match no. 27 pp. 175-189 1992

AEGIS, A STRUCTURE GENERATION PROGRAM IN PROLOG

H.J. Luinge

Analytische Molecuulspectrometrie, Universiteit Utrecht,
P.O. Box 80083, 3508 TB Utrecht, The Netherlands

(received: October 1992)

ABSTRACT

A description of the algorithm of the structure generator AEGIS is presented together with examples that illustrate the exhaustiveness and non-redundancy of the results.

INTRODUCTION

In our laboratory we have developed a knowledge-based system for the interpretation of infrared spectra (EXSPEC, [1]). This system contains a structure generator (AEGIS, [2]) capable of constructing all possible molecular structures starting from a molecular formula and any constraint on the presence and/or absence of structural fragments. An important topic in automated structure handling is the perception of rings. Therefore, AEGIS has been provided with a procedure that detects the smallest set of smallest rings (SSSR) and the largest ring in each structure generated. The system is written in Prolog using the list processing facilities of this programming language to expand fragments towards complete structures. This paper presents a summary of the algorithm and its implementation in Prolog. Furthermore, a number of examples serves to illustrate the exhaustiveness of the program as well as its non-redundancy.

THE ALGORITHM

Structure generation as performed by AEGIS is divided into the following steps:
(1) input of a molecular formula;
(2) calculation of the mass and the number of rings and/or unsaturated bonds;
(3) construction of the skeleton of the molecule consisting of atoms and single bonds;

(4) addition of bonds in order to form double and triple bonds and rings.

During structure generation two stacks are maintained onto which partial structures are stored. The first stack contains all intermediate structures. At each cycle of the generation process a partial structure is removed from the top of this stack. Addition of building blocks to this partial structure yields new structures that are stored onto the second stack. This procedure is repeated until no more structures are left on the first stack. Then the entire second stack is copied to the first stack, the second stack emptied and the process repeated. Hence, structure generation in AEGIS is performed in a breadth-first sense.

Partial and completed structures are compared for uniqueness with all other structures of the same size generated. As with increasing numbers of atoms the number of possible isomers grows roughly exponentially, the user is provided with the option to impose additional constraints on the generation process. These constraints are defined in terms of the presence or absence of structural fragments as suggested by the spectrum interpretation module of the EXSPEC system.

THE IMPLEMENTATION

The structure generation algorithm has been written in LPA MacProlog and implemented on an Apple Macintosh II computer. Details of the program are given below. For a more elaborate discussion on Prolog the reader is referred to the references [3,4,5,6].

Structures are represented as a list of non-hydrogen atoms and a list of interconnecting bonds. Butanol-2 for instance is represented as:

Atoms = [['C', a1, 0], ['C', a2, 0], ['C', a3, 0], ['C', a4, 0], ['O', a5, 0]]

Bonds = [[s, a1, a2], [s, a2, a3], [s, a3, a4], [s, a2, a5]].

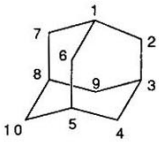
A label (a1 to a5) is assigned to each atom as well as a number stating the free valences left. Clearly, for completed structures this number equals zero. Bond types are represented by s, d, t or a for single, double, triple or aromatic bond respectively.

An important part of the structure generation process is the detection of sets of equivalent nodes in partial structures. Building blocks need only to be added to one node in each set in order to obtain expanded partial structures. A flowchart of the procedure is depicted in figure 1. The Prolog source code that is used for this purpose is as follows:

```
01  get_equivalent_atoms([Atoms, Bonds]) :-
02          remember(equivalent_sets, []),
03          remember(atoms_done, []),
04          on([Atomtype, Label_1, Free_valences], Atoms),
05          Free_valences > 0,
06          recall(atoms_done, Done),
07          not(on(Label_1, Done)),
08          remember(atoms_done, [Label_1 | Done]),
09          recall(equivalent_sets, Equivalent),
```

```
10        remember(equivalent_sets, [[Label_1] | Equivalent]),
11        on([Atomtype, Label_2, Free_valences], Atoms),
12        recall(atoms_done, Done_2),
13        not(on(Label_2, Done_2)),
14        one same_topology(Label_1, Label_2, Atoms, Bonds, Atoms, Bonds),
15        remember(atoms_done, [Label_2 | Done_2]),
16        recall(equivalent_sets, [Set | Rest]),
17        append(Set, [Label_2], New_Set),
18        remember(equivalent_sets, [New_Set | Rest]),
19        fail.
20  get_equivalent_atoms([Atoms, Bonds], Sets) :-
21        recall(equivalent_sets, Sets).
```

