

List Operations on Chemical Graphs. 1. Basic List Structures and Operations

R. Gautzsch and P. Zinn*

Lehrstuhl für Analytische Chemie, Ruhr-Universität Bochum, Postfach 102 148, D-4630 Bochum 1, Germany

Received May 27, 1992

A software system is described to perform list operations on chemical graphs. In this approach the representation of chemical constitutions is based on adjacency lists. This representation technique is often much smaller than the adjacency matrix representation and reduces the complexity of algorithms on chemical graphs in many cases. An introduction to typical data structures and basic operations is given. Concepts and ideas are illustrated by examples corresponding to the described list data structures and list operations.

INTRODUCTION

To perform operations on chemical graphs, it is customary to represent the topology of the graph in a matrix, the adjacency matrix. In most cases of practical relevance, the adjacency matrix of a chemical graph is sparsely filled with topological information because the number of edges of chemical graphs is generally small. Therefore, it is often more efficient to store the topology of a chemical graph in a list, the adjacency list. The representation in an adjacency list is often much smaller than the adjacency matrix representation and never much larger.¹ The second even more important advantage is the fact that the type of graph representation may have dramatic influence on the complexity of algorithms on chemical graphs. Several graph algorithms are discussed¹ that work essentially faster using adjacency lists than related algorithms using adjacency matrices.

Because of the advantages of list representation, it is the intention of this paper to present some basic list operations on chemical graphs. These operations are of interest in a wide range of applications of graph theory in chemistry.² On investigating similarity models for quantitative structure-physical property relationships,^{3,4} a software system was developed to perform such list operations.

For the implementation, the list processing language LISP was used. An introduction to LISP programming language and related concepts is given in refs 5 and 6. The COMMON LISP standard we used is described in ref 7. LISP programming in chemistry is mainly described with respect to expert system applications.⁸ Although the list-structured connection table is the predominant method for the representation of chemical information⁹ and concepts of list-structured chemical information processing are discussed in the literature,^{10,11} a systematical description of list data structures and list operations in a list processing language does not exist.

In this first paper we will introduce the basic list structures and operations, illustrating the concepts with some limited examples. In further publications we will discuss special algorithms on chemical graphs and applications of the developed software system in chemistry.^{4,13}

BASIC LIST STRUCTURES

In order to handle chemical constitutions or structures in a computer, it is necessary to differentiate between special data types. In the case of list-oriented representation of chemical graphs, we used three types of information to characterize list structures as shown in Table I.

The data type of 'chemical information' includes all list structures containing information about chemical atoms,

Table I. Classification of List Structured Data for LISP Implementation of List Operations on Chemical Graphs

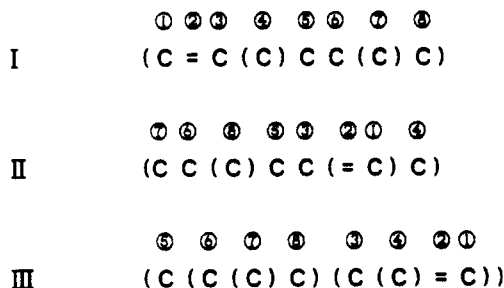
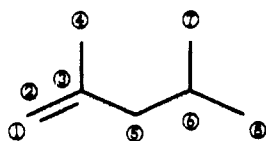
type of list structure	no.	list structure
chemical information	1	constitution
	2	expanded-constitution
numerical information	3	identification-list
	4	graph
	5	bond-list
	6	pure-bond-list
	7	tree
	8	path
mixed information	9	sphere
	10	atom-list
	11	atom-type-list
	12	bond-order-list
	13	bond-type-list

chemical bonds, charges, and electrons. 'Numerical information' contains mainly identification numbers of chemical information. The association between numerical and chemical information is given by lists of 'mixed information'. In the following, the most important list structures with respect to our research are discussed.

1. Constitution. The constitution is the interface to a system user. Because adjacency lists of chemical constitutions are lists of edges, it is difficult to handle them as input data structure for the representation of chemical constitution. Therefore, a line notation was chosen for the representation of chemical constitutions. There are several line notations discussed in literature.¹⁴⁻¹⁸ Our choice was a LISP implementation closely related to the SMILES¹⁶ notation, because the SMILES characterization of side chains corresponds with LISP sublist structures. This allows (unlike other line notations) an effective use of recursive programming techniques. In analogy to basic SMILES we implemented a basic LISP notation for the representation of the constitution of chemical compounds. This implementation consists of three rules.

- Constitutions are representations of hydrogen-depleted chemical constitutions containing the symbols for chemical atoms, bonds designated as "-" for single, "=" for double, "%" for triple, "+" for aromatic, "~" for nonaromatic delocalized double bonds; charges as positive "q" and negative "e"; free electrons as "**"; and electron pairs as "***". The "-" symbol for single bonds is optional.
- Branching is represented by parentheses directly following the node atom.
- The atom pair of a ring-closing bond is indicated by matching numbers following each of both atoms.

a) 2, 4-dimethyl-1-pentene



b) 1, 2-dimethyl-1-ethylcyclopropane

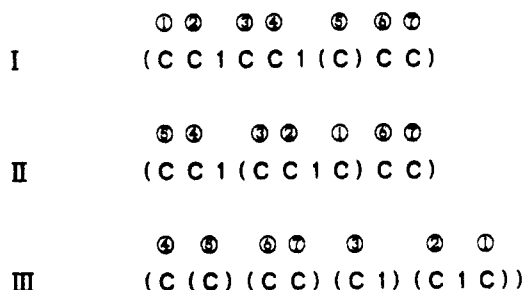
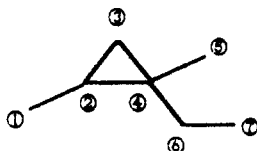


Figure 1. Examples of different representation of chemical constitutions with a SMILES like LISP notation. The choice of the starting atom and the way through the molecule are arbitrary.

The rules allow multiple representations of a constitution depending on the chosen start atom and the way through the molecule. To illustrate different possibilities of a constitution representation, the compounds in Figure 1, 2,4-dimethyl-1-pentene and 1,2-dimethyl-1-ethylcyclopropane, may serve as examples.

