Comput. Chem. Vol. 11, No. 4, pp. 281–297, 1987
Printed in Great Britain

0097-8485/87 $3.00 + 0.00
Pergamon Journals Ltd

CHEM—A PROGRAM FOR PHOTOTYPESETTING CHEMICAL STRUCTURE DIAGRAMS

JON L. BENTLEY, LYNN W. JELINSKI and BRIAN W. KERNIGHAN
AT&T Bell Laboratories, Murray Hill, NJ 07974, U.S.A.

(*Received* 15 *December* 1986)

Abstract—CHEM is a language for typesetting publication-quality chemical structure diagrams. It attempts to capture the way that a chemist would describe a structural formula to a colleague over the telephone. CHEM is implemented as a PIC [Kernighan, *Software—Pract. Exper.* **12**, 1 (1982)] preprocessor, and runs under the UNIX operating system. Its textual input makes CHEM input-device independent and does not require a graphics terminal. CHEM diagrams can be integrated with text, and can also be used to produce slides and viewgraphs. CHEM is best suited for organic chemists, natural products chemists, and polymer chemists, although it can also be used for inorganic structure diagrams.
This manuscript describes the design of CHEM and provides a tutorial on its use.

INTRODUCTION

Chemists communicate their findings largely through chemical structure diagrams, with such drawings forming an integral part of manuscripts, lectures, and grant proposals. As the trend toward electronic manuscript submission increases, driven both by efficiency and by reduction or elimination of page charges,* a facility for typesetting chemical structure diagrams becomes an essential tool for the chemist. The ability to integrate high-quality chemical structure diagrams with text is also necessary for complete "desk-top publishing" packages.

We report here on the implementation of CHEM, a language for producing publication-quality chemical structure diagrams. It has provisions for variable bond lengths and angles, rings, heterocycles and stereochemistry. CHEM differs from other programs for drawing chemical structures in that it is input-device independent, making it portable to university and industrial environments. A CHEM input file is simply a text file; embedded within it may be descriptions of chemical structures, expressed in a language that uses familiar terms like benzene, bond, OCH3 and ring. CHEM is a PIC preprocessor, similar to EQN for equations and TBL for tables, that operates under the UNIX environment. When the document is processed, CHEM converts the descriptions into commands in the PIC picture-drawing language, which in turn converts them into commands for the TROFF formatter. Because CHEM is a PIC preprocessor, the user can revert to PIC for constructing unusual molecules that are not provided for by CHEM itself. [For a survey of document preparation tools, including those of the UNIX system, see Furuta *et al.* (1982). For a discussion of the UNIX system, see Kernighan & Pike (1984).]

This paper is divided into four parts. The design strategy is described in the first section, where CHEM's development is put into the context of an ongoing attempt to construct "little languages"—specialized languages for narrowly focused domains (Aho *et al.*, 1986; Bentley, 1986). The second section, which contains a description of the language and numerous examples, is followed by a brief discussion. The final section is an appendix that contains the AWK (Aho *et al.*, 1988) implementation of CHEM.

All of the diagrams in this manuscript were produced directly by CHEM.

THE DESIGN OF CHEM

The development of "little languages" is a growing trend in computer science. Such languages are tailor-made for a particular restricted application. On the UNIX system, particularly, examples abound, including: YACC, a parser-generator language; PIC, a language for typesetting figures and diagrams (used by CHEM); EQN, a language for typesetting mathematical expressions; and a variety of graphics languages. The task of phototypesetting chemical structure diagrams seemed especially well-suited for a "little language" because chemists normally use a specialized language when describing structure diagrams. For example, among chemists terms like benzene, ring, bond, CH_3, and N have universal

* For example, some of the American Institute of Physics (AIP) journals such as *Physical Reviews* and the *Journal of Mathematical Physics* waive page charges for manuscripts that are submitted on tape. The *Biophysical Journal* reduces page charges, the exact amount depending upon whether the manuscript is transmitted to the publisher by floppy disk, magnetic tape, or electronically.

meanings. Our goal was to capture the natural language of chemists within the context of a new "little language." In addition to being easy to learn and use, CHEM (1) must provide publication-quality diagrams suitable for camera-ready copy for books and journals; (2) should be input-device independent, and therefore not require a graphics terminal; and (3) must have an "escape mechanism" for drawing unusual or difficult structures that are not provided for by the language itself.

There are already a number of schemes for computer representation of chemical structure diagrams (Ash *et al.*, 1985), perhaps the best-known of which is the Wiswesser Line Notation (Wiswesser, 1982). Although Wiswesser notation is an extremely efficient way to represent complex chemical structures, there are ambiguities in retrieving the structural diagram from the Wiswesser linear representation. This and other linear representations are therefore not suitable for drawing chemical structure diagrams. More recently connection tables have been used to represent chemical structures (Chemical Abstracts Service, 1986); the Chemical Abstracts Service maintains over 6 million chemical substances in the form of computer-searchable connection tables (Ash *et al.*, 1985). These have the advantage of representing stereochemistry of chiral centers. However, the massive amount of storage space required for such representations makes them generally infeasible for phototypesetting chemical structure diagrams. Moreover, the chemist often wants to make a particular point by drawing a structure in a particular way. For example, when referring to an electrophilic substitution reaction it may be desirable to represent the aromatic ring with conjugated double bonds, rather than with a circle as an indicator of aromatic character. For these reasons it did not seem satisfactory to build a drawing tool based on a linear or a connection-table representation of the structure.

A number of commercial software packages for producing chemical structure diagrams are available for personal computers (*Science*, 1986), and the recent literature also contains descriptions of interactive computer programs for this purpose (Watt & Kao, 1985; Kao & Watt, 1986; Marshall, 1986). These programs have the advantage that chemists can "draw" structures on the CRT screen much as they have drawn them by hand in the past. Although these approaches produce respectable diagrams, they fall short of the quality necessary for camera-ready copy.

Essentially no software is available for typesetting chemical structure diagrams for microcomputers and for mainframes (Dzonova-Jerman-Blazic & Trinajstic, 1982), although much sofware is available for three-dimensional molecular graphics, especially for biomolecules (Diamond, 1984; Rowlett, 1985; Todd *et al.*, 1983–84). These systems produce high-quality output (usually in color) and generally require expensive terminals and graphics devices (Pensak, 1983; Max *et al.*, 1981; Marsili *et al.*, 1983). Such systems are the method of choice for representing extremely large and complex macromolecules (Heller, 1986). However, they require that the x, y and z coordinates of all of the atoms be known, and are therefore not versatile enough for drawing the casual structure of a synthetic drug or natural product.

Publishing houses such as Academic Press and the American Chemical Society, and publications such as the *Dictionary of Organic Compounds* and the *Cambridge Structural Database* have developed in-house software and art departments to produce photo-ready chemical structure diagrams (Ash *et al.*, 1985). Most of these systems use special "tablets" and trained operators.

At present there is no portable, generally available system for typesetting high quality chemical structure diagrams suitable for camera-ready copy. CHEM addresses this deficiency.

THE CHEM LANGUAGE

CHEM was designed for chemists having little prior computer knowledge. It is implemented as an AWK program and will run on a UNIX system that has PIC and AWK. It is helpful, but not necessary, to have a passing acquaintance with PIC in order to use CHEM. Since the commands that tell the system how to draw a chemical structure are a text file, CHEM descriptions can be created at any terminal. Once created, the file that contains a CHEM description must be converted into TROFF output. To do this, the file must be run through CHEM, PIC and TROFF. For example:

```
chem filename|pic|troff > filename.out
```

takes a file that contains chemical structure diagrams and puts the formatted manuscript into filename.out. Hard copy can be obtained on any device that can print TROFF output, commonly a laser printer or typesetter.