First, two lists are initialised to empty lists for storage of the resulting sets of equivalent nodes and the atoms that have been handled (line 02-03). Next, an atom is selected from the list of atoms (line 04). As building blocks can only be added to nodes with free valences left, the current atom is checked for this feature (line 05) and stored in the list of handled atoms (line 06-08). Also, it is stored in an, at the moment, empty set of equivalent nodes (line 09-10). A second atom is selected from the list of atoms (line 11). This atom must have the same number of free valences as the previous atom. Furthermore, it should not be on the list of handled atoms (line 12-13) and it should have the same topology (line 14). If this is all true, the atom is added to the list of handled atoms (line 15) and appended to the same set of equivalent atoms as the previous atom (line 16-18). The 'fail' statement (line 19) causes backtracking to take place (to line 11), ensuring that all atoms with the same number of free valences and topology are added to the current set as well. When a set of equivalent nodes is completed, backtracking (to line 4) results in finding a new atom from the atom list, constructing a new set of equivalent nodes (line 10) and repeating the entire process until all atoms have been handled. The second clause for the relation 'get_equivalent_atoms' retrieves the sets of equivalent nodes found (line 20-21). As an example the equivalent sets of nodes as found for adamantane are shown in figure 2. Generation of adamantanol isomers starting from this skeleton is achieved by adding a hydroxyl unit to a node from each of the sets thus yielding two isomers.



Equivalent nodes: [1, 3, 5, 8] and [2, 4, 6, 7, 10]

FIGURE 2. Adamantane skeleton and the corresponding sets of equivalent nodes.

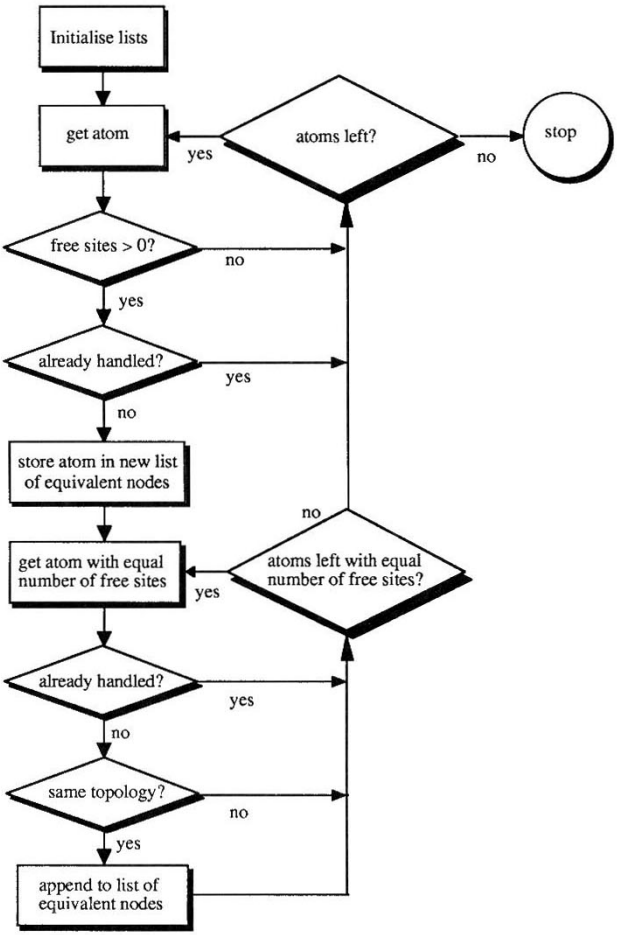


FIGURE 1. Flowchart of the procedure for detection of equivalent nodes.