The following examples with respect to the other list structures of Table I correspond to the notations in Figure 1.

2. Expanded-Constitution. The expanded-constitution is a hydrogen-filled analogue to the constitution. It is mainly used to generate atom-type-lists and bond-type-lists which will be explained later on. Examples of expanded-constitutions are given in Figure 2.

3. Identification-List. The identification-list consists of identification numbers for each atom and bond of the constitution. It has the same sublist structure (branching structure) as the constitution. To avoid conflict, the numbers indicating ring closures are negative. The association between identification numbers and corresponding chemical information is made by the atom-list. Figure 3 gives examples of

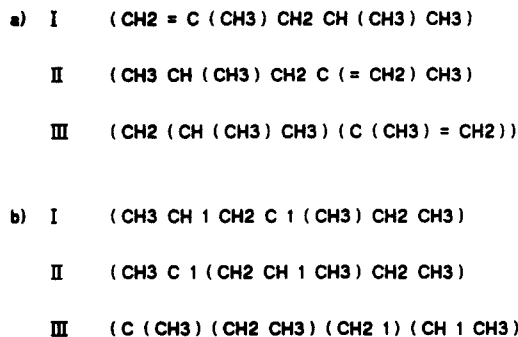


Figure 2. Expanded-constitutions corresponding to the examples in Figure 1.

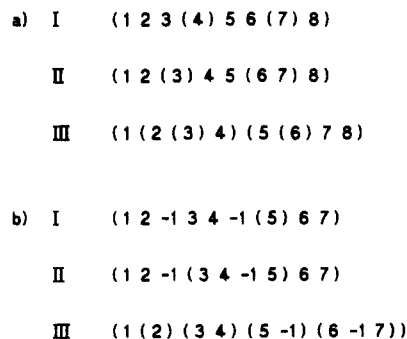


Figure 3. Identification-lists corresponding to the examples in Figure 1.

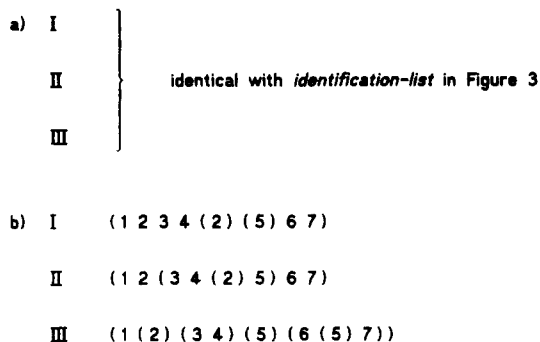


Figure 4. Graphs corresponding to the examples in Figure 1.

identification-lists and shows that the identification-list does not differentiate between chemical atoms and bonds.

4. Graph. The graph and the identification-list of acyclic compounds are identical. In the case of cyclic compounds, the first of a pair of ring bond indicating numbers is removed from the identification-list. The second one is replaced by the corresponding atom, set in parentheses. This causes multiple appearance of one identification number responsible for the ring-closing bond. The advantage of the graph structure is that acyclic and cyclic compounds can be handled in the same way. Ring-closing bonds are treated like bonds between node atoms and side chains. Figure 4 shows graph structures for the cyclic example compound.

5. Bond-List. The bond-list contains pairs of identification numbers representing all connections between LISP atoms in a constitution. The LISP atoms linked together in a bond-list may be chemical atoms or chemical atoms and chemical bonds. Keeping connections between chemical atoms and bonds in the bond-list simplifies the positioning of bond symbols during the rebuilding of constitutions from bond-lists. Examples of bond-lists are shown in Figure 5.

6. Pure-Bond-List. The pure-bond-list is the analogue to the bond-list containing only connections between chemical atoms. It is comparable with the adjacency list of graph theory.

a) I ((1 2) (2 3) (3 4) (3 5) (5 6) (6 7) (6 8))
 II ((1 2) (2 3) (2 4) (4 5) (6 7) (5 6) (5 8))
 III ((2 3) (2 4) (1 2) (5 6) (5 7) (7 8) (1 5))

b) I ((1 2) (2 3) (3 4) (4 2) (4 5) (4 6) (6 7))
 II ((1 2) (3 4) (4 2) (4 5) (2 3) (2 6) (6 7))
 III ((1 2) (3 4) (1 3) (1 5) (6 5) (6 7) (1 6))

Figure 5. Bond-lists corresponding to the examples in Figure 1.

a) I ((1 3) (3 4) (3 5) (5 6) (6 7) (6 8))
 II ((1 2) (2 3) (2 4) (4 5) (5 7) (5 8))
 III ((2 3) (2 4) (1 2) (5 6) (5 8) (1 5))

b) I }
 II } identical with bond-lists in Figure 5
 III }

Figure 6. Pure-bond-lists corresponding to the examples in Figure 1.

Bond-lists and pure-bond-lists of constitutions without bond symbols are identical, as shown in Figure 6.

7. **Tree.** The tree is a special notation of a graph. The tree starts with one root atom that branches in sublists which themselves may branch in further sublists again. For acyclic graphs, the tree is equivalent to graph theoretical definitions.¹⁷ All identification numbers of the graph are appearing exactly once. For cyclic graphs, the same tree-like structure is used. In these cases identification numbers corresponding to atoms in ring-closing bonds may appear multiple times. The structure of a tree depends on the chosen root element and the way through the graph. Examples for tree structures of acyclic and cyclic graphs are given in Figure 7.

8. **Path.** The path is a list of linked LISP atoms of a constitution represented by their identification numbers. It describes a way from a start to an end point of a constitution without crossings. Examples are shown in Figure 8.

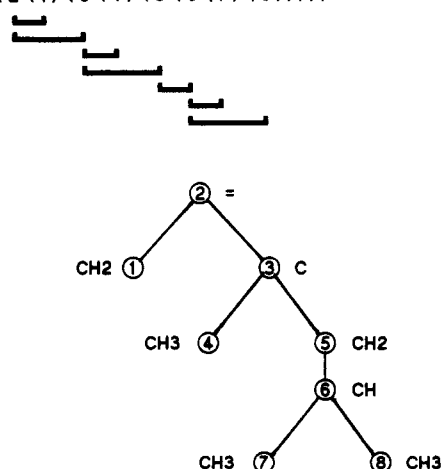
9. **Sphere.** The sphere is a list of chemical atoms represented by their identification numbers with a defined topological distance from a given atom. In the case of cyclic compounds, each atom is positioned only once in the sphere with the shortest distance. The sequence of the atoms in the sphere depends on the chosen constitution. Examples are shown in Figure 9.