The CHEM language is rather small. It provides for rings, heterocycles, bonds, atoms and stereochemistry. Objects are normally connected together in the order that they are written down. CHEM tries to connect them at the natural places, but provides ways to specify precisely where parts of the diagram should be placed and connected in ambiguous situations. Because CHEM is a PIC preprocessor, it is possible to include PIC statements in the middle of a diagram to draw things that are not provided by CHEM itself.

Each component of the diagram is written on a single line; blank lines and comments can be used freely to make the description easy to read later on. The commands in the text file must be surrounded by .cstart and .cend, with the "." always in the first column. (This is analogous to the use of .EQ and .EN

for equations or .TS and .TE for tables.) For example:

```
.cstart
    CH3
    bond
    CH2
    bond
.cend
```

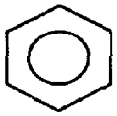
will draw the ethyl group

$$CH_3—CH_2—$$

and

```
.cstart
    benzene
.cend
```

draws a benzene ring:

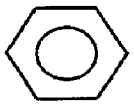


Whatever is between .cstart and .cend is converted into PIC commands by CHEM; everything outside is passed through untouched.

Sometimes it is helpful to include comments among the commands that specify the structure, especially when describing complex molecules. A comment begins with a #; any characters from there to the end of the line are ignored. For example:

```
.cstart
    benzene pointing right
    # a rotated benzene ring
.cend
```

produces the following diagram:



Bonds, directions and atoms

Bonds are specified in the following general form, where the brackets specify optional qualifiers:

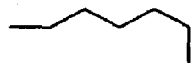
```
BONDTYPE [DIRECTION] [length N]
    [from NAME] [to NAME] [ATTRIBUTE]
```

The options must appear in this order. BONDTYPE can be bond, double bond, or triple bond; stereochemistry is specified by front bond and back bond. The default bond length is 0.2 in., but it can be adjusted by specifying a length in inches. [This feature is illustrated in the structure of isotactic poly(methyl methacrylate).] Possible bond attributes are dotted and dashed.

The *direction* of bonds in CHEM is handled in several ways. CHEM recognizes up, down, left and right, as well as the corresponding compass points N, S, W and E. It is also possible to specify the actual angle of the desired bond direction. Zero corresponds to up or N, 90 to right, −90 to left, and so on. For example, this input:

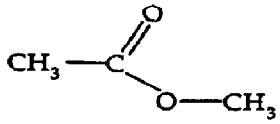
```
.cstart
    bond right
    bond 60
    bond 120
    bond 60
    bond 120
    bond down
.cend
```

produces the following stick structure:



The following diagram of methyl acetate shows how bonds can be used to connect moieties. A group or moiety (CH_3 in this example) must begin with a capital letter. Numbers are automatically converted to subscripts.

```
.cstart
CH3  # the 3 is automatically turned into a subscript
bond  # the implicit direction is right
      # implicit connection is to right side of CH3
C
double bond 30    # by default, from the substituent
                  # C
O
bond 120 from C  # must be "from C"; otherwise
                  # would leave from O
O
bond right
CH3
.cend
```



These commands could have been written more compactly by putting the substituents on the same lines as the bonds, separated by semicolons, as in:

```
.cstart
    CH3
    bond ; C
    double bond 30 ; O
    bond 120 from C ; O
    bond right ; CH3
.cend
```

The following example shows that dots are centered unless the combination appears to be a fractional value, as in N2.5H. For example:

```
.cstart
    HCl.H2O
.cend
```

produces

$$HCl \cdot H_2O$$

but attempting to do the same with $CaSO_4 \cdot 2H_2O$, i.e.

```
.cstart
    CaSO4.2H2O
.cend
```

produces

$$CaSO_{4.2}H_2O.$$

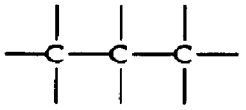
Special cases like this require the "right of" construct (see below). Normally a group is placed immediately after the last item mentioned, but it may be manually positioned by PIC-like commands, e.g.

```
CH3 at C + (0.5, 0.25)
                    # up 1/2 in., right 1/4 in.
```

Names

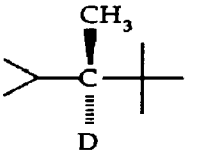
In the methyl acetate example, the carbon atom C was used both to draw something and as a *name* for a place. An atom or a group always defines a name for a place. When the group or moiety contains special characters, like the parentheses in N(C2H5)2, the name is what is left after the special characters are stripped out. In this example, the name would be NC2H52. Each new occurrence of a name overrides the definition of the previous one:

```
.cstart
    bond ; C    # 1st definition of C
    bond up from C
    bond down from C
    bond right from C ; C    # 2nd definition of C
    bond up from C
    bond down from C
    bond right from C ; C    # 3rd definition of C
    bond up from C
    bond down from C
    bond right from C
.cend
```



BP is a special name that is used to specify a "branch point," a place where no moiety is printed. For example.

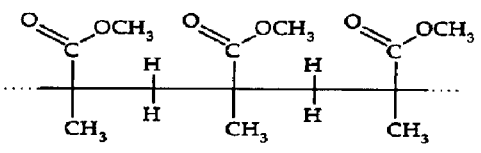
```
.cstart
# this is the isopropyl group
    bond 120 ; BP
# BP is right end of this bond
    bond -120 from BP
    bond right from BP ; C
    front bond up ; CH3
    back bond down from C ; D
    bond right from C ; BP
# redefine BP to mean the center carbon of this
    # t-butyl group
    bond up from BP
    bond right from BP
    bond down from BP
.cend
```



Note that CHEM connects bonds to the proper parts of moieties by connecting to carbon atoms where possible. If necessary it is possible to override the automatic connections to carbon (see below).

This structure of several repeat units of isotactic poly(methyl methacrylate) shows the use of bonds, directions, groups and names.

```
.cstart
    bond dotted
# begin first segment of polymer
    bond right ; BP
    bond up from BP ; C
    double bond -60 from C ; O
    bond 60 length .1 from C ; OCH3
    bond down from BP ; CH3
# begin second segment of polymer
    bond right length .5 from BP ; BP
    bond up length .1 from BP ; H
    bond down length .1 from BP ; H
# begin third segment of polymer
    bond right length .5 from BP ; BP
    bond up from BP ; C
    double bond -60 from C ; O
    bond 60 length .1 from C ; OCH3
    bond down from BP ; CH3
# begin fourth segment of polymer
    bond right length .5 from BP ; BP
    bond up length .1 from BP ; H
    bond down length .1 from BP ; H
# begin fifth segment of polymer
    bond right length .5 from BP ; BP
    bond up from BP ; C
    double bond -60 from C ; O
    bond 60 length .1 from C ; OCH3
    bond down from BP ; CH3
    bond right from BP
    bond dotted
.cend
```



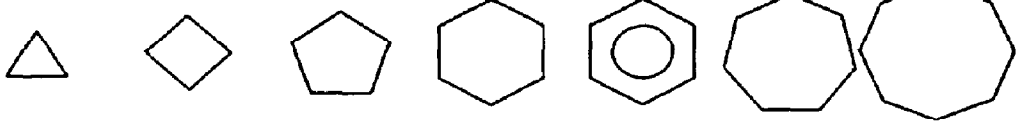
Rings

Rings are specified in the following general form:

[RING NAME:] ring[N] [DIRECTION]
[HETEROATOMS] [DOUBLEBOND PATTERN]

where the optional parts are enclosed in square brackets. Cyclopropane through cyclooctane can be obtained by ring3, ring4, and so on:

```
.cstart
R3: ring3
R4: ring4 at R3 + (.75,0)
R5: ring5 at R4 + (.75,0)
R6: ring6 at R5 + (.75,0)
B: benzene at R6 + (.75,0)
R7: ring7 at B + (.75,0)
R8: ring8 at R7 + (.75,0)
.cend
```



"ring" is a synonym for ring6. The attribute "at NAME + (.75,0)" is a PIC command that puts each object 3/4 in. to the right of the center of the previous one.

