```
working space:
rule Rl                                <not defined yet>
 := ?                                  <query for options>
one of the following:
CLEAR  FETCH  NAME  GRAPH BREAK PEAKGROUP DRAW SHOW ADD QUIT
??
 := GRAPH
entering editstruc.                    <use EDITSTRUC for defining
                                        subgraphs>
(NEW STRUCTURE)
>RING 5
>BRANCH 5 1 1 1 1 1 2 1
>NDRAW

Rl
    3   9
   / \ /
  4   2
  |   |                                 <define and draw subgraph>
  5---1
 /   / \
6   8   7
>DONE
(Rl DEFINED)
                                         <specify cleavage by naming
                                          bonds cleaved between
                                          numbered atoms>
 := BREAK (1 2)(5 4)
 := PEAKGROUP                            <specify what peaks are
new list of peak groups.                  produced by the cleavage>
 ::= ?                                   <query for options>
one of the following:
DELETE  CLEAR  FETCH  NAME  TRANSFERS  SIGNIFICANCE  INTENSITY
SHOW  ADD  QUIT  ??
 ::= NAME Pl                             <name peakgroup>
 ::= T H -1                              <include loss of hydrogen>
 ::= SHOW
working space:
peak group Pl
(TRANSFERS -1)
 ::= INTENSITY 80                        <assign relative intensity
 ::= SIGNIFICANCE 90                      and plausibility>
 ::= ADD
Pl included in PEAKGROUPS
next:
 ::= NAME P2                             <name P2>
 ::= T H -1 H2O -1                       <accompanied by loss of
 ::= SH                                   water and hydrogen>
working space:
peak group P2
(TRANSFERS :-H-H2O)
 ::= I 50                                <assign relative intensity
 ::= SI 60                                and plausibility>
 ::= SH
```

```
working space:
peak group P2
(TRANSFERS :-H-H2O)
(INTENSITY 50)
(SIGNIFICANCE 60)
 ::= AD
P2 included in PEAKGROUPS
next:
 ::= Q
 := SHOW
working space:
rule R1                                  <summarize rule R1>
show subgraph drawing? Y/N. : N
show connection table? Y/N. : N
(BREAK (1 . 2) (5 . 4))
peak group P1
(TRANSFERS -1)
(INTENSITY 80)
(SIGNIFICANCE 90)
peak group P2
(TRANSFERS :-H-H2O)
(INTENSITY 50)
(SIGNIFICANCE 60)
 := ADD
R1 included in RULES                     <add R1 to list of rules>
next:
 := N R2                                 <define R2>
 := G                                    <etc....>
```

Applying these rules to a set of candidate structures produces a predicted spectrum. This predicted spectrum is different from the one created by MSPRED or MSRANK, and more closely resembles an observed spectrum. The peaks in this predicted spectrum have different intensity values, and the density of the spectrum i.e. the number of peaks per mass range is smaller. This minimizes the number of incorrect predictions and makes the entire predicted spectrum more closely related to an observed spectrum of an unknown compound. We are also working on a program to plot predicted spectra. This will be useful for visual comparison of plotted observed spectrum against a predicted one.

The next step is to explore ways to compare a predicted and an observed spectrum. We are experimenting with different ranking functions (see section 4) and developing a program which will allow the user to define in a simple mathematical equation his individual ranking function. The problem of ranking candidate structures based on spectrum comparison is closely related to the problem of library search. In our case, however, we do not have authentic spectra of our structural candidates in most instances. The density of a predicted spectrum for a candidate is quite low because we do not attempt to predict the complete spectrum. Rather, we predict major fragmentations. This fact must be taken into account in designing a function to rank candidates based on comparison of their predicted spectra to that of the unknown.

2.6 Molecular Ion Determination

The original MOLION program [10] was based upon the postulate: "There exists at least one SECONDARY LOSS in a spectrum that will match a PRIMARY LOSS from the molecular ion irrespective of whether the molecular ion is present in the spectrum."

Given this postulate, then one method of generating candidate masses for a molecular ion (M+) is to identify all possible secondary losses apparent in a spectrum, and then to add each of these losses to the masses of those ions observed in the high mass region of the spectrum. This, together with some refinements, was the basis of the original MOLION program. The most important of these refinements were:

(i) "PLANNING" i.e. the filtering of the set of apparent secondary losses against a table of "bad losses" (containing chemically implausible values like 9 amu and 23 amu), thus reducing the number of initial candidate M+s.

(ii) "TESTING". An acceptable candidate M+ had to be greater than, or equal in mass to the highest mass ion observed, and none of its immediate losses to observed ions could be in the list of "bad losses".

There are, however, a number of problems with the algorithm used in MOLION. The most crucial problem is that the algorithm requires good spectra! Impurities such as column bleed or co-eluting minor components can result in ions that would constitute bad losses —— causing the rejection of distinct and well supported molecular ions recorded in the spectrum. Further, the program did not allow the user to modify the "bad loss" set, nor to have access to the molecular ion scoring mechanisms. These scoring mechanisms incorporated a considerable measure of class dependency. Thus when testing a candidate M+, the program could modify the score associated with the M+ by the intensity combination formula: e.g. a mass difference of 101amu between the candidate M+ and an observed ion resulted in a 1.8 times increase in that M+'s score whereas a mass difference of 2 or 16 reduced the score by 85 per cent and a difference of 44, 56, 60 or 72 reduced the score by 25 per cent.

[10] R.G.Dromey, B.G.Buchanan, D.H.Smith, J.Lederberg and C.Djerassi. "Applications of Artificial Intelligence to Chemical Inference. XIV. A General Method for Predicting Molecular Ions in Mass Spectra." Journal of Organic Chemistry40770 (1975).

