(B),
      ch2(B)/ch2(C),
      ch2(C)/ch(D),
      ch(D)/ch3(E),
      ch(D)/ch2(F),
      ch2(F)/ch2(G),
      ch2(G)/ch2(H),
      ch2(H)/ch3(I)]).
chem_db(c5,'2,2-dimethylheptane',
      [ch3(A)/c(B),
      c(B)/ch3(C),
      c(B)/ch3(D),
      c(B)/ch2(E),
      ch2(E)/ch2(F),
      ch2(F)/ch2(G),
      ch2(G)/ch2(H),
      ch2(H)/ch3(I)]).
chem_db(c6,'2,3-dimethylheptane',
      [ch3(A)/ch(B),
      ch(B)/ch3(C),
      ch(B)/ch(D),
      ch(D)/ch3(E),
      ch(D)/ch2(F),
      ch2(F)/ch2(G),
      ch2(G)/ch2(H),
      ch2(H)/ch3(I)]).
chem_db(c7,'2,4-dimethylheptane',
      [ch3(A)/ch(B),
      ch(B)/ch3(C),
      ch(B)/ch2(D),
      ch2(D)/ch2(E),
      ch2(E)/ch2(F),
      ch2(F)/ch(G),
      ch(G)/ch3(H),
      ch(G)/ch3(I)]).
chem_db(c10,'3,3-dimethylheptane',
      [ch3(A)/ch2(B),
      ch2(B)/c(C),
      c(C)/ch3(D),
      c(C)/ch3(E),
      c(C)/ch2(F),
      ch2(F)/ch2(G),
      ch2(G)/ch2(H),
      ch2(H)/ch3(I)]).
chem_db(c11,'4,4-dimethylheptane',
      [ch3(A)/ch2(B),
      ch2(B)/ch2(C),
      ch2(C)/c(D),
      c(D)/ch3(E),
      c(D)/ch3(F),
      c(D)/ch2(G),
      ch2(G)/ch2(H),
      ch2(H)/ch3(I)]).
chem_db(c12,'3,4-dimethylheptane',
      [ch3(A)/ch2(B),
      ch2(B)/ch(C),
      ch(C)/ch3(D),
      ch(C)/ch(E),
      ch(E)/ch3(F),
      ch(E)/ch2(G),
      ch2(G)/ch2(H),
      ch2(H)/ch3(I)]).
chem_db(c13,'3,5-dimethylheptane',
      [ch3(A)/ch2(B),
      ch2(B)/ch(C),
      ch(C)/ch3(D),
      ch(C)/ch2(E),
      ch2(E)/ch(F),
      ch(F)/ch3(G),
      ch(F)/ch2(H),
      ch2(H)/ch3(I)]).
chem_db(c14,'3-ethylheptane',
      [ch3(A)/ch2(B),
      ch2(B)/ch(C),
      ch(C)/ch2(D),
      ch2(D)/ch3(E),
      ch(C)/ch2(F),
      ch2(F)/ch2(G),
      ch2(G)/ch2(H),
      ch2(H)/ch3(I)]).
chem_db(c15,'4-ethylheptane',
      [ch3(A)/ch2(B),
      ch2(B)/ch2(C),
      ch2(C)/ch(D),
      ch(D)/ch2(E),
      ch2(E)/ch3(F),
      ch(D)/ch2(G),
      ch2(G)/ch2(H),
      ch2(H)/ch3(I)]).
chem_db(c16,'2,2,3-trimethylhexane',
      [ch3(A)/c(B),
      c(B)/ch3(C),
      c(B)/ch3(D),
      c(B)/ch(E),
      ch(E)/ch3(F),
      ch(E)/ch2(G),
      ch2(G)/ch2(H),
      ch2(H)/ch3(I)]).
chem_db(c17,'2,2,4-trimethylhexane',
      [ch3nA)/c(B),
      c(B)/ch3(C),
      c(B)/ch3(D),
      c(B)/ch2(E),
      ch2(E)/ch(F),
      ch(F)/ch3(G),
      ch(F)/ch2(H),
      ch2(H)/ch3(I)]).