Note that all of these rings are pointing up. That is, the top-most vertex, or vertex 1, points toward the top of the page. This is especially obvious for cyclobutane (ring 4), which is oriented in a manner that at first appears strange to a chemist. The normal orientation has the "point" or vertex 1 rotated 45°, so it is written as

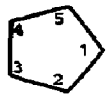
ring4 pointing 45

The vertices of rings are always numbered the same way: the "point" is the first vertex, which is called .V1, and .V2, .V3, etc., continue clockwise around the ring. For example, the following structures show that the "point" in cyclopentane, as in any other ring, is always numbered 1, but its position follows the direction in which the ring points:

ring5 (or ring5 pointing up)

ring5 pointing left

ring5 pointing down



ring5 pointing right

There are two types of 5-membered rings: the pentagonal ring5 above, and the "flatring." The flatring is useful for fusing onto the sides of other rings.

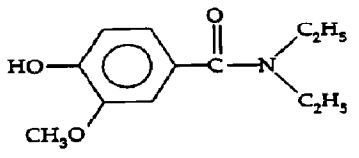
flatring pointing up flatring pointing down

It is often necessary to name the rings so that the substituents can be properly added. For example:

```
.cstart
    R:   ring3
    back bone 120 from R.V2 ; C2H5
    front bond -120 from R.V3 ; HO
.cend
```

The ring itself is named R; its second vertex is named R.V2. As another example,

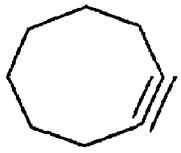
```
.cstart
    R:   benzene pointing right
         bond left from R.V4 ; HO
         bond -150 from R.V3 ; CH3O
         bond right from R.V1 ; C
         double bond up from C ; O
         bond right from C ; N
         bond 45 ; C2H5
         bond 135 from N ; C2H5
.cend
```



Double bonds within rings are specified by naming vertices between which they appear:

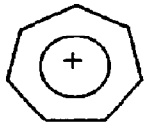
```
.cstart
    ring double 1,2, 3,4, 5,6
.cend
```

```
.cstart
    ring8 triple 3,4
.cend
```



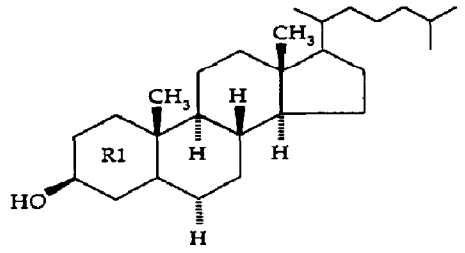
A circle can be placed inside any ring by using the modifier aromatic:

```
.cstart
R: aromatic ring7
   "+" at R
.cend
```



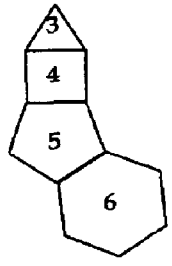
Fused ring structures are formed by specifying a common vertex on each ring. For example, consider cholestanol:

```
.cstart
R1: ring
    "R1" at R1   # this puts a label at R1
    front bond -120 from R1.V5 ; HO
        # the following line says "fuse the 6th
                                  # vertex
        # of ring R2 to the second vertex of R1"
R2: ring with .V6 at R1.V2
    front bond up from R2.V6 ; CH3
    back bond down from R2.V4 ; H
    back bond down from R2.V1 ; H
    front bond up from R2.V2 ; H
R3: ring with .V4 at R2.V2
R4: flatring with .V5 at R3.V2
    front bond up from R4.V5 ; CH3
    back bond down from R4.V4 ; H
    # this is the alkyl chain
    bond up from R4.V1 ; BP
    bond -60 from BP
    bond 60 from BP
    bond 120
    bond 60
    bond 120 ; BP
    bond down from BP
    bond 60 from BP
.cend
```



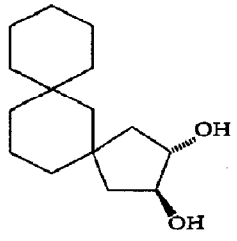
The next example shows how the different sized rings fuse together. Note that the fusion of the five- and six-membered rings requires an unusual angle:

```
.cstart
R3: ring3
R4: ring4 pointing 45 with .V1 at R3.V2
R5: ring5 pointing down with .V4 at R4.V2
R6: ring6 pointing 54 with .V6 at R5.V5
    # the following lines specify the labels inside
        # the rings
    "3" at R3
    "4" at R4
    "5" at R5
    "6" at R6
.cend
```



Spiro ring junctions are formed in an analogous way:

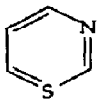
```
.cstart
    R1: ring6
    R2: ring6 with .V1 at R1.V4
    R3: ring5 with .V5 at R2.V3
        back bond 60 from R3.V2 ; OH
        front bond 150 from R3.V3 ; OH
.cend
```



Heteroatoms in rings are written as "put X at V", where X is the heteroatom and V is the vertex

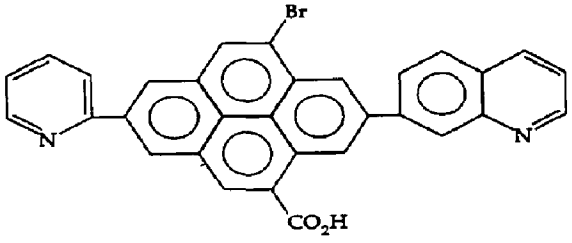
number. For example:

```
.cstart
    ring put N at 2 put S at 4 double 2,3 4,5 6,1
.cend
```



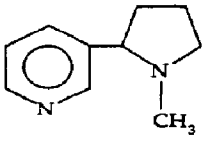
As a more complex example, this polycyclic aromatic compound is produced with the following input (note that this diagram starts with the bromo-substituted ring, but that there are many other equally good ways to start this structure):

```
.cstart
R1: benzene pointing right
    bond 30 from R1.V6 ; Br
R2: benzene pointing right with .V5 at R1.V1
R3: benzene pointing right with .V1 at R2.V3
    bond 150 from R3.V2 ; CO2H
R4: benzene pointing right with .V1 at R1.V3
# next line names bond B1 so we can refer to its
    # end
B1: bond left from R4.V4
    ring6 put N at 4 double 2,3 4,5 6,1 with .V3 at
                                        B1.end
R5: benzene with .V5 at B2.end
    ring6 put N at 4 double 1,2 3,4 with .V5 at
                                        R5.V3
.cend
```



Substituents are placed on heteroatoms just as they are placed on non-heterocycles. The nicotine molecule provides an example of bond positioning:

```
.cstart
    benzene put N at 4
    bond right
    ring5 pointing down put N at 1
    bond down from .N ; CH3      # or .V1
.cend
```

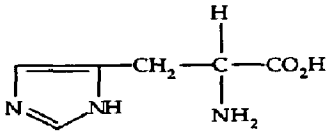


The second bond refers to .N, which is an alternative name for .V1; .N refers to the immediately preceding object, the unnamed ring5.