In devising the new version of the molecular ion program, an attempt has been made to recognize and overcome some of these problems. The resulting program has the following new characteristics:

1) The user has complete control of all aspects of the candidate evaluation procedures; these evaluation procedures being defined in terms of conventional chemical concepts.

2) The scoring algorithm allows for the separate accumulation of evidence supporting and disconfirming a particular candidate mass for M+. Simple yes/no tests, like MOLION's "bad losses", are not used. In this way, the program is made a little more tolerant of impurities in the spectrum, etc.

The basic algorithm remains: candidate M+s are generated and then ranked according to whether they are of the expected parity, show chemically favorable or unfavorable losses etc.

The candidate generation procedure allows for candidates in the mass range I-115 to I+115 where I is the highest mass observed ion. Any ion in this region is a candidate, as is any mass that can be obtained by adding an apparent neutral loss to the mass of an observed ion. The apparent neutral losses are simply the mass differences between all pairs of ions in the spectrum. No chemical information is used at this stage. The set of apparent neutral losses is not filtered against any "bad loss" set; consequently, the set will contain many losses that cannot correspond to any conceivable chemical fragmentation (e.g. 9amu). This initial candidate generation procedure does contribute to the scores of candidate M+s; a candidate is scored proportional to (i) the number of ways it can be generated, and (ii) the importance accorded to the losses involved.

Once all M+ candidates have been generated, each is scored according to rules describing molecular ion properties. These rules include tests on parity, testing that the candidate M+ ion is the most intense ion in the region M-2 to M+4, determining whether there are ions observed at higher masses etc.

Finally, rules defining chemically important fragmentation processes are applied. These rules can be general in form, e.g. just specifying that losses such as 7, 9, 22 amu etc are chemically implausible and so, if ions are observed that would involve such losses from a candidate M+ then that candidate has to be down rated. In addition, it is possible for the chemist to specify class specific rules. Thus, if the chemist knows that loss of methyl groups and water is characteristic of the compounds he is analyzing, then he may specify that 15 and 18 are good losses which, if observed, should increase the score of a candidate M+.

The scoring scheme uses the Confidence Factor model of the MYCIN program[11] This Confidence Factor (CF) model is intended for situations where a proper Bayesian statistical approach is inappropriate (because the requisite a priori and conditional probabilities are not known). The CF model simply requires that the chemist be able to express, in semiquantitative form, ideas like:

"the occurrence of an ion with the mass of a particular candidate M+ strongly increases my belief in that candidate being correct."

"the observation of an ion that could be due to loss of H2O from a candidate M+ slightly increases my belief in that candidate."

"the occurrence of ions at masses higher than a candidate M+ greatly increases my disbelief in that candidate."

Separate measures are kept of the total evidence supporting and opposing each hypothesis. These are the measures of belief in a hypothesis given some evidence (BF(h,e)), and the measure of disbelief in the hypothesis (DBF(h,e)). The overall confidence in a hypothesis is given by the difference of these measures:

$$CF(h,e) = BF(h,e) - DBF(h,e)$$

The range of values allowed for the measures BF and DBF is from 0 (corresponding to no evidence) to 1 (proof); thus, the CF value for a hypothesis can range from -1 (total disbelief) to 1 (total acceptance).

As additional evidence is found that supports some hypothesis, the measure of belief in that hypothesis increases asymptotically to 1:

$$BF(h,e1\&e2) = BF(h,e1) + (1 - BF(h,e1))*BF(h,e2)$$

(it is implicit that e1 and e2 are independent). A similar formula defines how the accumulation of negative evidence causes the disbelief to increase asymptotically.

In many cases, there may be uncertainty about the premise of some rule for scoring candidate M+s. The MYCIN model allows rules to be used with reduced strength when there is uncertainty about the premise. Thus, given a rule of the form

"if a candidate M+ can be generated by adding a known neutral loss to some observed ion's mass then my confidence in that M+ is increased by 0.1"

[11] E.H.Shortliffe and B.G.Buchanan. "A Model of Inexact Reasoning in Medicine." Mathematical Biosciences,23,351 (1975).

then if one were only 0.6 confident that some apparent neutral loss was significant in a spectrum this rule would be used to support a candidate M+ with a strength of belief of 0.06 (product of confidence in premise of rule and strength of inference if one were certain that premise was true).

The chemist's control over MOLION's M+ evaluation scheme is expressed through the following parameters:

1) Parameters that influence the preference for even/odd mass candidate M+s.

2) Parameters expressing the importance accorded to a candidate M+ actually being observed in the recorded spectrum.

3) Significance of ions above a candidate M+.

4) Chemically significant neutral losses.

5) Identifying secondary losses from the recorded spectrum.

6) Use of H+ transfer to add extra losses to the set of apparent secondary losses.