The equivalence of two nodes in two (different or equal) structures is achieved by the procedure described below.

```
22 same_topology(Label_1, Label_2, Atoms_1, Bonds_1, Atoms_2, Bonds_2) :-
23       one on([Atomtype, Label_1, Free_valences], Atoms_1),
24       on([Atomtype, Label_2, Free_valences], Atoms_2),
25       one find_neighbours(Label_1, Bonds_1, Neighbours_1),
26       length(Neighbours_1, Length),
27       one find_neighbours(Label_2, Bonds_2, Neighbours_2),
28       length(Neighbours_2, Length),
29       check_neighbours(Label_1, Label_2, Neighbours_1, Neighbours_2,
                          Atoms_1, Bonds_1, Atoms_2, Bonds_2).
```

An atom with label 1 is selected from list 1 (line 23) and an identical atom (regarding type and free valences) is retrieved from list 2 (line 24). Then all neighbouring atoms of atom 1 are gathered in a list (line 25), which length is calculated (line 26). The neighbouring atoms of atom 2 are determined as well (line 27) and the length of the resulting list should equal the previously calculated length (line 28). Then each pair of neighbours is checked for having identical topology (line 29). Backtracking occurs when different topologies are found for a pair of atoms. It ensures that each single atom in structure 1 (line 23) is compared with all possible atoms in structure 2 (line 24) until an equivalent pair is met.

```
30 find_neighbours(Label, Bonds, [[Neighbour, Type] | Neighbours]) :-
31       delete_bond([Type, Label, Neighbour], Bonds, Left), !,
32       find_neighbours(Label, Left, Neighbours).
33 find_neighbours(Label, Bonds, []).
```

The procedure 'find_neighbours' collects all neighbouring atoms of a particular atom by retrieving (and deleting) all bonds from the atom to a neighbouring one (line 30-33). 'Check_neighbours' is a procedure that performs the comparison of topologies for each pair of neighbouring atoms (line 34-40) using the 'same_topology' procedure. Finally, 'delete_bond' removes a bond between two atoms irrespective of whether the bond is regarded as existing between atom 1 and atom 2 or between atom 2 and atom 1 (line 41-44).

```
34 check_neighbours(Label_1, Label_2, [[Neighbour_1, Type] | Left_1],
                    Neighbours_2, Atoms_1, Bonds_1, Atoms_2, Bonds_2) :-
35       delete_bond([Type, Label_1, Neighbour_1], Bonds_1, Bonds_left_1),
36       remove([Neighbour_2, Type], Neighbours_2, Left_2),
37       delete_bond([Type, Label_2, Neighbour_2], Bonds_2, Bonds_left_2),
38       same_topology(Neighbour_1, Neighbour_2,
                       Atoms_1, Bonds_left_1, Atoms_2, Bonds_left_2),
39       check_neighbours(Label_1, Label_2, Left_1, Left_2,
                       Atoms_1, Bonds_left_1, Atoms_2, Bonds_left_2).
40 check_neighbours(_, _, [], [], _, _, _, _).

41 delete_bond([Type, Label, Neighbour], Bonds, Bonds_left) :-
42          remove([Type, Label, Neighbour], Bonds, Bonds_left).
43 delete_bond([Type, Label, Neighbour], Bonds, Bonds_left) :-
44          remove([Type, Neighbour, Label], Bonds, Bonds_left).
```

Expansion of partial structures takes place by selecting an atom from each set of equivalent atoms, reducing its number of free valences by one, and adding a new atom and single bond to the lists of atoms and bonds respectively. The formula and calculated mass of the new partial structure are checked for agreement with the corresponding target values. The partial structure is compared with other partial structures of the same size for uniqueness.