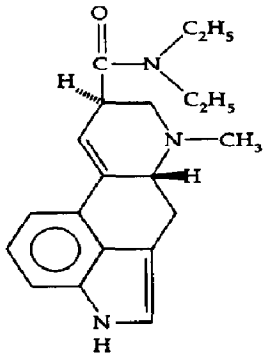
Positioning substituents

When drawing heterocyclic rings, it is often useful to position a group or atom in the vicinity of the heteroatom. This is done with the commands above, below, right of, and left of, as in the NH of the imidazole ring of histidine:

```
.cstart
R1: flatring pointing down put N at 2 put N at 5
                                        double 1,2 3,4
    H right of R1.V5
    bond right from R1.V4 ; CH2
    bond right ; C
    bond up from C ; H
    bond down from C ; NH2
    bond right from C ; CO2H
.cend
```



The lysergic acid diethylamide structure below provides another example of positioning substituents on heteroatoms:




```
.cstart
B:  benzene pointing right
F:  flatring pointing left put N at 5 double 3,4 with
                                        .V1 at B.V2
    H below F.N
R:  ring pointing right with .V4 at B.V6
    front bond right from R.V6 ; H
W:  ring pointing right with .V2 at R.V6 put N at
                                        1 double 3,4
    bond right from W.N ; CH3
    back bond -60 from W.V5 ; H
    bond up from W.V5 ; C
    double bond up from C ; O
    bond right from C ; N
    bond 45 from N ; C2H5
    bond 135 from N ; C2H5
.cend
```

We mentioned before that CHEM attempts to connect bonds to the proper parts of moieties, connecting

carbon atoms when possible. Once in a while this may provide an unwanted effect, as shown in the attempted (left) diagram of anisole. The CH_3 can be manually positioned by using right of, as shown in the diagram on the right.

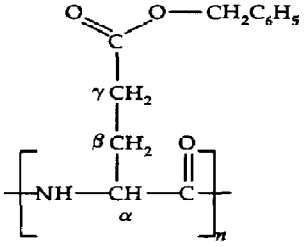
```
.cstart
R1: benzene
    bond down from R1.V4 ; OCH3
R2: benzene at R1 + (1.5,0)
    bond down from R2.V4 ; O
    CH3 right of O
.cend
```

The following diagram of reserpine shows that the connectivity to carbon is ambiguous only when the OCH_3 is connected to a bond up or a bond down.

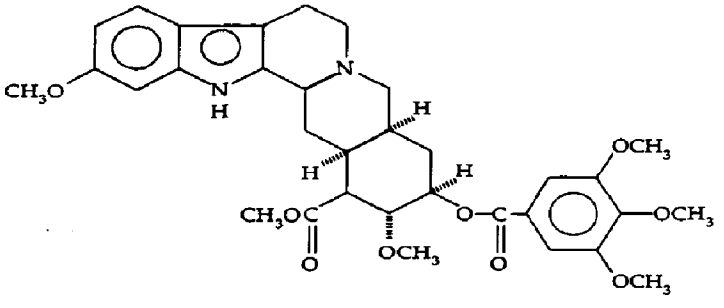
```
.cstart
    CH3O
    bond 60
R1: benzene
R2: aromatic flatring5 pointing down put N at 1
                        with.V3 at R1.V2
    H below R2.V1
R3: ring put N at 3 with .V5 at R2.V5
R4: ring put N at 1 with .V1 at R3.V3
    back bond −120 from R4.V4 ; H
    back bond 60 from R4.V3 ; H
R5: ring with .V1 at R4.V3
    bond −120 ; C
    double bond down from C ; O
    CH3O left of C
    back bond 60 from R5.V3 ; H
    back bond down from R5.V4 ; O
    CH3 right of O
    bond 120 from R5.V3 ; O
    bond right length .1 from O ; C
    double bond down ; O
    bond right length .1 from C
B:  benzene pointing right
    bond right from B ; OCH3
    bond 150 from B ; OCH3
.cend
```

Brackets and text

It is possible to make brackets of any size by using the branch point, BP. This is shown by the structure of poly(benzyl glutamic acid). This example also shows how to obtain Greek text as structure labels. The $ alpha $ is converted into α by EQN, and is enclosed in double quotes so that it will be passed from CHEM to EQN untouched. (Of course one must then add EQN to the pipeline that compiles the description.)



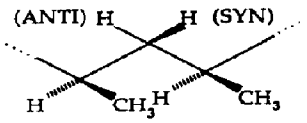

```
.cstart
# a left bracket
    bond right length .1 ; BP
    bond up length .3
    bond right length .1
    bond down length .3 from BP
    bond right length .1
#this is the mainchain amide structure
    bond right length .1 from BP ; NH
    bond right ; CH
# label the CH with an alpha, intended for eqn.
# this line says "put the north edge of the alpha
# at the south edge of the CH"
    "$ alpha $" with .n at CH.s
    bond right from CH ; C
    double bond up from C ; O
    bond right length .1 from C ; BP
# a right bracket
# a right bracket
    bond up length .3
    bond left length .1
    bond right length .1 from BP
    bond down length .3 from BP ; BP
    bond left length .1
# label the degree of polymerization
    "$n$" with .w at BP.se
```



```
# this is the sidechain
    bond up from CH ; CH2
    "$beta$" with .e at CH2.w
    bond up from CH2 ; CH2
    "$gamma$" with .e at CH2.w
    bond up from CH2 ; C
# this is the benzyl ester part
    double bond −60 from C ; O
    bond 60 from C ; O
    bond right ; CH2C6H5
.cend
```

Text can be positioned with various PIC commands, as shown in the Greek letters above, and in this example:

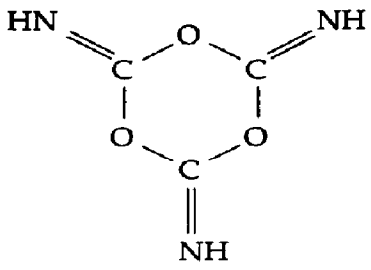
```
.cstart
    bond 120 dotted
    bond 120 length .3 ; BP
    back bond −120 length .25 from BP ; H
    front bond 120 length .25 from BP ; CH3
    bond 60 length .5 from BP ; BP
    bond −60 length .25 from BP ; H
# note the pic move command to position the text
    move left .35 ; "(ANTI)"
    front bond 60 length .25 from BP ; H
# another positioning of text
    move right .35 ; "(SYN)"
    bond 120 length .4 from BP ; BP
    back bond −120 length .25 from BP ; H
    front bond 120 length .25 from BP ; CH3
    bond 60 length .5 from BP
    bond 60 dotted
.cend
```



Changing the size of structures

The default size for CHEM is 10 point text. The size of bonds, rings, etc., can be adjusted within the .cstart and .cend pair by specifying size n, where n is the desired point size. The text size itself is not changed, however; and that must be done separately with TROFF .ps commands. The following example shows how CHEM could be used to make a diagram more suitable for a viewgraph.