chem_db(c18,'2,2,5-trimethylhexane',
      [ch3(A)/c(B),
      c(B)/ch3(C),
      c(B)/ch3(D),
      c(B)/ch2(E),
      ch2(E)/ch2(F),
      ch2(F)/ch(G),
      ch(G)/ch3(H),
      ch(G)/ch3(I)]).
chem_db(c19,'2,3,3-trimethylhexane',
      [ch3(A)/ch(B),
      ch2(B)/ch3(C),
      ch2(B)/c(D),
      c(D)/ch3(E),
      c(D)/ch3(F),
      c(D)/ch2(G),
      ch2(G)/ch2(H),
      ch2(H)/ch3(I)]).
chem_db(c20,'3,3,4-trimethylhexane',
      [ch3(A)/ch2(B),
      ch2(B)/c(C),
      c(C)/ch3(D),
      c(C)/ch3(E),
        c(C)/ch(F),
        ch(F)/ch3(G),
        ch(F)/ch2(H),
        ch2(H)/ch3(I)]).
chem_db(c21,'3,3,5-trimethylhexane',
        [ch3(A)/ch2(B),
        ch2(B)/c(C),
        c(C)/ch3(D),
        c(C)/ch3(E),
        c(C)/ch2(F),
        ch2(F)/ch(G),
        ch(G)/ch3(H),
        ch(G)/ch3(I)]).
chem_db(c22,'2,3,4-trimethylhexane',
        [ch3(A)/ch(B),
        ch(B)/ch3(C),
        ch(B)/ch(D),
        ch(D)/ch3(E),
        ch(D)/ch(F),
        ch(F)/ch3(G),
        ch(F)/ch2(H),
        ch2(H)/ch3(I)]).
chem_db(c23,'2,3,5-trimethylhexane',
        [ch3(A)/ch(B),
        ch(B)/ch3(C),
        ch(B)/ch(D),
        ch(D)/ch3(E),
        ch(D)/ch2(F),
        ch2(F)/ch(G),
        ch(G)/ch3(H),
        ch(G)/ch3(I)]).
chem_db(c24,'3-ethyl-2-methylhexane',
        [ch3(A)/ch(B),
        ch(B)/ch3(C),
        ch(B)/ch(D),
        ch(D)/ch2(E),
        ch2(E)/ch3(F),
        ch(D)/ch2(G),
        ch2(G)/ch2(H),
        ch2(H)/ch3(I)]).
chem_db(c25,'3-ethyl-3-methylhexane',
        [ch3(A)/ch2(B),
        ch2(B)/c(C),
        c(C)/ch3(D),
        c(C)/ch2(E),
        ch2(E)/ch3(F),
        c(C)/ch2(G),
        ch2(G)/ch2(H),
        ch2(H)/ch3(I)]).
chem_db(c26,'3-ethyl-4-methylhexane',
        [ch3(A)/ch2(B),
        ch2(B)/ch(C),
        ch(C)/ch2(D),
        ch2(D)/ch3(E),
        ch(C)/ch(F),
        ch(F)/ch3(G),
        ch(F)/ch2(H),
        ch2(H)/ch3(I)]).
chem_db(c27,'4-ethyl-2-methylhexane',
        [ch3(A)/ch(B),
        ch(B)/ch3(C),
        ch(B)/ch2(D),
        ch2(D)/ch(E),
        ch(E)/ch2(F),
        ch2(F)/ch3(G),
        ch(E)/ch2(H),
        ch2(H)/ch3(I)]).
chem_db(c28,'3-ethyl-2,4-dimethylpentane',
        [ch3(A)/ch(B),
        ch(B)/ch3(C),
        ch(B)/ch(D),
        ch(D)/ch2(E),
        ch2(E)/ch3(F),
        ch(D)/ch(G),
        ch(G)/ch3(H),
        ch(G)/ch3(I)]).