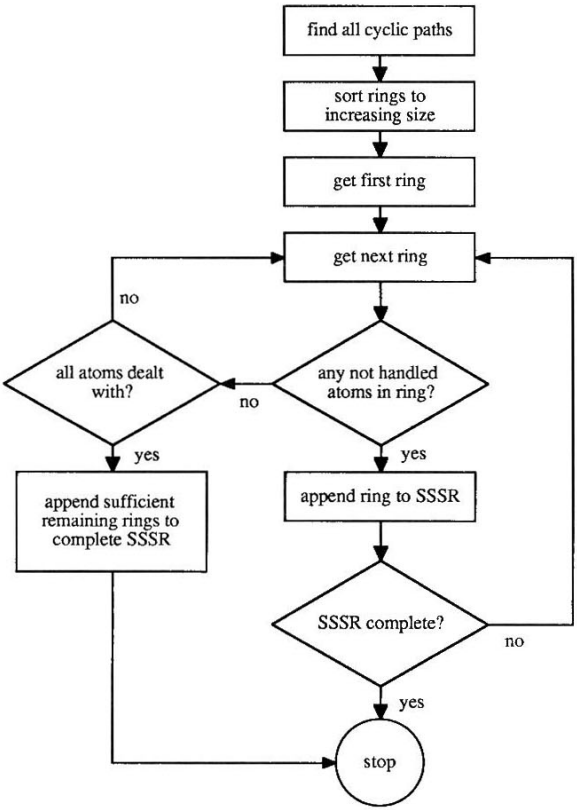


FIGURE 3. Flowchart of the procedure for the determination of the smallest set of smallest rings.

Once the acyclic part of the structure generation process has been completed bonds are added between all pairs of atoms with free bonding sites left. Again, after each addition the resulting structures are examined for uniqueness.

For all cyclic structures the total number of rings, the largest ring and the smallest set of smallest rings (SSSR) are determined. Figure 3 contains a flowchart showing the procedure for the determination of the SSSR. All atoms that can take part in a ring (*i.e.* atoms with at least two bonds to other atoms) are collected and all paths between two ring atoms (labelled L1 and L2) are determined using the following procedure:

```
45 find_path(L1, L2, Ring_atoms, Bonds, [L2 | Paths]) :-
46       L1 @=< L2,
47       delete_bond([_, L2, L3], Bonds, Remaining_bonds),
48       remove(L3, Ring_atoms, Remaining_atoms),
49       find_path(L1, L3, Remaining_atoms, Remaining_bonds, Paths).
50 find_path(L, L, Ring_atoms, Bonds, []).
```

During the search along a path bonds and ring atoms are removed (line 47-48) and atoms are added to the list named "Paths". This process stops in either of two cases: 1) when the path cannot be continued due to the lack of bonded ring atoms (in that case backtracking takes place to find alternatives) or 2) when the path reaches its starting point. In the latter case line 50 is fulfilled and the cyclic path is closed using the empty list symbol. By solving the "find_path" procedure in all possible ways all cyclic paths are found.

In order to improve the speed of the algorithm the labels of all possible ring atoms are sorted in increasing order and line 46 ensures that equal paths such as [1,2,3,4], [4,3,2,1] or [2,3,4,1] are calculated only once. (In fact all paths start at the atom with the lowest label). As an example all 7 cyclic paths found for bullvalene are depicted in fig. 4.

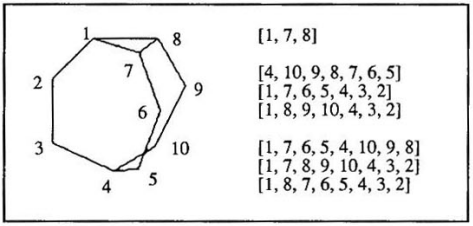


FIGURE 4. The structure of bullvalene and all cyclic paths found by AEGIS.