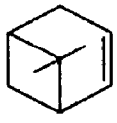
```
.cstart
.ps 14
size 16
R: ring6 put O at 1 put C at 2 put O at 3 put C
                    at 4 put O at 5 put C at 6
    double bond 6 from R.V2 ; NH
    double bond down from R.V4 ; NH
    double bond −60 from R.V6 ; HN
size 10   # this resets the size to the default value
.ps 10
.cend
```



Communicating with PIC

Since CHEM translates input commands into PIC, it is possible to use PIC commands to draw structures that are not provided for by CHEM. Some examples of PIC commands have already been introduced, such as the at commands in the section on rings. The following are several more difficult examples.

```
.cstart
R: ring double 2,3
    line from R.V6 to R.C
    line from R.C to R.V4
X1: 1/2 <R.V5,R.C>
X2: 1/2 <R.C,R.V2>
    bond from X1 to X2
.cend
```



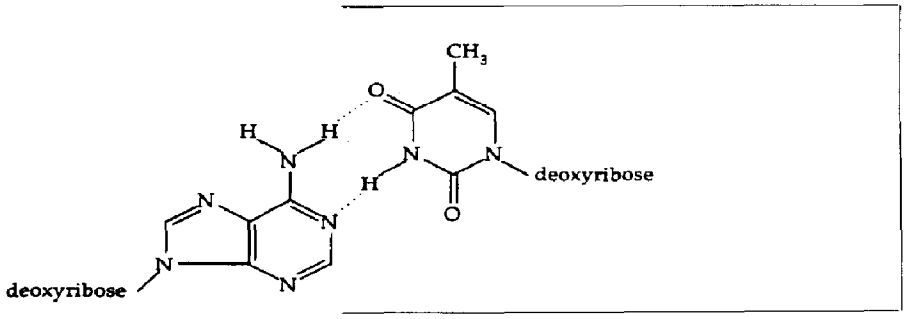
This relies on knowing that the center of a ring R is called R.C. The PIC construct $\alpha\langle p_1, p_2\rangle$ defines a point α of the way from p_1 to p_2.

The following diagram of part of DNA shows how to use PIC's bracketing construct to make two fragments that are joined together at the desired position.

```
.cstart
P: [
R1:  flatring pointing up put N at 1 put N at 4
                                    double 5,1
     bond −135 from R1.V4 ; BP
     "deoxyribose" rjust with .e at BP.w
R2:  ring6 put N at 2 put N at 4 double 1,2 3,4 5,6
                    with .V6 at R1.V2
     pic Conn: R2.V2.ne
     #because naming is too restricted in pic
     bond up from R2.V1 ; N
     bond −60 from N ; H
     bond 60 from N ; H
]
     # thymine
Q: [
R3:  ring6 put N at 3 put N at 5 double 1,2
     bond up from R3.V1 ; CH3
     bond 120 from R3.V3 ; BP
     "deoxyribose" 1just with .w at BP.e
     double bond down from R3.V4 ; O
```

```
        double bond -60 from R3.V6 ; O
        bond -120 from R3.V5 ; H
] with .O at P.H + (.3,.3)
        bond from Q.O.sw to P.H.ne dotted
        bond from Q.H.sw to P.Conn dotted
.cend
```

Bentley (1986) for a brief history] and the program is short (see Appendices 1 and 2). We found it valuable to develop a prototype quickly, then refine the language in the light of experience by real users.



DISCUSSION

CHEM has been used by about a dozen people over the past year. They find it easy to learn and to retain for long periods of time; they are also generally pleased with the quality of the output. The main drawback cited is the lack of an interactive mode.

Interactive graphics using hand/eye coordination is arguably a natural way for a chemist to draw structure diagrams. Nevertheless, we chose to develop a "little language" rather than an interactive system for several reasons. First, the language fits naturally into the UNIX environment and can be used with existing programs, so that math, tables, text, and pictures can be interspersed with the chemical diagrams. Second, the use of simple text input feeding into TROFF provides device independence and portability: CHEM works anywhere that TROFF does since the input is a text file, and the output is TROFF. Third, because the input is just text, any text editor can be used to make systematic, global changes to structure diagrams, or to make new diagrams by cut and paste from old ones. Furthermore, there is complete and accurate control of diagram positioning and size. Finally, and perhaps most importantly, interactive systems become evolutionary dead-ends, whereas one could readily envision constructing another "little language" for a special kind of diagram that would compile into CHEM itself.

Because CHEM was implemented in AWK, the development of a prototype was very rapid [see

REFERENCES

Aho A. V., Kernighan B. W. & Weinberger P. J. (1988) *The AWK Programming Language*. Addison–Wesley, Reading, Mass.

Aho A. V., Sethi R. & Ullman J. D. (1986) *Compilers: Principles, Techniques, and Tools*. Addison–Wesley, Reading, Mass.

Ash J. E., Chubb P. A., Ward S. E., Welford S. M. & Willett P. (1985) *Communication, Storage and Retrieval of Chemical Information*. Wiley, New York.

Bentley J. (1986) *Comm. ACM* **29,** 711.

Chemical Abstracts Service (1986) *CAS Online News*, January/February, p. 3.

Diamond R. (1984) *Comput. Graphics Forum* **11,** 3.

Dzonova-Jerman-Blazic B. & Trinajstic N. (1982) *Comput. Chem.* **6,** 121.

Furuta R., Scofield J. & Shaw A. (1982) *Comput. Surv.* **14,** 417.

Heller S. R. (1986) *Ind. Chem. News* **7,** 40.

Kao J. & Watt L. (1986) *Comput. Chem.* **10,** 47.

Kernighan B. W. (1982) *Software—Pract. Exper.* **12,** 1.

Kernighan B. W. & Pike R. (1984) *The Unix Programming Environment*. Prentice–Hall, Englewood Cliffs, N.J.

Marshall J. C. (1986) *J. Chem. Inf. Comput. Sci.* **26,** 87.

Marsili M., Floersheim P. & Dreiding A. S. (1983) *Comput. Chem.* **7,** 175.

Max N. L., Malhotra D. & Hopfinder A. (1981) *Comput. Chem.* **5,** 19.

Pensak D. A. (1983) *Ind. Res. Dev.* January 74.

Rowlett R. (1985) *Comput. Chem.* **9,** 301.

Science (1986) **230,** 198.

Smith E. G. & Baker P. A. (1975) *The Wiswesser Line-formula Chemical Notation (WLN)*, 3rd edition. Chemical Information Management, Cherry Hill, N.J.

Todd S., Morffew A. & Burridge J. (1983–1984) In *Brit. Nat'l. Conf. on Databases*.

Watt L. & Kao J. (1985) *Comput. Chem.* **9,** 269.

Wiswesser W. J. (1982) *J. Chem. Inf. Comput. Sci.* **22,** 88.