The following example shows the set of rules used to process some example sterol spectra:

IF the ratio of even mass ions / odd mass ions exceeds 0.50 THEN: there is evidence for Nitrogen presence and belief in all odd mass M+s is increased by 0.10 ELSE: by default disbelief in odd mass M+ is increased by 0.20

IF the ratio of accumulated intensities of even and odd mass ions exceeds 0.50 THEN: there is evidence for Nitrogen presence and belief in all odd mass M+s is increased by 0.10 ELSE: by default, disbelief in odd mass M+ is increased by 0.20

IF a candidate even mass molecular ion is in the recorded spectrum, THEN: belief in that candidate is increased by 0.50 ELSE: disbelief in that candidate is increased by 0.30

IF a candidate odd mass molecular ion is in the recorded spectrum THEN: belief in that candidate is increased by 0.25 ELSE: disbelief in that candidate is increased by 0.25

IF ions occur above M+4 for some candidate molwt M THEN: disbelief in that candidate is increased by 0.20

IF the candidate molecular ion M is not the most intense in the range M-2 to M+4 THEN: disbelief in that candidate is increased by 0.40

The following neutral losses are held to be chemically significant and confirm belief in a candidate M+

15	0.50
18	0.50
31	0.20
33	0.25

The following neutral losses should not occur. If such a loss is implied by a candidate M+ then disbelief in that candidate is increased by the specified amount

3	0.05	26	0.40
4	0.10	28	0.10
5	0.40	30	0.10
6	0.40	34	0.40
7	0.40	35	0.40
8	0.40	36	0.40
9	0.40	37	0.40
10	0.40	38	0.40
11	0.40	39	0.40
12	0.40	40	0.40
13	0.40	44	0.10
14	0.40	45	0.10
16	0.10	46	0.10
19	0.10	47	0.10
20	0.40	48	0.10
21	0.40	49	0.10
22	0.40	50	0.10
23	0.40	51	0.10
24	0.40	52	0.10
25	0.40	53	0.10
		60	0.10
		61	0.10
		62	0.10

Each time that it is observed in the spectrum, belief in the reality of an apparent 2ndry loss is increased by 0.05

Belief in a candidate M+ is increase by 0.10 each time that it is generated by adding the mass of a known 2ndry loss to an observed fragment ion.

If loss of I amu has been identified as a 2ndry loss, but loss of I+1 amu was not apparent Then loss of (I+1) amu can, with confidence 0.50, be added to the set of losses used for M+ generation.

If loss of I amu has been identified as a 2ndry loss, but loss of I-1 amu was not apparent Then loss of (I-1) amu can, with confidence 0.50, be added to the set of losses used for M+ generation.

An example of the determination of the molecular ion of 23-

nor-gorgost-5-en-3beta-ol is given below. The voluminous output is included for illustrative purposes to see the operation of various parts of the program. In normal operation all but the conclusion of the program is omitted.

```
   23-NOR-GORGOST-5-EN-3BETA-OL

       [This spectrum includes some column bleed, e.g. m/e  405 at
M+ - 7amu.]

 SPECTRUM AFTER TRIMMING AND CLUSTERING --

    M/E   INT    M/E   INT    M/E   INT    M/E   INT
     44    25     55  1375     67   433     69   908
     79   407     81   774     83  1165     91   367
     93   459     95   660    105   531    107   601
    109   360    119   384    121   341    123   207
    131   240    133   554    135   251    143   215
    145   465    147   356    158   231    159   480
    161   351    171   179    173   242    175   191
    185   152    187   180    189   166    197   116
    199   172    203    87    211   217    213   418
    217   231    228   181    229   495    231   273
    239   135    243   210    246   137    253   247
    255   395    267   147    271   667    272   672
    273   280    281  1258    296   306    299   341
    300   226    314  1877    328   345    351    92
    352    40    369    64    370    34    379   114
    380    38    393   311    394   166    397   104
    405   393    412   503

 CANDIDATE MOLWTS & SCORES
          .
          .
          .
  407  0.49  0.00
  408  0.51  0.00
  409  0.45  0.00
  410  0.40  0.00
  411  0.43  0.00
  412  0.43  0.00          [correct M+ not particularly highly
  413  0.39  0.00          ranked.]
  414  0.33  0.00
  415  0.37  0.00
          .
          .
          .

CANDIDATE M+s AND BF/DBF RATINGS AFTER MOLTST [Now start  to make
use of chemical information, like do we expect even or odd parity
M+, should the molecular ion be there, candidates at masses below
highest mass observed are less likely etc]
          .
          .
          .
  408   0.51   0.58
  409   0.45   0.71
  410   0.40   0.58
  411   0.43   0.71
  412   0.71   0.00          [Correct M+ is doing quite well, it
  413   0.39   0.71          occurs, it is most intense in its
  414   0.33   0.58          group etc.]
  415   0.37   0.52
          .
          .
          .
 CANDIDATE M+s AND BF/DBF RATINGS AFTER CHMFLT
          .
          .
          .
  408   0.75   0.95
  409   0.72   0.95
  410   0.52   0.93
  411   0.77   0.91
  412   0.95   0.56          [although belief in 412 increased by some
  413   0.54   0.96          good losses, we also have bad losses
  414   0.33   0.98          (e.g. loss of 7 to 405) that increase
  415   0.68   0.97          disbelief.]
          .
          .
          .
 TOP RANKED MOLECULAR WEIGHTS AND THEIR "CF" SCORES
  412     0.38
```

2.6.1 Summary of Results From New MOLION Program

Results from four experiments with the MOLION program are summarized in Table III below.

Experiment	Compounds processed.
A	sterols.
B	acid methyl esters.
C	acid TMS esters.
D	amino acid TAB esters.

The table distinguishes cases where the molecular ion was correctly identified by being the highest ranked candidate (usually with a CF score considerably greater than any alternative candidate) and cases where the correct molecular ion was merely listed in the top ten candidates (in these cases all the top ten candidates having approximately equal CF scores).