After having filtered out all duplicate paths and sorting the remaining paths with increasing pathlength, the smallest set of smallest rings is determined using the following procedure:

```
51 smallest_rings(_, Nr_of_rings, _, [], Ring_atoms).
52 smallest_rings(Paths, Nr_of_rings, Initial_ring, [Pathlength | Rest], Start) :-
53         Next is Start + 1,
54         append(T, [Path | Remaining_paths], Paths),
55         length(Path, Pathlength),
56         length(Initial_ring, LI),
57         one unification(Path, Initial_ring, SoFar),
58         length(SoFar, LS),
59         LI \= LS, !,
60         smallest_rings(Remaining_paths, Nr_of_rings, SoFar, Rest, Next).
61 smallest_rings(Paths, Nr_of_rings, Initial_ring, Rings, Start) :-
62         Left is Nr_of_rings - Start,
63         append(X, Y, Paths),
64         length(X, Left),
65         lengths(X, Rings).

66 unification(X, [H | T], Z) :-
67         on(H, X), !,
68         unification(X, T, Z).
69 unification(X, [H | T], Z) :-
70         append(X, [H], X1), !,
71         unification(X1, T, Z).
72 unification(X, [], X).

73 lengths([R | Rings], [L | LS]) :-
74         length(R, L), !,
75         lengths(Rings, LS).
76 lengths([], []).
```

The procedure starts with the first ring from the list of all rings (line 52) and appends the next ring to the set of smallest rings (line 54). However, this is the number of atoms in the unification of both rings (line 57-58) exceeds the length of the initial ring (line 56). In other words it is required that the newly added ring contains atoms that were not already present in the previous ring. If this is not the case, backtracking takes place and a new ring is selected (line 54). The size of each newly added ring is stored in a separate list (line 52 and 55). If all ring atoms in the molecule have been dealt with and the number of rings in the SSSR is still not equal to the number of rings in the molecule, the smallest remaining rings are added to the SSSR until the set is complete (line 61-65).

For bullvalene the smallest set of smallest rings is found to be [3,7,7], whereas the largest ring is of size 8.

Two more examples are depicted in fig. 5. For compound 5a 94 possible rings are found of which 7 are sufficient to describe the structure (SSSR = [4,4,4,4,4,4,6]). The largest ring is of size 12. Compound 5b contains 6 rings, SSSR equals [6,6,6] and the largest ring has size 14. The results are in agreement with those found by Baumer *et al* [7].

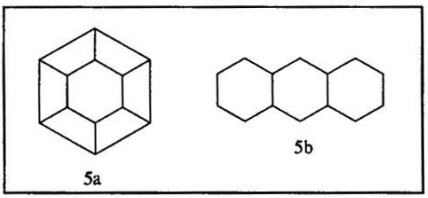


FIGURE 5. Two ring compounds.

RESULTS AND DISCUSSION

The time needed to generate structures strongly depends on the size of the molecules, the number of unsaturations, and the number of constraints. Table I, II and III show the number of isomers and the CPU times for a number of molecular formulas as measured on an Apple Macintosh II computer equipped with a Motorola 68020 processor and 68881 co-processor with a clock speed of 16 MHz.

TABLE I. Number of isomers (upper) and CPU times (lower) for structure generation of C_iH_j compounds using AEGIS.

C_iH_j	0	2	4	6	8	10	12	14	16	18	20	22
0	--	1 0	--	--	--	--	--	--	--	--	--	--
1	--	--	1 1	--	--	--	--	--	--	--	--	--
2	1 1	1 1	1 1	1 1	--	--	--	--	--	--	--	--
3	1 4	2 4	3 3	2 1	1 1	--	--	--	--	--	--	--
4	3 34	7 33	11 22	9 10	5 4	2 1	--	--	--	--	--	--
5	6 490	21 490	40 364	40 167	26 49	10 10	3 2	--	--	--	--	--
6				217 6140	159 1801	77 321	25 35	5 5	--	--	--	--
7							222 2426	56 150	9 12	--	--	--
8								654 22452	139 783	18 37	--	--
9										338 4687	35 136	--
10												75 592

TABLE II. Number of isomers (upper) and CPU times (lower) for structure generation of C_iH_jO compounds using AEGIS.