10. **Atom-List.** The atom-list associates chemical information to identification numbers. It has the form of a LISP association list. The atom-list is an essential data structure because it is the connecting link between chemical and numerical information. Examples are shown in Figure 10.

11. **Atom-Type-List.** The atom-type-list is an expanded form of the atom-list. The association pairs describing chemical bonds are removed with respect to the atom-list. The symbols of chemical atoms are substituted by LISP atom-types of Table II. Examples are shown in Figure 11.

12. **Bond-Order-List.** The bond-order-list is an extended version of the pure-bond-list. Additionally, the symbols for the bond types are added to the pairs of bonded atoms

a) I (2 (1) (3 (4) (5 (6 (7) (8))))))



b) I (4 (3 (2 (1))) (2) (5) (6 (7)))

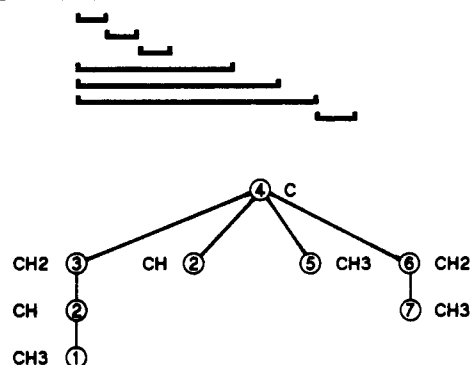


Figure 7. Tree-like LISP structures corresponding to the examples in Figure 1. Branching within the LISP structure is represented by indicators.

a) **Paths** from labeled atom ① to labeled atom ⑧ corresponding to the constitutions in Figure 1

I (1 2 3 5 6 8)
 II (7 6 5 4 2 3)
 III (8 7 5 1 2 4)

b) **Longest paths** from labeled atom ① to labeled atom ⑧ corresponding to the constitutions in Figure 1

I (1 2 3 4 6 7)
 II (5 4 3 2 6 7)
 III (7 6 5 1 3 4)

Figure 8. Paths corresponding to the examples in Figure 1.

represented by their identification numbers. Examples are given in Figure 12.

13. **Bond-Type-List.** The bond-type-list is an expanded form of the bond-order-list, where the bond symbols are substituted by the expanded LISP bond-types of Table III. The sequence of the bonds in the bond-type-list depends on the constitution, while the bond-types are invariant with respect to the constitution. Examples of bond-type-lists are shown in Figure 13.

a) Spheres of topological distance 2 for the labeled atom ④ corresponding to the constitutions in Figure 1

- I (1 4 7 8)
 II (1 3 7 8)
 III (3 4 6 8)

b) Spheres of topological distance 1 for the labeled atom ④ corresponding to the constitutions in Figure 1

- I (3 2 5 6)
 II (1 4 3 6)
 III (2 3 5 6)

Figure 9. Spheres corresponding to the examples in Figure 1.

- a) I ((1 C) (2 =) (3 C) (4 C) (5 C) (6 C) (7 C) (8 C))
 II ((1 C) (2 C) (3 C) (4 C) (5 C) (6 =) (7 C) (8 C))
 III ((1 C) (2 C) (3 C) (4 C) (5 C) (6 C) (7 =) (8 C))

- b) I }
 II } ((1 C) (2 C) (3 C) (4 C) (5 C) (6 C) (7 C))
 III }

Figure 10. Atom-lists corresponding to the examples in Figure 1.

Table II. LISP Notation for Typical Chemical Atom-Types of C-, O-, and N-Containing Compounds

skeleton atom	LISP atom-type	skeleton atom	LISP atom-type
H ₃ C—	CH3	HO—	OH
—CH ₂ —	CH2	—O—	O
>CH—	CH	O=	(O =)
>C<	C	H ₂ N—	NH2
H ₂ C=	(CH2 =)	—NH—	NH
—CH=	(CH =)	>N—	N
>C=	(C =)	HN=	(NH =)
HC≡	(CH %)	—NH=	(N =)
—C≡	(C %)	N≡	(N %)

BASIC LIST OPERATIONS

In order to handle and manipulate the described list data structures, procedures have been developed to perform basic list operations. These procedures allow the transformation from one list structure into another. The combination of the basic procedures leads to more complex operations that will be discussed in the following paper.¹²

The procedures are developed on a VAX-Station 3100 (VAX-Station is a trademark of Digital Equipment Co., Maynard, MA) under LUCID COMMON LISP (LUCID COMMON LISP is a trademark of Lucid Inc., Menlo Park, CA). A PC version to run under GOLDEN COMMON LISP (GOLDEN COMMON LISP is a trademark of Gold Hill Computers, Cambridge, MA) will soon be available.

Implementing list operations in LISP has led to procedures of high modularity. The result is a system of nested procedures,

- a) I ((1 (CH2 =)) (3 (C =)) (4 CH3) (5 CH2) (6 CH) (7 CH3) (8 CH3))
 II ((1 CH3) (2 CH) (3 CH3) (4 CH2) (5 (C =)) (7 (CH2 =)) (8 CH3))
 III ((1 CH2) (2 CH) (3 CH3) (4 CH3) (5 (C =)) (6 CH3) (8 (CH2 =)))

- b) I ((1 CH3) (2 CH) (3 CH2) (4 C) (5 CH3) (6 CH2) (7 CH3))
 II ((1 CH3) (2 C) (3 CH2) (4 CH) (5 CH3) (6 CH2) (7 CH3))
 III ((1 C) (2 CH3) (3 CH2) (4 CH3) (5 CH2) (6 CH) (7 CH3))

Figure 11. Atom-type-lists corresponding to the examples in Figure 1.