	COMPOUND CLASS			
	A	B	C	D
M+ present & identified :	44	8	14	4
M+ present & listed :	7	5	1	-
M+ present and not identified:	1 (a)	4 (b)	2 (c)	-
M+ absent but identified :	-	6	18	7
M+ absent but listed :	-	1	-	8
M+ absent and not identified:	-	-	2 (c)	4 (b,d)

Notes on errors:

(a) errors due to ions recorded at masses considerably above M+
(b) errors due to impurities.
(c) errors due to simple parity tests failing to detect presence of nitrogen.
(d) errors due to mass differences of more than 115amu between the highest mass ion recorded and the true molecular weight.

Table III. MOLION Results with Four Classes of Compounds.

2.6.2 Proposed Additional Work on the MOLION Program

The MOLION program is currently being implemented on the PDP11/45 based computerized GC/MS systems in the Departments of Chemistry and Genetics. The program will be evaluated through tests on the analysis of urine samples and other body fluids. Further developments of the system will be made in accord with the results of these tests.

2.7 Congen Improvements

During the past year many improvements have been made in the version of CONGEN available for outside use. These improvements allow the user more flexibility and range in the use of existing commands. Further, some new commands have been created which increase the power and utility of CONGEN. The program has become easier to use and more robust. Finally in almost every subsection of the program the user can inspect the computation as it proceeds. This means that fewer long, wasteful computations will be performed.

2.7.1 Error Detection in Substructure Definition

We now differentiate between two types of substructures called patterns and superatoms. This simplifies the chemist's interaction with the program in that it makes explicit the two different ways in which substructural information is used in the current version of CONGEN. Further, this distinction helped us write very complete error detecting routines in a relatively small number of lines of code.

When the chemist indicates that he(she) is finished defining a substructure by typing "done" in editstruc, we check the substructure for errors. If we find errors we ask the chemist whether or not he chooses to fix them. However, if the chemist tries to enter this substructure on the composition list or on the constraints list without fixing it we indicate that the errors have not been fixed. Further we ask him if he would like to fix them. If he says yes, we put him inside Editstruc. In Edit-struc there is a new command called ERRORS which will print out all of the errors made in definition.

If the chemist does not choose to fix the errors, we warn very clearly that the results will be unpredictable or erroneous. We allow the chemist to go ahead on the philosophy that he may have a perfectly good reason for doing what seems to us to be nonsense.

Some examples of the types of errors detected are:

a) x names and polynames in superatoms;

b) undefined atom names in patterns or superatoms;

c) free valences on patterns;

d) no free valences on superatoms;

e) an atom with too many attached bonds or a conflict between the hydrogen range specified and the number of free valences specified;

f) the lack of exactly one tag in a proton constraint; and

g) problems connected with use of multiple link nodes.

With link nodes we detect when one link node is illegally bonded to another link node, when a link node wrongly has more than two neighbors, when a link node is monovalently bonded, and finally when a link node has a tag. Much remains to be done in extending and integrating the link node concept through out the rest of congen. There also needs to be further error checking to warn users who change a superatom and then call the generate routines with out redefining composition. We have concentrated our efforts on mistakes which we have observed frequently when other chemists use CONGEN. Moreover, this error checking will serve to reduce substantially the errors made by chemists using the program.

2.7.2 Depth-First Imbedder

The IMBEDDER program was completely rewritten. Four major improvements were implemented and the efficiency of almost all of the different subsections was improved. First, the method of computation was changed from breadth first (all structures delivered at the same time) to depth first (the structures delivered one at a time as they are created). The chemist can now check the computation as it proceeds by using the cntrl-S and cntrl-I features. Use of the cntrl-S feature often will allow the chemist to see that a certain computation is much larger than he anticipated and to stop it before computer time is wasted. Use of cntrl-I allows the chemist to see if the imbedding is proceeding in producing the kind of structures he anticipated. If not, the computation can be stoped and the problem redefined with only minimal loss of human and computer time. Previously the chemist would have had to have waited excessive amounts of time before seeing any results only to find, for example, that another constraint should have been used or, for another example, that more pruning should have been done before imbedding. This new improvement will result in the saving of large amounts of computer time.

Second, all the constraints testing during imbedding is now done in the SAIL portion of CONGEN and structures violating the constraints are not returned. Previously all structures were returned to the LISP portion of CONGEN before any constraints checking and subsequent pruning were done. This new approach represents a real gain in efficiency because these programs run much faster in SAIL than they do in LISP.

Third, the canonicalization routines were rewritten. When they were first written there was the prospect that our system might be interfaced with Chemical Abstracts. With this prospect in mind, the canonicalization was done using a modified form of Morgan's algorithm which was easy for people to understand and closely related to the canonicalization algorithms used at Chemical Abstracts. Recently, the procedures were redesigned with efficiency as the only criterion. New algorithms were found and the process of assigning a canonical number to a structure is now much less costly in terms of time. Further, two structures which are aromatically equivalent are given the same key (related to its canonical number). Previously they were given different keys and special methods in Lisp were needed to insure their equality. Since many different parts of CONGEN use the canonicalizer this resulted in a gain in efficiency for all of them.

Fourth, a change was made so that any number of superatoms can be imbedded at one time. This means when large numbers of superatoms need to be imbedded the chemist can in one set of commands perform the entire task, rather than the more time consuming approach of one-at-a-time. However, the chemist can still choose for reasons of efficiency to imbed a single superatom when special tests on the environment of that superatom are required. This also provides the opportunity for large, multiple imbeddings to be done in batch mode. (The batch command was rewritten so that it would accept multiple superatoms.) The large batch job is then run after midnight when the load average is low to increase the amount of computer time for other uses.