C_iH_jO	0	2	4	6	8	10	12	14	16	18	20	22
0	--	--	--	--	--	--	--	--	--	--	--	--
1	--	1 1	1 1	--	--	--	--	--	--	--	--	--
2	1 4	3 4	3 3	2 2	--	--	--	--	--	--	--	--
3	2 33	9 32	13 19	9 8	3 3	--	--	--	--	--	--	--
4			62 414	55 146	26 31	7 7	--	--	--	--	--	--
5					205 1435	74 155	14 21	--	--	--	--	--
6						747 --	211 1025	32 77	--	--	--	--
7									72 --	--	--	--

TABLE III. Number of isomers (upper) and CPU times (lower) for structure generation of C_iH_jN compounds using AEGIS.

C_iH_jN	1	3	5	7	9	11	13	15	17	19	21	23
0	--	1	--	--	--	--	--	--	--	--	--	--
1	1 1	1 1	1 1	--	--	--	--	--	--	--	--	--
2	2 7	5 5	4 3	2 2	--	--	--	--	--	--	--	--
3	7 81	19 63	21 30	12 10	4 3	--	--	--	--	--	--	--
4			116 864	85 240	35 40	8 8	--	--	--	--	--	--
5						100 212	17 25	--	--	--	--	--
6							284 1395	39 91	--	--	--	--
7									89 409	--	--	--
8										211 2153	--	--

In fig. 6 the relation between the amount of CPU time required for the generation of all isomers containing C and H and the number of carbon atoms is depicted graphically. It can be seen that the logarithm of the CPU time is approximately proportional to the square of the number of C-atoms. Furthermore, the CPU time depends on the number of

unsaturations present in the molecule. The slope of the curves increases approximately linearly with the number of unsaturations. The following model could be constructed:

$$\ln t = -0.61 + (0.048 \cdot u + 0.068) \cdot n^2,$$

with t = CPU time in seconds, u = number of unsaturations and n = number of C-atoms. Similar relations are obtained for isomers of oxygen and nitrogen containing compounds. Clearly, for large molecules with many unsaturations the approach chosen gives rise to very large calculation times. Hence, the possibility to restrain the generation process with structural fragments that are forbidden or obligatory is essential.

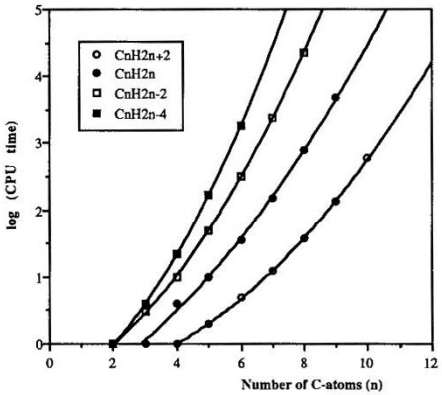
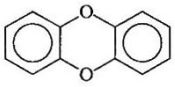


FIGURE 6. Logarithm of CPU time (in seconds) versus number of C-atoms.

With AEGIS it is possible to apply constraints to the generation process as required or prohibited structural fragments. As an example the results of the generation of isomers of dioxine are shown in table IV.

To evaluate the results obtained from AEGIS a comparison with several other structure generation programs has been made. One of the first structure generators was found in the Dendral system [8]. A large number of isomers was calculated for different molecular formulae with and without additional constraints. When comparing the results only one discrepancy between Dendral and AEGIS was found. For the number of conjugated acetals (fig.7) with formula $C_8H_{16}O_2$ Dendral finds 46 isomers, whereas AEGIS yields 52 structures as depicted in fig. 8. It should be noted that neither program distinguishes cis- and trans-isomers.