Appendices on following pages

APPENDIX 1

Listing of CHEM.AWK

The following is the listing of the CHEM program in AWK

```
BEGIN {
        macros = "chem.macros"
        pi = 3.141592654
        deg = 57.29578
        setparams(1.0)
        set(dc, "up 0 right 90 down 180 left 270 ne 45 se 135 sw 225 nw 315")
        set(dc, "0 n 30 ne 45 ne 60 ne 90 e 120 se 135 se 150 se 180 s")
        set(dc, "300 nw 315 nw 330 nw 270 w 210 sw 225 sw 240 sw")
}
function init() {
        printf ".PS\n"
        if (firsttime++ == 0) {
                printf "copy \"%s\"\n", macros
                printf "\ttextht = %g; textwid = .1; cwid = %g\n", textht, cwid
                printf "\tlineht = %g; linewid = %g\n", lineht, linewid
        }
        printf "Last: 0,0\n"
        RING = "R"; MOL = "M"; BOND = "B"; OTHER = "O"  # manifests
        last = OTHER
        dir = 90
}
function setparams(scale) {
        lineht = scale * 0.2
        linewid = scale * 0.2
        textht = scale * 0.16
        db = scale * 0.2                # bond length
        cwid = scale * 0.12             # character width
        cr = scale * 0.08               # rad of invis circles at ring vertices
        crh = scale * 0.16              # ht of invis ellipse at ring vertices
        crw = scale * 0.12              # wid
        dav = scale * 0.015             # vertical shift up for atoms in atom macro
        dew = scale * 0.02              # east-west shift for left of/right of
        ringside = scale * 0.3          # side of all rings
}
        { lineno++ }
/^\.cstart|^begin chem/ { init(); inchem = 1; next }
/^\.cend|^end/          { inchem = 0; print ".PE"; next }
/^\./           { print; next }         # troff
inchem == 0     { print; next }         # everything else
$1 == "pic"     { shiftfields(1); print; next } # pic pass-thru
$1 ~ /^#/       { next }        # comment
$1 == "textht"  { textht = $NF; next }
$1 == "cwid"    { cwid = $NF; next }
$1 == "db"      { db = $NF; next }
$1 == "size"    { if ($NF <= 4) size = $NF; else size = $NF/10
                  setparams(size); next }
        { print "\n#", $0 }      # debugging, etc.
        { lastname = "" }
$1 ~ /^[A-Z].*:$/ {     # label;  falls thru after shifting left
        lastname = substr($1, 1, length($1)-1)
        print $1
        shiftfields(1)
}
$1 ~ /^\"/      { print "Last: ", $0; last = OTHER; next }
$1 ~ /bond/     { bond($1); next }
$1 ~ /^(double|triple|front|back)$/ && $2 == "bond" {
                  $1 = $1 $2; shiftfields(2); bond($1); next }
$1 == "aromatic" { temp = $1; $1 = $2; $2 = temp }
$1 ~ /ring|benz/ { ring($1); next }
$1 == "methyl"  { $1 = "CH3" }  # left here as an example
$1 ~ /^[A-Z]/   { molecule(); next }
$1 == "label"   { label(); next }
/./     { print "Last: ", $0; last = OTHER }
END     { if (firsttime == 0) error("did you forget .cstart and .cend?")
          if (inchem) printf ".PE\n"
}
function bond(type,     i, goes, from) {
        goes = ""
        for (i = 2; i <= NF; i++)
                if ($i == ";") {
                        goes = $(i+1)
                        NF = i - 1
                        break
                }
        leng = db
        from = ""
        for (cf = 2; cf <= NF; ) {
                if ($cf ~ /(\+|-)?[0-9]+|up|down|right|left|ne|se|nw|sw/)
                        dir = cvtdir(dir)
                else if ($cf ~ /^leng/) {
                        leng = $(cf+1)
                        cf += 2
                } else if ($cf == "to") {
                        leng = 0
                        from = fields(cf, NF)
                        break
                } else if ($cf == "from") {
                        from = dofrom()
```

```
                        break
                } else if ($cf ~ /^#/) {
                        cf = NF+1
                        break;
                } else {
                        from = fields(cf, NF)
                        break
                }
        }
        if (from ~ /( to )|^to/)        # said "from ... to ...", so zap length
                leng = 0
        else if (from == "")    # no from given at all
                from = "from Last." leave(last, dir) " " fields(cf, NF)
        printf "Last: %s(%g, %g, %s)\n", type, leng, dir, from
        last = BOND
        if (lastname != "")
                labsave(lastname, last, dir)
        if (goes) {
                $0 = goes
                molecule()
        }
}
function dofrom(        n, s) {
        cf++    # skip "from"
        n = $cf
        if (n in labtype)       # "from Thing" => "from Thing.V.s"
                return "from " n "." leave(labtype[n], dir)
        if (n ~ /^\.[A-Z]/)     # "from .V" => "from Last.V.s"
                return "from Last" n "." corner(dir)
        if (n ~ /^[A-Z][^.]*\.[A-Z][^.]*$/)     # "from X.V" => "from X.V.s"
                return "from " n "." corner(dir)
        return fields(cf-1, NF)
}
function molecule(      n, type) {
        n = $1
        if (n == "BP") {
                $1 = "\"\" ht 0 wid 0"
                type = OTHER
        } else {
                $1 = atom(n)
                type = MOL
        }
        gsub(/[^A-Za-z0-9]/, "", n)     # for stuff like C(OH3): zap non-alphanum
        if ($2 == "")
                printf "Last: %s: %s with .%s at Last.%s\n", \
                        n, $0, leave(type,dir+180), leave(last,dir)
        else if ($2 == "below")
                printf("Last: %s: %s with .n at %s.s\n", n, $1, $3)
        else if ($2 == "above")
                printf("Last: %s: %s with .s at %s.n\n", n, $1, $3)
        else if ($2 == "left" && $3 == "of")
                printf("Last: %s: %s with .e at %s.w+(%g,0)\n", n, $1, $4, dew)
        else if ($2 == "right" && $3 == "of")
                printf("Last: %s: %s with .w at %s.e-(%g,0)\n", n, $1, $4, dew)
        else
                printf "Last: %s: %s\n", n, $0
        last = type
        if (lastname != "")
                labsave(lastname, last, dir)
        labsave(n, last, dir)
}
function label( i, v) {
        if (substr(labtype[$2], 1, 1) != RING)
                error(sprintf("%s is not a ring", $2))
        else {
                v = substr(labtype[$2], 2, 1)
                for (i = 1; i <= v; i++)
                        printf("\"\\s-3%d\\s0\" at 0.%d<%s.C,%s.V%d>\n", i, v+2, $2, $2, i)
        }
}
function ring(type,     typeint, pt, verts, i) {
        pt = 0  # points up by default
        if (type ~ /[1-8]$/)
                verts = substr(type, length(type), 1)
        else if (type ~ /flat/)
                verts = 5
        else
                verts = 6
        fused = other = ""
        for (i = 1; i <= verts; i++)
                put[i] = dbl[i] = ""
        nput = aromatic = withat = 0
        for (cf = 2; cf <= NF; ) {
                if ($cf == "pointing")
                        pt = cvtdir(0)
                else if ($cf == "double" || $cf == "triple")
                        dblring(verts)
                else if ($cf ~ /arom/) {
                        aromatic++
                        cf++    # handled later
                } else if ($cf == "put") {
                        putring(verts)
                        nput++
```

```
                } else if ($cf ~ /^#/) {
                        cf = NF+1
                        break;
                } else {
                        if ($cf == "with" || $cf == "at")
                                withat = 1
                        other = other " " $cf
                        cf++
                }
        }
        typeint = RING verts pt         # RING | verts | dir
        if (withat == 0)
                fused = joinring(typeint, dir, last)
        printf "Last: [\n"
        makering(type, pt, verts)
        printf "] %s %s\n", fused, other
        last = typeint