2.7.3 EDITSTRUC Changes

The RENUMBER command was added to give the chemist flexibility in choosing schemes of numbering the atoms in the structure. There have been internal changes made to the editstruc commands BRANCH, LINK, CHAIN, and DELATS. All involve the method of numbering atoms. It is now possible to create a substructure which has "gaps"(missing numbers) in its numbering to atoms. These changes necessitated some further changes in the routines which prepare and send structures to the IMBEDDER in CONGEN.

2.7.4 BATCH

The BATCH command was rewritten to take advantage of the fact that the new lower fork programs can accept any number of Superatoms to be imbedded. As the system load continues to increase BATCH will become a more attractive option.

2.7.5 RESTORE

The RESTORE command was changed so that files written by REACT can be restored as well as files written by CONGEN. REACT users make use of the new EXAMINE subcommand (discussed elsewhere in this report) as well as the mass spectral ranking program MSRANK. Therefore it is very natural for a REACT user to save results using the SAVE subcommand in REACT and later to restore these structures in CONGEN in order to examine or rank them. The reason for the incompatibility between REACT format and CONGEN format is that REACT save files contain often many different structure lists whereas CONGEN save files contain only one. Thus the RESTORE command in CONGEN must ask the REACT user which structure list to restore.

2.7.6 TREEGEN

The TREEGEN command was rewritten to ensure that no duplicates are produced. Duplicates arise if there is a superatom on the composition list and further if from the remaining atoms a copy of that superatom or a portion of that superatom can be constructed. Another way of saying this is duplicates arise if there is a pattern in the superatom and that pattern can be built from the remaining atoms in the composition list. The new routines check for a superatom on the composition list and if there is one the canonicalization routines are called for each structure. This key is used to determine if the structure has already been added to the structure list. Several functions had to be rewritten to insure that this was done efficiently.

2.7.7 SURVEY AND EXAMINE

The command SURVEY was added to the experimental CONGEN and routines were written which allowed the chemist to look over the structure list for certain functional groups and other features defined by the. SURVEY has since been incorporated into a command called EXAMINE (see section 2.3).

2.7.8 DRAW

The draw command was extensively rewritten so that it would be more flexible. The user can either draw the whole structure list or give the command an argument which gives the range of structures to be drawn. The range given may or may not use the number associated with the structure as a key. The user can also give ranges such as 1-3 which means draw the first through the third structures on the list.

2.8 CONGEN Efficiency

We have a continuing long term commitment to improve the efficiency and the reliability of CONGEN. Algorithms under development are written quite differently from the way they should be rewritten to execute efficiently and reliably. For example, it is very natural to use free variables in the development of new code, but eventually when the function is assumed or proven to be working correctly these free variables should be eliminated to buy efficiency and modularity. LISP's inherent inefficiencies can often be circumvented by careful reprogramming. The project of block compiling CONGEN described below is a major step forward in providing an efficient and robust base for future development.

At the outset we had hoped that block compiling would speed up constrained structure generation by a factor of at least four. We ended up after a great deal of fine tuning with a speed-up factor of about two, but much higher factors in other parts of the code.

At the beginning of this project we used the new LISP subsystem Masterscope to analyze each of CONGEN's twenty one files. For each file we prepared a database for that file which contained information about its functions and their variables. These databases are important for maintenance and ease of learning CONGEN as well as for their short term purpose of facilitating block compilation. For any function on a file which we have analyzed we can ask the database which variables that function uses freely and which other functions it calls. When all the files have been analyzed we can find out which functions call a particular function.

Further we developed automatic batch programs to make and test new versions of CONGEN and to update the supporting databases. These programs can be run at night when the system is lightly loaded. This helps spread out the load on our heavily used system and leads to improved quality control in the version we supply to users since our testing can afford to be much more extensive and thorough.

Now that we have finished the work of block compiling CONGEN much testing remains to be done to insure that no new errors have been introduced. Once we are satisfied that the block compiled version is robust and reliable it will replace the version of CONGEN which we distribute to outside users. When we have reached this stage the new block compiled version will be used for development as well. Blocks which are under rapid development can be substituted into the block compiled version in their interpreted or normally compiled form. New blocks can be added without disruption of the existing program organization. Hence in gaining efficiency we have not lost flexibly.

The work on block compiling was done with the help and direction of Larry Masinter, a former member of the DENDRAL project, now with Xerox Palo Alto Research Center.

2.9 CONGEN Reprogramming

We have been investigating the reprogramming of CONGEN into an Algol-like language. The goals of reprogramming are threefold: first, to unify the program into a single language which can be used on a variety of computer systems; second, to begin to compact the program into a manageable, cost-effective size for current time-sharing systems; and third, to improve typical runtimes for CONGEN so that it becomes a more attractive means for scientists to solve structure elucidation problems. A version of CONGEN which fulfills these goals would be useful on a variety of computer systems and could be exported to many different chemical and biochemical laboratories.

2.9.1 Unification Into a Single Language

CONGEN is currently coded in three different programming languages. The constrained cyclic structure generator, which is the basic algorithm responsible for the generation of structures, the entire user interface and a number of control routines necessary to support communication between the three languages are all coded in Interlisp. Interlisp is a list-processing language, and the sections of CONGEN written in this language are heavily oriented toward the use of lists as data structures. The part of the program which deals with the drawing of structures, either on a teletype or on a graphics terminal, is coded in FORTRAN. The remaining parts of CONGEN, including those parts of the program responsible for obtaining final structures from intermediate representations and various routines to support these functions, are all written in SAIL, an ALGOL-60 variant designed for the PDP-10 computer.