TABLE IV. Numbers of isomers of dioxine as generated by AEGIS with the presence of the dioxine skeleton as constraint.

Formula	Constraint	Isomers
$C_{12}H_7ClO_2$		2
$C_{12}H_6Cl_2O_2$		10
$C_{12}H_5Cl_3O_2$		14
$C_{12}H_4Cl_4O_2$		22
$C_{12}H_3Cl_5O_2$		14
$C_{12}H_2Cl_6O_2$		10
$C_{12}H_1Cl_7O_2$		2
$C_{12}Cl_8O_2$		1
Total		75

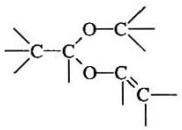


FIGURE 7. Structural fragment imposed as constraint during generation of conjugated acetals with formula $C_8H_{16}O_2$.

Also compared with CHEMICS [9,10] different numbers of isomers were found in several cases. AEGIS outperformed CHEMICS especially when halogen containing compounds were dealt with as is illustrated in table V.

TABLE V. Number of isomers generated for halogen containing compounds. Between brackets the number of acyclic isomers is shown.

Molecular formula	AEGIS	CHEMICS
C_2H_2BrCl	2 (2)	2
C_3H_4BrCl	10 (8)	8
C_4H_6BrCl	39 (27)	30
C_5H_8BrCl	140 (87)	108

When additional constraints are applied again significant differences are obtained. AEGIS yields 12 isomers (fig. 9) with formula C_5H_7NO containing both a vinyl (-$CH{=}CH_2$) and a carbonyl (>C=O) group and lacking a primary or secondary amine

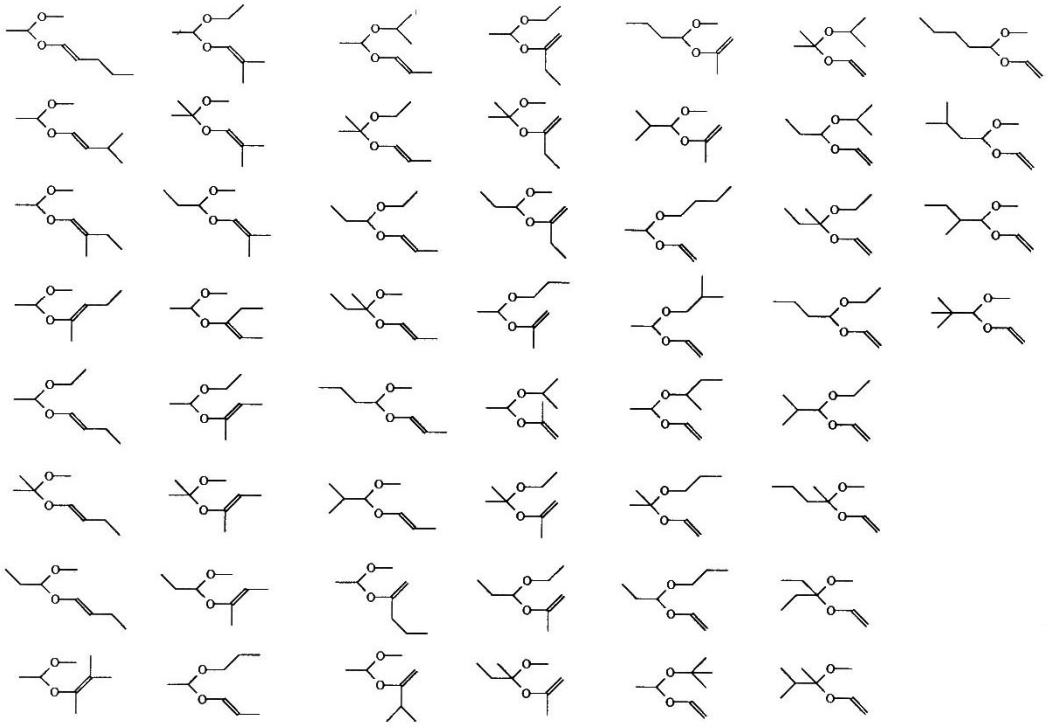


FIGURE 8. Conjugated acetals with formula $C_8H_{16}O_2$ as generated by AEGIS.