- a) I (((1 3) =) ((3 4) -) ((3 5) -) ((5 6) -) ((6 7) -) ((6 8) -))
 II (((1 2) -) ((2 3) -) ((2 4) -) ((4 5) -) ((5 7) =) ((5 8) -))
 III (((2 3) -) ((2 4) -) ((1 2) -) ((5 6) -) ((5 8) =) ((1 5) -))
 b) I (((1 2) -) ((2 3) -) ((3 4) -) ((4 2) -) ((4 5) -) ((4 6) -) ((6 7) -))
 II (((1 2) -) ((3 4) -) ((4 2) -) ((4 5) -) ((2 3) -) ((2 6) -) ((6 7) -))
 III (((1 2) -) ((3 4) -) ((1 3) -) ((1 5) -) ((6 5) -) ((6 7) -) ((1 6) -))

Figure 12. Bond-order-lists corresponding to the examples in Figure 1.

which is easy to extend and to implement for different hardware and software environments.

Because of the great number of procedures that have been developed, a unique nomenclature of the procedures seemed to be advantageous. Most names of procedures for the transformation of chemical information are constructed with a starting "generate-" and the appended name of the resulting list structure. The numerical information transforming procedures are named with a leading "make-". An overview about the developed basic list procedures is given in Table IV. In the following, the procedures are discussed without describing the algorithms but with examples. Showing the algorithms in detail would extend this paper too much. Nevertheless, we think that the ideas and concepts of list operations are to be understood looking at given examples. The examples are referring to the constitutions I of the compounds in Figure 1.

1. Make-identification-list. Make-identification-list transforms a graph into an identification-list. It takes a graph and tests for multiple appearance of identification numbers indicating cyclic compounds. In those cases the identification number that is set in parentheses is substituted by a ring-bond indicating negative number. The same indicator is appended to the corresponding identification number. Because a tree is a special kind of graph, make-identification-list also works

with trees. For acyclic compounds identification-list and graph are identical.

(make-identification-list graph)

Examples:

(a) input (1 2 3 (4) 5 6 (7) 8)
output (1 2 3 (4) 5 6 (7) 8)
(b) input (1 2 3 4 (2) (5) 6 7)
output (1 2 -1 3 4 -1 (5) 6 7)

2. Generate-identification-list. Generate-identification-list transforms a constitution into an identification-list. The procedure replaces the symbols of a constitution by increasing integer numbers. To avoid conflict, ring-bond indicating numbers of the constitution are negated. It also works with expanded-constitutions.

(generate-identification-list constitution)

Examples:

(a) input (C = C (C) C C (C) C)
output (1 2 3 (4) 5 6 (7) 8)
(b) input (C C 1 C C 1 (C) C C)
output (1 2 -1 3 4 -1 (5) 6 7)

3. Make-graph. Make-graph transforms an identification-list into a graph and is the inverse of example 1 above. The procedure tests for appearance of ring-bond indicators. The first of an indicator pair is removed from the identification-list, while the second one is replaced by the corresponding atom, set in parantheses. For acyclic compounds identification-list and graph are identical.

(make-graph identification-list)

Examples

(a) input (1 2 3 (4) 5 6 (7) 8)
output (1 2 3 (4) 5 6 (7) 8)
(b) input (1 2 -1 3 4 -1 (5) 6 7)
output (1 2 3 4 (2) (5) 6 7)

4. Generate-graph. Generate-graph transforms a constitution into a graph. It is a combination of the procedures generate-identification-list and make-graph. It also works with expanded-constitutions.

(generate-graph constitution)

Examples:

(a) input (C = C (C) C C (C) C)
output (1 2 3 (4) 5 6 (7) 8)
(b) input (C C 1 C C 1 (C) C C)
output (1 2 3 4 (2) (5) 6 7)

5. Make-bond-list. Make-bond-list derives the associative bond-list from a graph. To build the linked pairs of the bond-list, each atom of the graph is connected to its right neighbor atom. In cases of branching, the node atom is connected to the first atom of every following sublist and to the first atom following the sublists. Because of the definition of the graph structure, a special treatment of ring-closing bonds is not necessary.

(make-bond-list graph)

Examples:

(a) input (1 2 3 (4) 5 6 (7) 8)
output ((1 2) (2 3) (3 4) (3 5) (5 6) (6 7) (6 8))
(b) input (1 2 3 4 (2) (5) 6 7)
output ((1 2) (2 3) (3 4) (4 2) (4 5) (4 6) (6 7))

6. Generate-bond-list. Generate-bond-list derives the associative bond-list from a constitution. It is a combination of

generate-graph and make-bond-list. It also works with expanded-constitutions.

(generate-bond-list constitution)

Examples:

(a) input (C = C (C) C C (C) C)
output ((1 2) (2 3) (3 4) (3 5) (5 6) (6 7) (6 8))
(b) input (C C 1 C C 1 (C) C C)
output ((1 2) (2 3) (3 4) (4 2) (4 5) (4 6) (6 7))

7. Make-pure-bond-list. Make-pure-bond-list reduces a bond-list to a pure-bond list. It tests the linked pairs of a bond-list for pairs connecting chemical atoms with chemical bonds using the atom-list, which is also required. Those pairs are eliminated. The identification numbers of the chemical atoms of eliminated pairs are associated to new linked pairs.

(make-pure-bond-list bond-list atom-list)

Examples

(a) input ((1 2) (2 3) (3 4) (3 5) (5 6) (6 7) (6 8))
((1 C) (2 =) (3 C) (4 C) (5 C) (6 C) (7 C) (8 C))
output ((1 3) (3 4) (3 5) (5 6) (6 7) (6 8))
(b) input ((1 2) (2 3) (3 4) (4 2) (4 5) (4 6) (6 7))
((1 C) (2 C) (3 C) (4 C) (5 C) (6 C) (7 C))
output ((1 2) (2 3) (3 4) (4 2) (4 5) (4 6) (6 7))

8. Generate-pure-bond-list. Generate-pure-bond-list derives the pure-bond-list from a constitution. It is a combination of generate-atom-list and make-pure-bond-list. It also works with expanded-constitutions.