        if (lastname != "")
                labsave(lastname, last, dir)
}
function makering(type, pt, v,       i, a, r) {
        if (type ~ /flat/)
                v = 6
    # vertices
        r = ringside / (2 * sin(pi/v))
        printf "\tC: 0,0\n"
        for (i = 0; i <= v+1; i++) {
                a = ((i-1) / v * 360 + pt) / deg
                printf "\tV%d: (%g,%g)\n", i, r * sin(a), r * cos(a)
        }
        if (type ~ /flat/) {
                printf "\tV4: V5; V5: V6\n"
                v = 5
        }
    # sides
        if (nput > 0) { # hetero ...
                for (i = 1; i <= v; i++) {
                        c1 = c2 = 0
                        if (put[i] != "") {
                                printf("\tV%d: ellipse invis ht %g wid %g at V%d\n",
                                        i, crh, crw, i)
                                printf("\t%s at V%d\n", put[i], i)
                                c1 = cr
                        }
                        j = i+1
                        if (j > v)
                                j = 1
                        if (put[j] != "")
                                c2 = cr
                        printf "\tline from V%d to V%d chop %g chop %g\n", i, j, c1, c2
                        if (dbl[i] != "") {     # should check i<j
                                if (type ~ /flat/ && i == 3) {
                                        rat = 0.75; fix = 5
                                } else {
                                        rat = 0.85; fix = 1.5
                                }
                                c1 = c2 = 0
                                if (put[i] != "")
                                        c1 = cr/fix
                                if (put[j] != "")
                                        c2 = cr/fix
                                printf "\tline from %g<C,V%d> to %g<C,V%d> chop %g chop %g\n",
                                        rat, i, rat, j, c1, c2
                                if (dbl[i] == "triple")
                                        printf "\tline from %g<C,V%d> to %g<C,V%d> chop %g chop %g\n",
                                                2-rat, i, 2-rat, j, c1, c2
                        }
                }
        } else {        # regular
                for (i = 1; i <= v; i++) {
                        j = i+1
                        if (j > v)
                                j = 1
                        printf "\tline from V%d to V%d\n", i, j
                        if (dbl[i] != "") {     # should check i<j
                                if (type ~ /flat/ && i == 3) {
                                        rat = 0.75
                                } else
                                        rat = 0.85
                                printf "\tline from %g<C,V%d> to %g<C,V%d>\n",
                                        rat, i, rat, j
                                if (dbl[i] == "triple")
                                        printf "\tline from %g<C,V%d> to %g<C,V%d>\n",
                                                2-rat, i, 2-rat, j
                        }
                }
        }
    # circle
        if (type ~ /benz/ || aromatic > 0) {
                if (type ~ /flat/)
                        r *= .4
                else
                        r *= .5
```

```
                printf "\tcircle rad %g at 0,0\n", r
        }
}
function putring(v) {   # collect "put Mol at n"
        cf++
        mol = $(cf++)
        if ($cf == "at")
                cf++
        if ($cf >= 1 && $cf <= v) {
                m = mol
                gsub(/[^A-Za-z0-9]/, "", m)
                put[$cf] = m ":" atom(mol)
        }
        cf++
}
function joinring(type, dir, last) {    # join a ring to something
        if (substr(last, 1, 1) == RING) {       # ring to ring
                if (substr(type, 3) == substr(last, 3)) # fails if not 6-sided
                        return "with .V6 at Last.V2"
        }
        # if all else fails
        return sprintf("with .%s at Last.%s", \
                leave(type,dir+180), leave(last,dir))
}
function leave(last, d,         c, c1) {        # return vertex of last in dir d
        if (last == BOND)
                return "end"
        d = reduce(d)
        if (substr(last, 1, 1) == RING)
                return ringleave(last, d)
        if (last == MOL) {
                if (d == 0 || d == 180)
                        c = "C"
                else if (d > 0 && d < 180)
                        c = "R"
                else
                        c = "L"
                if (d in dc)
                        c1 = dc[d]
                else
                        c1 = corner(d)
                return sprintf("%s.%s", c, c1)
        }
        if (last == OTHER)
                return corner(d)
        return "c"
}
function ringleave(last, d,     rd, verts) {    # return vertex of ring in dir d
        verts = substr(last, 2, 1)
        rd = substr(last, 3)
        return sprintf("V%d.%s", int(reduce(d-rd)/(360/verts)) + 1, corner(d))
}
function corner(dir) {
        return dc[reduce(45 * int((dir+22.5)/45))]
}
function labsave(name, type, dir) {
        labtype[name] = type
        labdir[name] = dir
}
function dblring(v,     d, v1, v2) {    # should canonicalize to i,i+1 mod v
        d = $cf
        for (cf++; $cf ~ /^[1-9]/; cf++) {
                v1 = substr($cf,1,1)
                v2 = substr($cf,3,1)
                if (v2 == v1+1 || v1 == v && v2 == 1)   # e.g., 2,3 or 5,1
                        dbl[v1] = d
                else if (v1 == v2+1 || v2 == v && v1 == 1)      # e.g., 3,2 or 1,5
                        dbl[v2] = d
                else
                        error(sprintf("weird %s bond in\n\t%s", d, $0))
        }
}
function cvtdir(d) {    # maps "[pointing] somewhere" to degrees
        if ($cf == "pointing")
                cf++
        if ($cf ~ /^[+\-]?[0-9]+/)
                return reduce($(cf++))
        else if ($cf ~ /left|right|up|down|ne|nw|se|sw/)
                return reduce(dc[$(cf++)])
        else {
                cf++
                return d
        }
}
function reduce(d) {    # reduces d to 0 <= d < 360
        while (d >= 360)
                d -= 360
        while (d < 0)
                d += 360
        return d
}
function atom(s,    c, i, n, nsub, cloc, nsubc) { # convert CH3 to atom(...)
        if (s == "\"\"")
```

```
                return s
        n = length(s)
        nsub = nsubc = 0
        cloc = index(s, "C")
        if (cloc == 0)
                cloc = 1
        for (i = 1; i <= n; i++)
                if (substr(s, i, 1) !~ /[A-Z]/) {
                        nsub++
                        if (i < cloc)
                                nsubc++
                }
        gsub(/([0-9]+\.[0-9]+)|([0-9]+)/, "\\s-3\\d&\\u\\s+3", s)
        if (s ~ /([^0-9]\.)|(\.[^0-9])/)        # centered dot
                gsub(/\./, "\\v#-.3m#.\\v#.3m#", s)
        return sprintf("atom(\"%s\", %g, %g, %g, %g, %g, %g)",
                s, (n-nsub/2)*cwid, textht, (cloc-nsubc/2-0.5)*cwid, crh, crw, dav)
}
function shiftfields(n,          i) {    # move $n+1..$NF to $n..$NF-1, zap $NF
        for (i = n; i < NF; i++)
                $i = $(i+1)
        $NF = ""
        NF--
}
function fields(n1, n2,          i, s) {
        if (n1 > n2)
                return ""
        s = ""
        for (i = n1; i <= n2; i++) {
                if ($i ~ /^#/)
                        break;
                s = s $i " "
        }
        return s
}
function set(a, s,      i, n, q) {
        n = split(s, q)
        for (i = 1; i <= n; i += 2)
                a[q[i]] = q[i+1]
}
function error(s) {
        printf "chem: error on line %d: %s\n", lineno, s | "cat 1>&2"
}
```