group (>NH, -NH_2), whereas CHEMICS finds 13 isomers. For C_6H_9NO and as additional constraints the presence of both a vinyl and a carbonyl group with AEGIS 155 isomers are found in contrast to CHEMICS, which generates only 131 isomers.

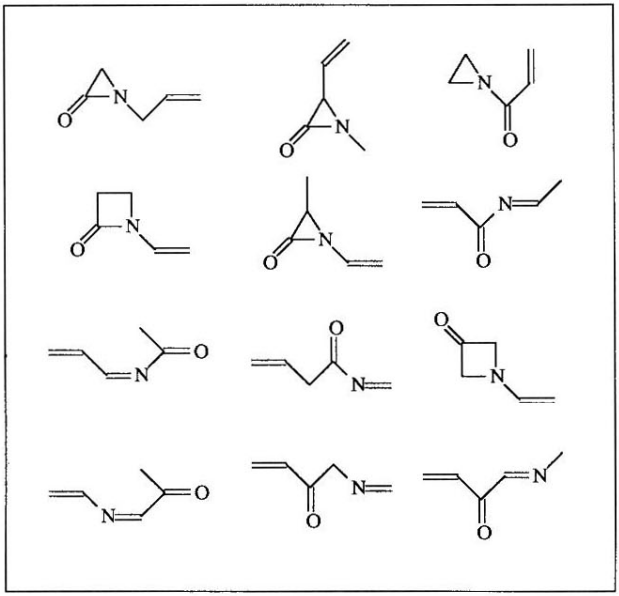


FIGURE 9. Isomers found by AEGIS for C_5H_7NO with >C=O and -CH=CH_2 present and without >NH or -NH_2.

CONCLUSION

The structure generator AEGIS has proven to be exhaustive and capable of generating unique structures in contrast to several structure generators described in the literature. It uses the simplicity of PROLOG as a programming language to generate and analyse molecular graphs. Its main drawback is the fact that it requires a relatively large amount of CPU time, which can be accounted for partly by the fact that PROLOG is not a very fast language.

REFERENCES

1 H.J. Luinge, EXSPEC, a Knowledge-Based System for Structure Analysis of Organic Molecules from Combined Spectral Data, thesis, Utrecht, 1989.

2 H.J. Luinge and J.H. van der Maas, Chemom. Intell. Lab. Syst., 8 (1990) 157.

3 W.F. Clocksin and C.S. Mellish, Programming in Prolog, Springer-Verlag, Berlin, 1984.

4 K.L. Clark, F.G. McCabe, N. Johns and C. Spenser, LPA MacPROLOG™ Reference Manual, Logic Programming Associates Ltd., London, 1987.

5 G.J. Kleywegt, H.J. Luinge and B.J. Schuman, Chemometrics Intell. Lab. Syst., 4 (1988) 273.

6 G.J. Kleywegt, H.J. Luinge and B.J. Schuman, Chemometrics Intell. Lab. Syst., 5 (1989) 117.

7 L. Baumer, G. Sala and G. Sello, Computers Chem., 15 (1991) 293.

8 R.K. Lindsay, B.G. Buchanan, E.A. Feigenbaum and J. Lederberg, Applications of Artificial Intelligence for Organic Chemistry: The Dendral Project, McGraw-Hill, New York, 1980.

9 H. Abe, T. Okuyama, I. Fujiwara and S.I. Sasaki, J. Chem. Inf. Comput. Sci., 24 (1984) 220.

10 K. Funatsu, N. Miyabayashi and S. Sasaki, J. Chem. Inf. Comput. Sci., 28 (1988) 18.