Although it would be possible to emulate Interlisp's list processing facilities using, for example, the REFERENCE and RECORD capabilities of SAIL, initial timing tests have shown that no significant increase in speed could be obtained by such a move. It is felt that this is a reflection of the fact that Interlisp is tuned for list processing, and that SAIL merely provides list processing as a "add-on" feature to ALGOL.

We believe that it will be possible to unify CONGEN into one ALGOL-like language by utilizing data structures more suitable in such a language. It would be desirable to use a language which provides as little overhead as possible to the size of a running program. Although the mathematics of the algorithms in CONGEN is quite complex, the algorithms themselves make no complex demands on a programming language. Our initial choice of language which we are exploring as a vehicle for reprogramming, BCPL, is discussed in more detail in a subsequent section.

2.9.2 Compaction Into a Manageable Size.

The Sumex computer system facilitates rapid development of complex programs: a virtual memory, paging environment with a full 256K core image available to each of CONGEN's different language segments. We estimate that a fairly direct translation (i.e., with a minimum of redesign of the algorithms beyond replacement of lists with arrays) would likely result in a program requiring approximately 300K words of memory in which to run. This is too large, even with extensive use of overlays.

With a certain amount of theoretical work, we can develop an algorithm related to one discussed by Sasaki. We have made significant progress on this problem (see below). The algorithm will need to be mathematically proven and made suitable to handle the majority of problems with which CONGEN is typically presented.

Using an adaptation of the Sasaki algorithm, and redesigning the current SAIL and FORTRAN portions of CONGEN, we expect that an overlaid version of the new CONGEN would need on the order of 20-30K words of core to run on a PDP-10.

Our preliminary estimates are of course subject to uncertainty. They presume an external device (random access disk) for storage of structures. Large problems (1-2000 structures) would require temporary files totaling about 100 pages (512/PDP-10 words per page). The whole package in one core image would require 51K words. Any mechanism for overlays would make the largest core image required about 21K.

2.9.3 Improvement in Runtime

In addition to decreasing the program size, it is also necessary to minimize execution time. An experimental version of the algorithm mentioned above has been written in BCPL. We have obtained initial timing information for this algorithm and compared results with the current CONGEN. The test cases used were representative of the types of problems with which the program will be confronted. The new generating algorithm is designed to replace the Interlisp structure generator. For the typical problem involving real structures, the generation problem deals with Superatoms. Thus, the empirical formulas in Table IV represent a whole class of problems. For example, the time to perform $C_2H_2N_2O_2$ represents the time for not only isomers of this formula, but also the time for any problem with two tetravalent, two trivalent and two bivalent Superatoms together with two hydrogen atoms.

Table IV. Preliminary Timing Estimates[a] for Isomer Generation

Empirical formula	Number of Isomers	Time in seconds for generation Interlisp CONGEN	BCPL algorithm
$C_2H_2N_2O_2$	506	233	10.0
C_6H_6	217	113	9.5
N_{10}	91	52	70.5

[a] The times were obtained by running the respective test cases on lightly loaded DEC KI-10's. The values obtained for the CONGEN in Interlisp were obtained on a system operating under Tenex. The values obtained for the new BCPL algorithm were obtained on a system operating under DEC's TOPS-10 operating system. The values presented are the average of three runs, but must be viewed only as approximate because of variations in system overhead, e.g., paging, expected during normal use.

The timing values in Table IV are what we expected given our knowledge of the two algorithms. It is characteristic of a Sasaki-like algorithm that as the number of atoms (or Superatoms in our implementation) of the same type increases, execution time increases exponentially. The considerations of symmetry built into the Interlisp version treat such problems more efficiently, so the increase in execution time is somewhat less than exponential. There is a point where the efficiency of both algorithms would be the same. This is illustrated by the data in Table IV. With diverse atom types the BCPL algorithm is 23 times faster for the example case $C_2H_2N_2O_2$. With six tetravalent atoms or Superatoms of the same type, the BCPL version is only about ten times faster. For N_{10}, where all atoms or Superatoms are of the same type and the same degree (same number of non-hydrogen neighbors) the Interlisp version is faster. It is characteristic of most problems with which CONGEN is confronted that there is a diversity of types of Superatoms. In our opinion, the factor of 10-25 in increased speed for such problems justifies using the new algorithm.

Work is also currently underway on a separate imbedding package, similar to the package in the current CONGEN, but restructured for efficiency. This, together with the structure drawing program will place all of CONGEN in a common language. We can report that as of Feb. 5, 1978, the first version of the imbedding algorithm in BCPL is running. Work is now under way to compare results with the production version of CONGEN to ensure the accuracy and reliability of the new imbedder.

2.9.4 Choice of Language for Reprogramming

Recursion seems essential to the clear phrasing of most of the algorithms, both in future development work and in the initial reprogramming effort. Since provision for recursion in FORTRAN is usually by an add-on package or other such assembly of special routines, and since this facility is not available on the machine on which CONGEN is being developed, FORTRAN is not viewed as a likely candidate as a language choice. In a similar vein, many ALGOL implementations are quite inefficient in handling recursion. In many of these ALGOL implementations, due to the fact that any function is allowed to be recursive, one must pay the price of recursion even for non-recursive portions of the program. Two ALGOL based languages which are exceptions to this generalized method for handling recursive routines are SAIL and BCPL. SAIL requires one to explicitly declare a procedure as recursive, and then to use a different calling technique than is used for non-recursive procedures. SAIL also provides more explicit control over allocation of recursive variables.