chem_db(c29,'3-ethyl-2,3-dimethylpentane',
        [ch3(A)/ch(B),
        ch2(B)/ch3(C),
        ch2(B)/c(D),
        c(D)/ch3(E),
        c(D)/ch2(F),
        c(D)/ch2(G),
        ch2(G)/ch3(H),
        ch2(F)/ch3(I)]).
chem_db(c30,'3-ethyl-2,2-dimethylpentane',
        [ch3(A)/c(B),
        c(B)/ch3(C),
        c(B)/ch3(D),
        c(B)/ch(E),
        ch(E)/ch2(F),
        ch(E)/ch2(G),
        ch2(G)/ch3(H),
        ch2(F)/ch3(I)]).
chem_db(c31,'2,3,3,4-tetramethylpentane',
        [ch3(A)/ch(B),
        ch(B)/ch3(C),
        ch(B)/c(D),
        c(D)/ch3(E),
        c(D)/ch3(F),
        c(D)/ch(G),
        ch(G)/ch3(H),
        ch(G)/ch3(I)]).
chem_db(c32,'2,2,3,4-tetramethylpentane',
        [ch3(A)/c(B),
        c(B)/ch3(C),
        c(B)/ch3(D),
        c(B)/ch(E),
        ch(E)/ch3(F),
        ch(E)/ch(G),
        ch(G)/ch3(H),
        ch(G)/ch3(I)]).
chem_db(c33,'2,2,3,3-tetramethylpentane',
        [ch3(A)/c(B),
        c(B)/ch3(C),
        c(B)/ch3(D),
        c(B)/c(E),
        c(E)/ch3(F),
        c(E)/ch3(G),
        c(E)/ch2(H),
        c(E)/ch3(I)]).
chem_db(c34,'2,2,4,4-tetramethylpentane',
        [ch3(A)/c(B),
        c(B)/ch3(C),
        c(B)/ch3(D),
        c(B)/ch2(E),
        ch2(E)/c(F),
        c(F)/ch3(G),
        c(F)/ch3(H),
        c(F)/ch3(I)]).
chem_db(c35,'3,3-diethylpentane',
        [ch3(A)/ch2(B),
        ch2(B)/c(C),
        c(C)/ch2(D),
        ch2(D)/ch3(E),
        c(C)/ch2(F),
    ch2(F)/ch3(G),
    c(C)/ch2(H),
    ch2(H)/ch3(I)]).
```

REFERENCES AND NOTES

(1) Silk, J. V. Realistic versus Systematic Nomenclature. *J. Chem. Inf. Comput. Sci.* **1981**, *21*, 146-148.
(2) Wiswesser, N. J. *A Line Formula Chemical Notation*; Crowell: New York, 1954.
(3) Balaban, A. T. Applications of Graph Theory to Chemistry. *J. Chem. Inf. Comput. Sci.* **1985**, *25*, 334-343.
(4) Weininger, D. SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules. *J. Chem. Inf. Comput. Sci.* **1988**, *28*, 31-36.
(5) Wirth, K. Coding of Relational Descriptions of Molecular Structures. *J. Chem. Inf. Comput. Sci.* **1986**, *26*, 242-249.
(6) Abe, H.; Kudo, Y.; Yamasaki, T.; Tanaka, K.; Sasaki, M.; Sasaki, S.-I. A Convenient Notation System for Organic Structure on the Basis of Connectivity Stack. *J. Chem. Inf. Comput. Sci.* **1984**, *24*, 212-216.
(7) 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. *J. Chem. Inf. Comput. Sci.* **1981**, *21*, 151-161.
(8) Sussenguth, E. H. A Graph-Theoretical Algorithm for Matching Chemical Structures. *J. Chem. Doc.* **1965**, *5*, 36-43.