(generate-pure-bond-list constitution)

Examples:

(a) input (C = C (C) C C (C) C)
output ((1 3) (3 4) (3 5) (5 6) (6 7) (6 8))
(b) input (C C 1 C C 1 (C) C C)
output ((1 2) (2 3) (3 4) (4 2) (4 5) (4 6) (6 7))

9. Make-tree. Make-tree derives a tree from a bond-list. In a first step, the procedure starts with a root element and appends the branch of all neighbor atoms. In a recursive way all subbranches are appended, until all atoms are considered. In a second step, the resulting list structure is transformed in order to receive a parentheses structure, that is compatible to a graph structure. The procedure also works with a pure-bond-list resulting in tree structures without chemical bonds. In both cases it is necessary to specify the identification number of the trees root element.

(make-tree identification-number bond-list)

Examples:

(a) input 5
((1 2) (2 3) (3 4) (3 5) (5 6) (6 7) (6 8))
output (5 (3 (2 (1)) (4)) (6 (7) (8)))
(b) input 4
((1 2) (2 3) (3 4) (4 2) (4 5) (4 6) (6 7))
output (4 (3 (2 (1))) (2) (5) (6 (7)))

10. Generate-tree. Generate-tree derives a tree from a constitution. It is a combination of the procedures generate-bond-list and make-tree. The identification number of the root element is required. The resulting tree is dependent on the chosen input constitution.

(generate-tree identification-number constitution)

Examples:

(a) input 1
(C = C (C) C C (C) C)

Table III. LISP Notation for Typical Chemical Bond-Types of C-C, O-C, and N-C Bonds^a

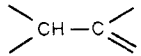
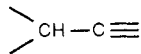
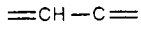
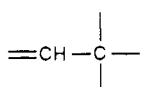
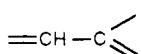
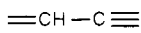
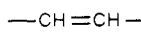
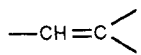
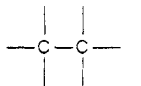
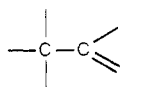
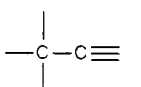
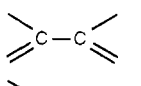
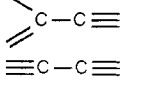

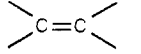
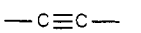
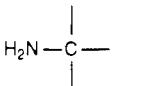
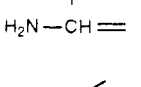
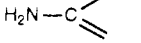
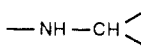
chemical bond	LISP bond-type	chemical bond	LISP bond-type
*H ₃ C-CH ₃	(CH3 - CH3)		(CH - (C =))
H ₃ C-CH ₂ -	(CH3 - CH2)		(CH - (C %))
H ₃ C-C<	(CH3 - CH)		((CH =) - (CH =))
H ₃ C-C-	(CH3 - C)		((CH =) - C)
H ₃ C-CH=	(CH3 - (CH =))		((CH =) - (C =))
H ₃ C-C=	(CH3 - (C =))		((CH =) - (C %))
H ₃ C-C≡	(CH3 - (C %))		(CH = CH)
-CH ₂ -CH ₂ -	(CH2 - CH2)		(CH = C)
-CH ₂ -CH<	(CH2 - CH)	*HC≡CH	(CH % CH)
-CH ₂ -C-	(CH2 - C)	HC≡C-	(CH % C)
-CH ₂ -CH=	(CH2 - (CH =))		(C - C)
-CH ₂ -C=	(CH2 - (C =))		(C - (C =))
-CH ₂ -C≡	(CH2 - (C %))		(C - (C %))
*H ₂ C=CH ₂	(CH2 = CH2)		((C =) - (C =))
H ₂ C=CH-	(CH2 = CH)		((C =) - (C %))
H ₂ C=C<	(CH2 = C)		((C %) - (C %))
<CH-CH<	(CH - CH)		(C = C)
<CH-C-	(CH - C)		(C % C)
<CH-CH=	(CH - (CH =))	*HO-CH ₃	(OH - CH3)
HO-CH ₂ -	(OH - CH2)		(NH2 - C)
HO-CH<	(OH - CH)		(NH2 - (CH =))
HO-C-	(OH - C)		(NH2 - (C =))
HO-CH=	(OH - (CH =))	H ₂ N-C≡	(NH2 - (C %))
HO-C=	(OH - (C =))	H ₂ N-CH ₃	(NH - CH3)
HO-C≡	(OH - (C %))	H ₂ N-CH ₂ -	(NH - CH2)
-O-CH ₃	(O - CH3)		(NH - CH)

Table III. (Continued)

chemical bond	LISP bond-type	chemical bond	LISP bond-type
$\text{—O—CH}_2\text{—}$	(O - CH2)	—NH—C—	(NH - C)
—O—CH<	(O - CH)	—NH—CH=	(NH - (CH =))
—O—C—	(O - C)	—NH—C=	(NH - (C =))
—O—CH=	(O - (CH =))	$\text{—NH—C}\equiv$	(NH - (C %))
—O—C=	(O - (C =))	$^*\text{HN=CH}_2$	(NH = CH2)
$\text{—O—C}\equiv$	(O - (C %))	HN=CH—	(NH = CH)
$^*\text{O=CH}_2$	(O = CH2)	HN=C<	(NH = C)
O=CH—	(O = CH)	>N—CH_3	(N - CH3)
O=C<	(O = C)	$\text{>N—CH}_2\text{—}$	(N - CH2)
$^*\text{H}_2\text{N—CH}_3$	(NH2 - CH3)	>N—CH<	(N - CH)
$\text{H}_2\text{N—CH}_2\text{—}$	(NH2 - CH2)	>N—C—	(N - C)
$\text{H}_2\text{N—CH<}$	(NH2 - CH)	>N—CH=	(N - (CH =))
>N—C=	(N - (C =))	=N—C<	((N =) - (C =))
$\text{>N—C}\equiv$	(N - (C %))	$\text{=N—C}\equiv$	((N =) - (C %))
=N—CH_3	((N =) - CH3)	—N=CH_2	(N = CH2)
$\text{=N—CH}_2\text{—}$	((N =) - CH2)	—N=CH—	(N = CH)
=N—CH<	((N =) - CH)	—N=C<	(N = C)
=N—C—	((N =) - C)	$^*\text{N}\equiv\text{CH}$	(N % CH)
=N—CH=	((N =) - (CH =))	$\text{N}\equiv\text{C—}$	(N % C)

^a An asterisk indicates relevant for one compound only.

output (1 (2 (3 (4) (5 (6 (7) (8))))))
 (b) input 7
 (C C 1 C C 1 (C) C C)
 output (7 (6 (4 (3 (2 (1)))) (2) (5))))

11. Generate-atom-list. Generate-atom-list derives the associative atom-list from a constitution to associate identification numbers with chemical information. The procedure takes a constitution and removes all ring-bond indicators and parentheses of sublists. The remaining constitution is used to derive an identification-list by the procedure generate-identification-list. The corresponding elements of both lists are associated in the atom-list.