BCPL is a "mini-ALGOL" with the same block structure and looping statements as ALGOL but with much more limited semantics. BCPL compilers are generally small and simple, and are available on a variety of machines. We feel that there are three advantages to BCPL. First, because its structure is simple one is to some extent insulated from collapse of or changes in the supported compilers. It would probably not take over a month for someone experienced in compiler implementation to construct a basic BCPL compiler from scratch. Thus, if programs are restricted to a fairly pure form of the language, even a total removal of support from the language by outside agencies would not be fatal to those programs. Secondly, the BCPL run-time system is generally quite small, and adds little overhead to the size of a running program (on the order of a few thousand words of core storage, compared to a few tens of thousand words of core storage for SAIL). Lastly, because BCPL is closely related to both SAIL and ALGOL, it would be fairly simple and largely mechanical to convert the BCPL code into its SAIL or ALGOL equivalent, if such a translation proved necessary or desirable.

Largely as a result of the effort required to analyze adequately the questions posed by the conversion study, work has already begun on the initial stages of CONGEN reorganization and translation. In an effort to study the the effect of BCPL, the target language, on program efficiency, as well as to study the speed of the generation algorithm described above a modified implementation of the algorithm was coded into BCPL. This code formed the basis for the estimates of speed and size improvements described previously.

2.9.5 A Version of CONGEN for the Chemical Information System

We have recently been discussing the prospects of a version of CONGEN available to the public on a fee-for-service basis as part of the NIH/EPA supported Chemical Information System (CIS). Discussions with Dr. Steve Heller, EPA, and Dr. William Milne, NIH, several months ago resulted in a contract with Stanford University to investigate the feasibility of translation of CONGEN into a language which could eventually be supported by the CIS. The outcome of this study was that such a translation was possible given the limited goals of the task. The previous section on reprogramming languages and progress was taken in part from the results of that study. We are currently drafting a detailed contract proposal to carry out the translation.

The limited goals include providing CIS with a version of the CONGEN program including some (but not all) of its current capabilities. This version is to be written in an Algol-like language and must run on the Division of Computer Resources and Technology's (DCRT's) DEC PDP-10 at NIH.

It is important to contrast these limited goals with the more ambitious objectives of reprogramming as discussed in the previous Section: 1) The translation for the CIS will produce a version which will run only on the DCRT/NIH system. It will presumably also run on similar PDP-10's using the TOPS-10 operating system. This, however, represents only a very small subset of computer systems available to the chemical community. Thus the CIS version will not meet our expressed objective of a version of CONGEN which is exportable to a large number of research groups; 2) the translation for the CIS will utilize the Basic Combined Programming Language (BCPL). This is an Algol-like language which will suffice for the CIS version. It is not clear that this language is optimum for a program which is to be widely distributed. The development of a more machine-independent language, such as MAINSAIL at SUMEX, may provide a much better vehicle for wide distribution, in which case our efforts under the current DENDRAL grant would be directed toward that language; and 3) the contract with CIS will have very limited provision for introducing new improvements in CONGEN and related programs (see Sections 2.2 and 2.4) to the DCRT/NIH version of CONGEN in BCPL. It is obviously essential to provide the chemical community with the most up-to-date version of CONGEN and related programs. Work under the present grant is directed at this broader goal.

At the same time there are obvious similarities to the CONGEN translation effort supported by CIS and the NIH under the current DENDRAL grant. We would be foolhardy to view these efforts as mutually exclusive. The major similarity is that the choice of language is subject to similar restrictions, meaning that some ALGOL-like language would be used for the exportable version for wider distribution. We feel that a translation of parts of the program, for example from BCPL into MAINSAIL, is a relatively simple task given the similarities of the languages. The translation efforts would, we feel, be synergistic, providing more rapid access of the chemical community to CONGEN and other programs.

3 THEORY FORMATION PROGRAMS - Meta-DENDRAL

3.1 Incremental Learning

In order to allow applying the Meta-DENDRAL program[3] to a wider range of chemically interesting problems, we have begun to remove one of the most important current program limitations - its inability to add piecewise to what it has learned. Meta-DENDRAL must currently process all training data at once, producing a set of rules which cover that data. The amount of training data which the program may examine when forming rules is therefore limited by available computer memory. We aim to give the program the ability to learn incrementally resulting in the following benefits:

1) The chemist will be able to generate rules from one set of training data, examine the rules, and if necessary obtain additional data for modifying and adding to the rule set. By examining the partial results produced at each step, the chemist may determine which additional training molecules are most appropriate for the next learning cycle. We expect the program to aid in making this decision by suggesting new training molecules whose spectra will resolve among rules which represent alternate plausible explanations of the observed data.

2) Since the amount of training data processed strongly influences the reliability of the learned rules, training on arbitrarily large data sets will allow Meta-DENDRAL to form more accurate rule sets than currently feasible.

3.1.1 The Approach

The proposed approach to incremental learning involves hypothesizing a set of rules on the basis of existing training data, then updating the rule set when new training data is provided. When existing rules apply incorrectly to new training molecules, these rules are modified. New rules are added to the rule set by applying the current one-pass Meta-DENDRAL program to the portion of the new training data which cannot be explained by existing rules. The figure below summarizes the proposed approach.

```
--------------------------------------------------------------
|                                   ----------------         |
|                                   | New data     |         |
|                                   ----------------         |
|                                          |                 |
|                                          v                 |
|         -----------          -----------------------------   |
|         | Current |--------->|                          |   |
|         |  rule   |          |  Modify current rules, and  |  |
|         |  set    |<---------| filter out "explained" data |  |
|         -----------          |                          |   |
|              ^               |                          |   |
|              |               ----------------------------    |
|              |                          |                  |
|              |  Rules                   | unexplained      |
|              |  covering                |    data          |
|              |  new data                v                  |
|              |               -----------------------------   |
|              |               |                          |   |
|              ----------------|     Current Meta-DENDRAL |   |
|                              |                          |   |
|                              -----------------------------   |
--------------------------------------------------------------
```

Figure 8. Approach to Incremental Learning

3.1.2 Modifying Existing Rules

Rules must be modified so that they become consistent with the new data while remaining consistent with previous data as well. In short, the method involves storing along with each rule a summary of alternate acceptable versions of the rule (those with the same evidential support in the observed training data). The summary of all acceptable versions of a given rule, refered to as the version space[12] of the rule, is useful for a number of tasks associated with rule learning, including incremental learning.