APPENDIX 2

Listing of PIC Macros for CHEM

These macros are used in the output that CHEM generates for PIC

```
# macros for chem

pi = 3.141592654
deg = 57.29578
cr = 0.08               # radius of invis circle at ring vertices (see cr[vh])
crh = 0.16; crw = 0.12  # ht & wid of invis ellipse around atoms at ring vertices
dav = 0.03              # vertical shift up for atoms in atom macro
ringside = 0.3          # side of rings

# makering(numverts, radius, rotation, invis):  make symmetric ring
define makering {
        verts=$1; thisrad=$2; rot=$3
        if verts<3 || verts>8 then { illegal verts }
                                V0: ellipse invis ht crh wid crw at tt(-1)
                                V1: ellipse invis ht crh wid crw at tt(0)
                                V2: ellipse invis ht crh wid crw at tt(1)
                                V3: ellipse invis ht crh wid crw at tt(2)
                                V4: ellipse invis ht crh wid crw at tt(3)
        if verts >= 4 then {    V5: ellipse invis ht crh wid crw at tt(4) }
        if verts >= 5 then {    V6: ellipse invis ht crh wid crw at tt(5) }
        if verts >= 6 then {    V7: ellipse invis ht crh wid crw at tt(6) }
        if verts >= 7 then {    V8: ellipse invis ht crh wid crw at tt(7) }
        if verts >= 8 then {    V9: ellipse invis ht crh wid crw at tt(8) }
        C: 0,0
}

#ringlines(invis):  fill in all lines in this ring
define ringlines {
        for i=0 to verts-1 do { line $1 from tt(i) to tt(i+1) }
}

# tt(i) -- make vertex i of verts for this ring
define tt { (thisrad * sin(((($1)/verts*360+rot)/deg), thisrad * cos(((($1)/verts*360+rot)/deg)) }

# ringcirc(relative radius): make circle at center of ring
define ringcirc { circle radius $1*thisrad at C }
```

```
# atom(text, wid, ht, position of carbon)
define atom { [
        T: $1 wid $2 ht $3
        C: ellipse invis ht crh wid crw at T.w + ($4,dav)
        L: ellipse invis ht crh wid crw at T.w + (cwid/2,dav)
        R: ellipse invis ht crh wid crw at T.e + (-cwid/2,dav)
] }


# double(ring, v1, v2):  interior double bond in a ring
define double { line from .8<$1,$1.V$2> to .8<$1,$1.V$3> }

define triple { line from .7<$1,$1.V$2> to .7<$1,$1.V$3>;
                line from .85<$1,$1.V$2> to .85<$1,$1.V$3> }

define aromatic { circle rad $1. thisrad * $2 at $1 }

# bond(length, angle in degrees, whatever)
define bond {
        line $3 by ($1) * sin(($2)/deg), ($1) * cos(($2)/deg)
}

# fancy bonds:  r, theta, from/at
define doublebond {
        line $3 invis by ($1) * sin(($2)/deg), ($1) * cos(($2)/deg)
        V1: last line.start; V2: last line.end; dx = V2.x-V1.x; dy = V2.y-V1.y
        norm = sqrt(dx*dx + dy*dy)
        ny = dx * .02 / norm
        nx = -dy * .02 / norm
        line from V1 + (nx,ny) to V2 + (nx,ny)
        line from V1 - (nx,ny) to V2 - (nx,ny)
        move to V2
}
define dblbond { doublebond($1, $2, $3) }
define triplebond {
        line $3 invis by ($1) * sin(($2)/deg), ($1) * cos(($2)/deg)
        V1: last line.start; V2: last line.end; dx = V2.x-V1.x; dy = V2.y-V1.y
        norm = sqrt(dx*dx + dy*dy)
        ny = dx * .025 / norm
        nx = -dy * .025 / norm
        line from V1 + (nx,ny) to V2 + (nx,ny)
        line from V1 - (nx,ny) to V2 - (nx,ny)
        line from V1 to V2
        move to V2
}
define backbond {
        line $3 invis by ($1) * sin(($2)/deg), ($1) * cos(($2)/deg)
        V1: last line.start; V2: last line.end; dx = V2.x-V1.x; dy = V2.y-V1.y
        norm = sqrt(dx*dx + dy*dy)
        n = norm / .025
        ny = dx * .02 / norm
        nx = -dy * .02 / norm
        for i = 1 to n do {
                XZ: i/n <V1,V2>
                line from XZ + (nx,ny) to XZ - (nx,ny)
        }
        move to V2
}
define frontbond {
        line $3 invis by ($1) * sin(($2)/deg), ($1) * cos(($2)/deg)
        V1: last line.start; V2: last line.end; dx = V2.x-V1.x; dy = V2.y-V1.y
        ah = arrowht; aw = arrowwid; ahead = arrowhead
        arrowht = sqrt(dx*dx + dy*dy)
        arrowwid = 0.05
        arrowhead = 7
        line <- from V1 to V2
        arrowht = ah; arrowwid = aw; arrowhead = ahead
}

define label {
        "\s-31\s0" at .8<$1.C,$1.V1>
        "\s-32\s0" at .8<$1.C,$1.V2>
        "\s-33\s0" at .8<$1.C,$1.V3>
        if $1.verts >= 4 then { "\s-34\s0" at .8<$1.C,$1.V4> }
        if $1.verts >= 5 then { "\s-35\s0" at .8<$1.C,$1.V5> }
        if $1.verts >= 6 then { "\s-36\s0" at .8<$1.C,$1.V6> }
}

define ring8 {[
        makering(8, ringside * 8/6, $1);
        if $2 == 0 then { ringlines() } else { $3 }
]}

define ring7 {[
        makering(7, ringside * 7/6, $1);
        if $2 == 0 then { ringlines() } else { $3 }
]}

define ring6 {[
        makering(6, ringside, $1);
        if $2 == 0 then { ringlines() } else { $3 }
]}
```

```
define ring5 {[
        makering(5, ringside * 5/6, $1);
        if $2 == 0 then { ringlines() } else { $3 }
]}
define ring4 {[
        makering(4, ringside * 4/6, $1);
        if $2 == 0 then { ringlines() } else { $3 }
]}
define ring3 {[
        makering(3, ringside * 3/6, $1);
        if $2 == 0 then { ringlines() } else { $3 }
]}

define benzene {[
        makering(6, ringside, $1);
        if $2 == 0 then { ringlines() } else { $3 }
        ringcirc(.5)
]}

define ring { ring6($1, $2, $3) }

define flatring {[
        makering(6, ringside, $1); V4: V5; V5: V6; verts = 5
        if $2 == 0 then { line from V1 to V2 to V3 to V4 to V5 to V1 } else { $3 }
]}
define flatring5 { flatring($1, $2, $3) }

# encoded substs, subst at V-1 or null, ...
        # so far handles only 4 to 6 properly
define hcycle {
        c1 = c2 = c3 = c4 = c5 = c6 = 0; dc = cr
        if $1 % 2 == 0 then { $2 at V1; c1 = dc }
        if $1 % 3 == 0 then { $3 at V2; c2 = dc }
        if $1 % 5 == 0 then { $4 at V3; c3 = dc }
        if $1 % 7 == 0 then { $5 at V4; c4 = dc }
        if $1 % 11 == 0 then { $6 at V5; c5 = dc }
        if $1 % 13 == 0 then { $7 at V6; c6 = dc }
        line from V1 to V2 chop c1 chop c2
        line from V2 to V3 chop c2 chop c3
        line from V3 to V4 chop c3 chop c4
        if verts == 4 then { line from V4 to V1 chop c4 chop c1 }
        if verts > 4  then { line from V4 to V5 chop c4 chop c5 }
        if verts == 5 then { line from V5 to V1 chop c5 chop c1 }
        if verts > 5  then { line from V5 to V6 chop c5 chop c6 }
        if verts >= 6 then { line from V6 to V1 chop c6 chop c1 }
```