(generate-atom-list constitution)

Examples:

(a) input (C = C (C) C C (C) C)
 output ((1 C) (2 =) (3 C) (4 C) (5 C) (6 C) (7 C) (8 C))
 (b) input (C C 1 C C 1 (C) C C)

output ((1 C) (2 C) (3 C) (4 C) (5 C) (6 C) (7 C))

12. Make-chemical-information. Make-chemical-information substitutes identification numbers in any list of numerical information by the associated chemical information. Therefore the corresponding atom-list is required. Negative numerical information is interpreted as ring-bond indicator. (make-chemical-information numerical-information atom-list)

Examples:

(a) input (1 2 3 (4) 5 6 (7) 8)
 ((1 C) (2 =) (3 C) (4 C) (5 C) (6 C) (7 C) (8 C))
 output (C = C (C) C C (C) C)
 (b) input (1 2 -1 3 4 -1 (5) 6 7)
 ((1 C) (2 C) (3 C) (4 C) (5 C) (6 C) (7 C))
 output (C C 1 C C 1 (C) C C)

13. Generate-expanded-constitution. Generate-expanded-constitution transforms a hydrogen-depleted constitution into

a) I ((1 3) (CH2 = C)) ((3 4) (CH3 - (C =)))
 ((3 5) (CH2 - (C =))) ((5 6) (CH2 - CH))
 ((6 7) (CH3 - CH)) ((6 8) (CH3 - CH))

II ((1 2) (CH3 - CH)) ((2 3) (CH3 - CH))
 ((2 4) (CH2 - CH)) ((4 5) (CH2 - (C =)))
 ((5 7) (CH2 = C)) ((5 8) (CH3 - (C =)))

III ((2 3) (CH3 - CH)) ((2 4) (CH3 - CH))
 ((1 2) (CH2 - CH)) ((5 6) (CH3 - (C =)))
 ((5 8) (CH2 = C)) ((1 5) (CH2 - (C =)))

b) I ((1 2) (CH3 - CH)) ((2 3) (CH2 - CH))
 ((3 4) (CH2 - C)) ((4 2) (CH - C))
 ((4 5) (CH3 - C)) ((4 6) (CH2 - C))
 ((6 7) (CH3 - CH2))

II ((1 2) (CH3 - C)) ((3 4) (CH2 - CH))
 ((4 2) (CH - C)) ((4 5) (CH3 - CH))
 ((2 3) (CH2 - C)) ((2 6) (CH2 - C))
 ((6 7) (CH3 - CH2))

III ((1 2) (CH3 - C)) ((3 4) (CH3 - CH2))
 ((1 3) (CH2 - C)) ((1 5) (CH2 - C))
 ((6 5) (CH2 - CH)) ((6 7) (CH3 - CH))
 ((1 6) (CH - C))

Figure 13. Bond-type-lists corresponding to the examples in Figure 1.

the hydrogen-filled constitution. The procedure counts the neighbored atoms and bonds of each chemical atom and calculates the numbers of hydrogens. The atoms are replaced by hydrogen-containing atoms.

(generate-expanded-constitution constitution)

Examples:

(a) input (C = C (C) C C (C) C)
 output (CH2 = C (CH3) CH2 CH (CH3) CH3)
 (b) input (C C 1 C C 1 (C) C C)
 output (CH3 CH 1 CH2 C 1 (CH3) CH2 CH3)

14. Reduce-expanded-constitution. Reduce-expanded-constitution transforms a hydrogen-filled constitution into a hydrogen-depleted constitution. The hydrogen-filled atoms of an expanded-constitution are replaced by the skeleton atoms.
 (reduce-expanded-constitution expanded-constitution)

Examples:

(a) }
 (b) } reverse to generate-expanded-constitution

15. Make-sphere. Make-sphere takes a bond-list and derives a list of all neighbors in a specified topological distance from a given node. Identification number of the node atom and the distance are required. In cyclic compounds, there may exist different distances between two atoms. In such cases the atom is positioned in the lowest sphere.

(make-sphere identification-number distance pure-bond-list)

Examples:

(a) input 5
 2
 ((1 3) (3 4) (3 5) (5 6) (6 7) (6 8))
 output (1 4 7 8)
 (b) input 4
 1
 ((1 2) (2 3) (3 4) (4 2) (4 5) (4 6) (6 7))
 output (3 2 5 6)

16. Generate-sphere. Generate-sphere derives a list of neighbors with the same topological distance from a specified node starting with a constitution. It is a combination of generate-pure-bond-list and make-sphere. It also works with expanded-constitution. Identification number of the node atom and the topological distance to the sphere are required.

(generate-sphere identification-number distance constitution)

Examples:

(a) input 3
 2
 (C = C (C) C C (C) C)
 output (6)
 (b) input 2
 2
 (C C 1 C C 1 (C) C C)
 output (5 6)

17. Make-atom-paths. Make-atom-paths derives all paths beginning with a given atom. The bond-list and the atom's identification number are required. It also works with pure-bond-lists. This is useful to derive paths without chemical bonds.

(make-atom-paths identification-number bond-list)

Examples:

(a) input 5
 ((1 2) (2 3) (3 4) (3 5) (5 6) (6 7) (6 8))
 output ((5) (5 3) (5 6) (5 3 2) (5 3 4) (5 3 2 1) (5 6 7) (5 6 8))
 (b) input 4
 ((1 2) (2 3) (3 4) (4 2) (4 5) (4 6) (6 7))
 output ((4) (4 3) (4 2) (4 5) (4 6) (4 3 2) (4 3 2 1) (4 2 1) (4 2 3) (4 6 7))

18. Generate-atom-paths. Generate-atom-paths takes a constitution and an identification number and derives all paths beginning with the specified identification-number. The optional input parameter 'pure' enables the computing of paths without chemical bonds. It is a combination of generate-bond-list or optional generate-pure-bond-list and make-atom-paths. It also works with expanded-constitutions.