(9) Prolog terms are data structures used in logic programs. They are defined inductively as follows: A constant (i.e., an integer or atom) is a term; a variable is a term; a compound term is a term. A compound term is written by using the notation f(t1,t2,t3...tN), where f is the name of the compound term and each of the N arguments t1,t2...tN is a term. f is often referred to as the *functor* and N as the *arity* of the term. It is conventional to use the notation f/N to refer to some functor f of arity N. For more details see references 10 and 11.
(10) Bratko, I. *Prolog Programming for Artificial Intelligence*; Addison-Wesley: New York, 1986.
(11) Clocksin, W. F.; Mellish, C. S. *Programming in Prolog*; Springer-Verlag: Bonn, 1981.
(12) The programs in this paper have been tested and run on Quintus Prolog Version 1.5 running under BSD 4.2 UNIX, ALS Prolog Version 1.2 running under MS-DOS Version 3.1, and Sussex Poplog Prolog Version 10.
(13) Read, R. C. A New System for the Design of Chemical Compounds. 1. Theoretical Preliminaries and the Coding of Acyclic Compounds. *J. Chem. Inf. Comput. Sci.* **1983**, *23*, 135-149.

PAD Programming and Its Application in Chemistry

ZHENG XIANMIN

Chemistry Department, Hua Chiao University, Quanzhou, Fujian, China

Received July 26, 1988

Problem analysis diagram (PAD) is the most vital representation of software design at present. A synopsis of its principles, graphic representation, writing mode, and structure is given. The application of this software engineering method to structured programming in chemistry is also discussed and exemplified with problems common to the teaching and research of chemistry. Users who are not computer professionals can program efficiently in solving chemical problems with the computer.

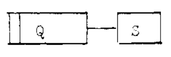
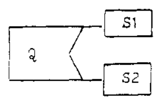
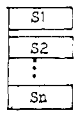
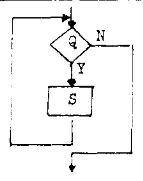
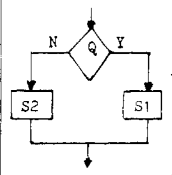
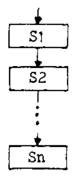
Software has developed into an epoch of software engineering in which the engineering method of expressing software design, manufacture, checking, and maintenance by graphics came into being. For applying software engineering methods to the field of chemistry so as to lessen the trouble that may be encountered in solving chemical problems, this paper presents the problem analysis diagram (PAD) method with special reference to its principle, graphic representation, writing mode, and structure. The application of PAD to structured programming in problems common to the teaching and research of chemistry is also discussed here.

SYNOPSIS OF PAD

PAD is the presentation of software design characterized as a two-dimensional tree structure.[1,2] By using PAD in structured programming, it is possible to represent program logic tersely so as to raise the efficiency of software design, manufacture, checking, and maintenance greatly and to make the program easy to read, remember, and understand.

(1) Basic Principle of PAD. As a two-dimensional tree structure representation for software design, PAD was generated on the basis of the improved Warnier diagram. Because of the use of the control structure of Pascal, it may be regarded as a two-dimensional expansion graph. Thus, PAD may also be regarded as an abbreviation of a Pascal diagram, known as the expansion graph of Pascal. Its essence lies in using the basic concept of top-down design and continual improvement so as to transfer the sketchy, vague idea for solving a problem into definite and thorough computerized processes.

Table I. Program Structure and Elementary Forms of PAD

type	circle type	choice type	sequence type
PAD Description			
FC Description			

Six symbols are used by PAD to describe treatment, repetition, selection, statement label, definition, and process, as shown in Figure 1.

Similar to other stipulated software methods, PAD provides procedures that should be followed by the software designer in designing a system or program.

(a) Petitioning of Sequence. Beginning with the design of a fuzzy concept of procedure and system, mark the sequence of each part of the existing process that is freely defined.

(b) Petitioning of Circulation. Mark the part that will be repeated and the condition of the beginning and end of repeating, that is, the condition of the beginning and end of the principal circulation process.

(c) Petitioning of Choice. Mark the condition of every process that will be implemented.