Version spaces provide an explicit representation of the space of all alternate versions of a given rule - i.e. those which cannot be disambiguated by the currently observed training data. As such, version spaces will allow Meta-DENDRAL to reason more thoroughly with the choice among alternate rule versions. Some expected benefits and uses of this increased ability are listed below.

Modifying current rules using new training data. Since version spaces provide a summary of all alternate versions of a given rule which are consistent with previous evidence associated with the rule, they delimit the range of allowed future modifications to the rule. Thus, those modifications to the rule which are consistent with past data are exactly those which yield versions of the rule contained in the version space. At any point, the version space contains all rule versions which are consistent with a given set of evidence. The "best" rule version can then be chosen (e.g.,. on the basis of simplicity or chemical plausibility) from the entire space of rule versions consistent with the data.

More complete exploration of alternate rules. The current RULEMOD portion of Meta-DENDRAL tries to make rules more general or more specific in order to improve their performance on the training data. RULEMOD tries many such ways of modifying rules, but it cannot afford to try all ways. This portion of RULEMOD will be replaced by a new routine which will use version spaces to explore all ways of generalizing or further specifying rules in order to eliminate negative evidence or add additional positive evidence.

Intelligent selection of training instances. Since version spaces represent the range of plausible alternate versions of a rule, they contain the information needed to select new training instances designed to discriminate among competing rule versions. For instance, by examining a given version space the program ought to be able to suggest a set of compounds whose spectra would allow ruling out many of the plausible rule versions while strengthening the evidence associated with other versions.

3.1.3 Version Spaces

This section presents a sample version space generated by Meta-DENDRAL, and discusses how version spaces may be updated to take into account new training data. Notice that the program is dealing not with a single rule which will be later modified, but with the space of all plausible rule versions. The algorithm for updating version spaces using new training data is a **candidate elimination** algorithm: candidate rule versions are eliminated from the version space as they are found to perform incorrectly on new training data. The candidate elimination algorithm is assured of finding all rule versions consistent with a given set of positive and negative training instances. This is accomplished without backtracking and independent of the order of presentation of the training instances.

3.1.3.1 Definition and Representation

The key to an efficient representation of version spaces lies in observing that a general-to-specific ordering is defined on the space of rule subgraphs. The version space may be represented in terms of its maximal and minimal elements according to this ordering.

To see exactly how the general-to-specific ordering comes about, consider an example. Suppose that R1 and R2 are two rules which predict the same action. Then R1 is said to be **more specific** (or, equivalently, **less general**) than R2 if and only if it will apply to a proper subset of the instances in which R2 will apply. This definition is simply a formalization of the intuitive ideas of "more specific" and "less general".

The general-to-specific ordering will in general be a partial ordering; that is, given any two rules we cannot always say that one is more general than the other. Therefore, when all elements of the version space are ordered according to generality, there may be several maximally general and maximally specific versions.

Version spaces can be represented by these sets of **maximally general versions**, MGV, and **maximally specific versions**, MSV. Given such a representation it is quite easy to determine whether a given rule belongs to a given version space. A rule statement belongs to the version space of a given set of positive training instances and negative training instances if and only if it is (1) less general than or equal to one of the maximally general versions, and (2) less specific than or equal to one of the maximally specific versions. Condition (1) assures that the rule cannot match any training instance in I-, while condition (2) assures that it will match every training instance in I+. Since the sets MGV and MSV are by definition complete, (1) and (2) will be necessary as well as sufficient conditions for membership of a rule statement in the version space.

3.1.3.2 Example From C13 NMR Rule Formation

Meta-DENDRAL has been used to determine rules associating substructures of molecules with data peaks in a carbon-13 nuclear magnetic resonance spectrum [11]. Figure 9 shows a version space represented by the program in terms of the sets of maximally specific rule versions (rule MSV1) and maximally general rule versions (rules MGV1 and MGV2). This version space contains all rules which predict a CMR shift of from 14.0 to 14.7 ppm. downfield from TMS and which are consistent with a set of paraffin and acyclic amine data presented to the program. The rule pattern which expresses the conditions for application of each rule is stated in the language of chemical subgraphs. Each node in the subgraph represents an atom in a molecular structure. Each subgraph node has the four attributes shown, with values constrained as shown in Figure 9.

Rule Subgraph		Constraints on Subgraph Node Attributes			
subgraph	node name	atom type	number of non-hydrogen neighbors	number of hydrogen neighbors	number of unsaturated electrons
MSV1:					
v-w-x-y-z	v	carbon	1	3	0
	w	carbon	2	2	0
	x	carbon	2	2	0
	y	carbon	2	2	0
	z	carbon	>=1	any	0
MGV1:					
v-w-x	v	carbon	1	any	any
	w	any	2	any	any
	x	any	>=1	2	any
MGV2:					
v-w-x	v	carbon	1	any	any
	w	any	2	any	any
	x	any	2	any	any

Figure 9. A Version Space Represented by It's Extremal Sets