(generate-atom-paths identification-number constitution & optional 'pure')

Examples:

(a) input 5
 (C = C (C) C C (C) C)
 'pure
 output ((5) (5 3) (5 6) (5 3 1) (5 3 4) (5 6 7) (5 6 8))
 (b) input 4
 (C C 1 C C 1 (C) C C)
 'pure
 output ((4) (4 3) (4 2) (4 5) (4 6) (4 3 2) (4 3 2 1) (4 2 1) (4 2 3) (4 6 7))

19. Make-all-paths. Make-all-paths takes a bond-list and derives all paths for each atom. The procedure uses multiple calls of make-atom-paths. It also works with pure-bond-lists. The resulting list includes all paths and the corresponding reverse paths.

Table IV. Overview about Some Basic List Operations^a

input	output							
	constitution	identification-list	graph	bond-list	tree	atom-paths	all-paths	sphere
constitution		generate-identification-list	generate-graph	generate-bond-list	generate-tree	generate-atom-paths	generate-all-paths	generate-sphere
identification-list	make-chemical-information		make-graph					
graph		make-identification-list		make-bond-list				
bond-list					make-tree	make-atom-paths		make-sphere
tree		make-identification-list		make-bond-list				
atom-paths							make-all-paths	

^a The procedures starting with "make-" are one-step operations manipulating numerical information, while the procedures starting with "generate-" are mostly combined operations manipulating chemical information.

(make-all-paths bond-list)

Examples:

(a) input ((1 3) (3 4) (3 5) (5 6) (6 7) (6 8))
 output ((1) (1 3) (1 3 4) (1 3 5) (1 3 5 6) (1 3 5 6 7) (1 3 5 6 8))
 (3) (3 1) (3 4) (3 5) (3 5 6) (3 5 6 7) (3 5 6 8)
 (4) (4 3) (4 3 1) (4 3 5) (4 3 5 6) (4 3 5 6 7) (4 3 5 6 8)
 (5) (5 3) (5 6) (5 3 1) (5 3 4) (5 6 7) (5 6 8)
 (6) (6 5) (6 7) (6 8) (6 5 3) (6 5 3 1) (6 5 3 4)
 (7) (7 6) (7 6 5) (7 6 8) (7 6 5 3) (7 6 5 3 1) (7 6 5 3 4)
 (8) (8 6) (8 6 5) (8 6 7) (8 6 5 3) (8 6 5 3 1) (8 6 5 3 4))
 (b) input ((1 2) (2 3) (3 4) (4 2) (4 5) (4 6) (6 7))
 output ((1) (1 2) (1 2 3) (1 2 4) (1 2 3 4) (1 2 3 4 5) (1 2 3 4 6) (1 2 3 4 6 7) (1 2 4 3) (1 2 4 5) (1 2 4 6) (1 2 4 6 7))
 (2) (2 1) (2 3) (2 4) (2 3 4) (2 3 4 5) (2 3 4 6) (2 3 4 6 7) (2 4 3) (2 4 5) (2 4 6) (2 4 6 7)
 (3) (3 2) (3 4) (3 2 1) (3 2 4) (3 2 4 5) (3 2 4 6) (3 2 4 6 7) (3 4 2) (3 4 5) (3 4 6) (3 4 2 1) (3 4 6 7)
 (4) (4 3) (4 2) (4 5) (4 6) (4 3 2) (4 3 2 1) (4 2 1) (4 2 3) (4 6 7)
 (5) (5 4) (5 4 3) (5 4 2) (5 4 6) (5 4 3 2) (5 4 3 2 1) (5 4 2 1) (5 4 2 3) (5 4 6 7)
 (6) (6 4) (6 7) (6 4 3) (6 4 2) (6 4 5) (6 4 3 2) (6 4 3 2 1) (6 4 2 1) (6 4 2 3)
 (7) (7 6) (7 6 4) (7 6 4 3) (7 6 4 2) (7 6 4 5) (7 6 4 3 2) (7 6 4 3 2 1) (7 6 4 2 1) (7 6 4 2 3))

20. Generate-all-paths. Generate-all-paths takes a constitution and derives all paths including all reverse paths. The procedure is a combination of generate-bond-list or optionally generate-pure-bond-list and make-all-paths. The optional input parameter 'pure' enables the computing of all paths without chemical bonds.

(generate-all-paths constitution & optional 'pure')

Examples:

(a) input (C = C (C) C C (C) C)
 'pure
 output identical with example 19.a

(b) input (C C 1 C C 1 (C) C C)

'pure

output identical with example 19.b

21. Make-paths-count. Make-paths-count calculates the number of paths with identical length starting with paths of 1 atom up to the longest paths. As input a list of paths is required.

(make-paths-count list-of-paths)

Examples:

(a) input output of example 19.a

output (7 12 14 8 8)

(b) input output of example 19.b

output (7 14 22 22 10 2)

22. Generate-atom-type-list. Generate-atom-type-list drives the atom-type-list from a constitution. It uses generate-expanded-constitution and builds atom-types explained in Table II.

(generate-atom-type-list constitution)

Examples:

(a) input (C = C (C) C C (C) C)

output ((1 (CH2 =)) (3 (C =)) (4 CH3) (5 CH2) (6 CH) (7 CH3) (8 CH3))

(b) input (C C 1 C C 1 (C) C C)

output ((1 CH3) (2 CH) (3 CH2) (4 C) (5 CH3) (6 CH2) (7 CH3))

23. Make-bond-order-list. Make-bond-order-list takes a bond-list and the corresponding atom-list and derives the bond-order-list. It uses internally the make-pure-bond-list-procedure.

(make-bond-order-list bond-list atom-list)

Examples:

(a) input ((1 2) (2 3) (3 4) (3 5) (5 6) (6 7) (6 8))
 ((1 C) (2 =) (3 C) (4 C) (5 C) (6 C) (7 C) (8 C))

output (((1 3) =) ((3 4) -) ((3 5) -) ((5 6) -) ((6 7) -) ((6 8) -))

(b) input ((1 2) (2 3) (3 4) (4 2) (4 5) (4 6) (6 7))
 ((1 C) (2 C) (3 C) (4 C) (5 C) (6 C) (7 C))

output (((1 2) -) ((2 3) -) ((3 4) -) ((4 2) -) ((4 5) -) ((4 6) -) ((6 7) -))

24. Generate-bond-order-list. Generate-bond-order-list derives the bond-order-list from a constitution. It is a combination of generate-bond-list, generate-atom-list, and make-bond-order-list. It also works with expanded-constitutions.

(generate-bond-order-list constitution)

Examples:

(a) input (C = C (C) C C (C) C)

output identical with example 23.a

(b) input (C C 1 C C 1 (C) C C)
 output identical with example 23.b

25. Generate-bond-type-list. Generate-bond-type-list derives the bond-type-list from a constitution. The procedure expands the bond-order-list generated by generate-bond-order-list and uses the types of the bonded atoms from the atom-type-list generated by generate-atom-type-list. Multiple appearance of a bond symbol is avoided. The sequence of a bond type is determined by a priority list. Resulting bond-types are shown in Table III.

(generate-bond-type-list constitution)

Examples:

(a) input (C = C (C) C C (C) C)
 output (((1 3) (CH2 = C)) ((3 4) (CH3 - (C =)))
 ((3 5) (CH2 - (C =))) ((5 6) (CH2 -
 CH)) ((6 7) (CH3 - CH)) ((6 8) (CH3
 - CH)))

(b) input (C C 1 C C 1 (C) C C)
 output (((1 2) (CH3 - CH)) ((2 3) (CH2 - CH))
 ((3 4) (CH2 - C)) ((4 2) (CH - C)) ((4
 5) (CH3 - C)) ((4 6) (CH2 - C)) ((6 7)
 (CH3 - CH2)))

CONCLUSIONS

List operations on chemical graphs are an approach to handle chemical information about the constitution of molecules in a computer. In this paper, the introduction of basic list structures and operations should demonstrate some fundamental concepts and ideas of this approach. A first application of the developed software system with respect to the investigation of the relationship between chemical constitution and gas chromatographic retention index data³ is already published. Further applications of list operations on chemical graphs are in preparation.⁴

Combining basic list operations leads to new procedures for a wide range of applications in chemistry dealing with molecular topology. One of these applications is the study of quantitative structure-activity relationship (QSAR). An important point of view in relation with QSAR studies is the development of topological indices. The combination of basic list operations offer new possibilities for the development of

new and the implementation of known topological indices. Examples of such combination will be given in the following paper.¹²

REFERENCES AND NOTES

- (1) Mehlhorn, K. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*; Springer-Verlag: Berlin, 1984.
- (2) Balaban, A. T. Applications of Graph Theory in Chemistry. *J. Chem. Inf. Comput. Sci.* **1985**, *25*, 334-343.
- (3) Gautzsch, R.; Zinn, P.; Haffer, Chr. M. Experiences with Symbolic Programming in the Automated Estimation of Retention Index Data. *Chem. Intell. Lab. Sys.* **1991**, *12*, 81-90.
- (4) Duvenbeck, Chr.; Zinn, P. List Operations on Chemical Graphs. 3. Development of Vertex and Edge Models for Fitting Retention Index Data. *J. Chem. Inf. Comput. Sci.*, in press.
- (5) Winston, P. H.; Horn, B. K. P. *LISP*; Addison-Wesley: Reading, MA, 1984.
- (6) Abelson, H.; Susman, G. J.; Susman, I. *Structure and Interpretation of Computer Programs*; MIT Press: Cambridge, MA, 1986.
- (7) Steele, G. L. *Common LISP Reference Manual*. Digital Press: Bedford, MA, 1984.
- (8) Lindsay, R. K.; Buchanan, B. G.; Feigenbaum, E. A.; Lederberg, J. *Applications of Artificial Intelligence for Organic Chemistry. The Dendral Project*; McGraw-Hill: New York, 1980.
- (9) Ash, J. E.; Warr, W. A.; Willett, P. *Chemical Structure Systems*; Ellis Horwood Series in Chemical Computation, Statistics and Information; Ellis Horwood: Chichester, 1991.
- (10) Gasteiger, J.; Hendriks, B. M. P.; Hoever, P.; Jochum, C.; Somberg, H. JCAMP-CS: A Standard Exchange Format for Chemical Structure Information in Computer-Readable Form. *Appl. Spectrosc.* **1991**, *45*, 4-11.
- (11) Wipke, W. T.; Dyott, T. M. Stereochemically Unique Naming Algorithm. *J. Am. Chem. Soc.* **1974**, *96*, 4834-4842.
- (12) Gautzsch, R.; Zinn, P. List Operations on Chemical Graphs. 2. Combining Basic List Operations. *J. Chem. Inf. Comput. Sci.*, following paper in this issue.
- (13) Gautzsch, R.; Zinn, P. List Operations on Chemical Graphs. 4. Unique Constitution Notation. In preparation.
- (14) Smith, E. G. *The Wiswesser Line Formula Chemical Notation*; McGraw-Hill: New York, 1968.
- (15) Hippe, Z.; Achmatowicz, O., Jr.; Hippe, R. Some Problems of Computer-Aided Discovery of Organic Synthesis. In *Data Processing in Chemistry*; Hippe, Z., Ed.; Elsevier Scientific Publishing Co.: Amsterdam, 1980; p 210.
- (16) 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.
- (17) Barnard, J. M.; Jochum, C. J.; Welford, S. M. ROSDAL: A Universal Structure Representation for PC-host Communication. In *Chemical Structure Information: Interfaces, Communication and Standards*; ACS Symposium Series No. 400; Warr, W. A., Ed.; American Chemical Society: Washington, DC, 1989, pp 76-81.
- (18) Ghoshal, N. Yet Another Linear Notation Scheme for Organic Compounds. 1. *J. Chem. Inf. Comput. Sci.* **1990**, *30*, 308-311.
- (19) Harary, F. *Graph Theory*; Addison-Wesley: Reading, MA, 1972.