graphs in a sequence of search operations, e.g., R/N reduced graphs, followed by R/NC/NZ reduced graphs, followed by reduced graphs with non-a-variant nodes. In addition, it is possible to vary the amount of information within nodes, while for the refined search all the parameters must be evaluated by EPVs. The effectiveness of the reduced graph as a screening method will be discussed in the next paper in the series.

ACKNOWLEDGMENT

We gratefully acknowledge financial support by International Documentation in Chemistry GmbH (IDC), and an early contribution to the redesign of the ECTR by Dr. G. U. Schwarz of BASF.

REFERENCES AND NOTES

(1) Schoch-Grubler, U. (Sub)structure Searches in Databases Containing Generic Chemical Structure Representations. *Online Rev.* **1990**, *14*, 95–108.
(2) Downs, G. M.; Gillet, V. J.; Holliday, J. D.; Lynch, M. F. Computer Storage and Retrieval of Generic Chemical Structures in Patents. 10. The Generation and Logical Bubble-Up of Ring Screens for Structurally Explicit Generics. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 215–224.
(3) Gillet, V. J.; Downs, G. M.; Ling, A. I.; Lynch, M. F.; Venkataram, P.; Wood, J. V.; Dethlefsen, W. Computer Storage and Retrieval of Generic Chemical Structures in Patents. 8. Reduced Chemical Graphs and Their Applications in Generic Chemical Structure Retrieval. *J. Chem. Inf. Comput. Sci.* **1987**, *27*, 126–137.
(4) Fisanick, W. The Chemical Abstracts Service Generic Chemical (Markush) Structure Storage and Retrieval Capability. 1. Basic Concepts. *J. Chem. Inf. Comput. Sci.* **1990**, *30*, 145–155.
(5) Dethlefsen, W.; Lynch, M. F.; Gillet, V. J.; Downs, G. M.; Holliday, J. D.; Barnard, J. M. Computer Storage and Retrieval of Generic Chemical Structures in Patents. 12. Principles of Search Operations Involving Parameter Lists: Matching-Relations, User-Defined Match Levels, and Transition from the Reduced Graph Search to the Refined Search. *J. Chem. Inf. Comput. Sci.* **1991**, *31*, 253–260.
(6) Barnard, J. M.; Lynch, M. F.; Welford, S. M. Computer Storage and Retrieval of Generic Chemical Structures in Patents. 4. An Extended Connection Table Representation for Generic Structures. *J. Chem. Inf. Comput. Sci.* **1982**, *22*, 160–164.
(7) Welford, S. M.; Barnard, J. M.; Lynch, M. F. Computer Storage and Retrieval of Generic Chemical Structures in Patents. 5. Algorithmic Generation of Fragment Descriptors for Generic Structure Screening. *J. Chem. Inf. Comput. Sci.* **1984**, *24*, 57–66.
(8) Nilsson, N. *Principles of Artificial Intelligence.* Springer-Verlag: Berlin, 1982.
(9) Dethlefsen, W.; Lynch, M. F.; Gillet, V. J.; Downs, G. M.; Holliday, J. D.; Barnard, J. M. Computer Storage and Retrieval of Generic Chemical Structures in Patents. 11. Theoretical Aspects of the Use of Structure Languages in a Retrieval System. *J. Chem. Inf. Comput. Sci.* **1991**, *31*, 233–253.
(10) Downs, G. M.; Gillet, V. J.; Holliday, J. D.; Lynch, M. F. Computer Storage and Retrieval of Generic Chemical Structures in Patents. 9. An Algorithm To Find the Extended Set of Smallest Rings (ESSR) in Structurally Explicit Generics. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 207–214.

A LISP Program for the Generation of IUPAC Names from Chemical Structures

K. W. RAYMOND
Department of Chemistry and Biochemistry, Eastern Washington University, Cheney, Washington 99004

Received February 21, 1990

The nomenclature software described in this paper is designed to assist students in becoming proficient at using the nomenclature rules of organic chemistry. Features of the software include a graphics routine that converts structural formulas into drawings, the ability to determine correct IUPAC names for depicted molecules, and the capacity to identify errors in names that are input by the user. This paper describes the current stage of development of the LISP-based software and discusses its effectiveness in dealing with alkanes, alkenes, alkynes, and related halides.

INTRODUCTION

The work described in this paper results from the ongoing development of software[1] designed to help beginning students of organic chemistry learn to use the IUPAC rules.[2] The goal of the project is to give these students the opportunity to practice assigning IUPAC names to a great number of organic compounds and to have the computer supply them with the same type of assistance that an instructor might provide. The initial version of the software has been used in two upper division and two health sciences organic chemistry courses at Eastern Washington University. Although no formal study to judge the effectiveness of the software has been undertaken, student response has been exceptionally favorable.

Recent reports in this journal have documented studies related to the application of the IUPAC rules by computer analysis. Davidson[3] has described an IUPAC-based method for naming complex alkanes, and Cooke-Fox and co-workers[4–6] have reported on their work with development of a grammar system to interpret IUPAC names. Starting from structural formulas, the nomenclature software described in this paper draws molecules and determines their correct IUPAC names. Additionally, errors in IUPAC names supplied by the user are identified and assistance in making corrections is provided.

The computer programs were developed using the LISP language.[7–9] LISP is an acronym for LISt Processing, which refers to the form of both LISP programs and data. Lists are defined as sequences of objects (or other lists) embedded within a pair of parentheses. The great flexibility of the language makes LISP especially suitable for applications related to organic nomenclature. Built-in and programmer-defined functions allow lists to be manipulated in terms of any number of parameters including length, alphabetical position, and numerical value. Selection of an appropriate representation for a structural formula or name allows direct application of the IUPAC rules.

GRAPHICS OUTPUT

The first stage of program use involves the drawing of a molecule of choice. For acyclic compounds, the molecule to be drawn is specified by providing its structural formula. The formula may be provided by the user or taken from a supplied library of molecules. Once entered, the formula is parsed into a list in which each atom other than hydrogen is given an individual identity and is described in terms of its position relative to other atoms, the number of attached hydrogen atoms it carries, and the number of bonds required to complete its valency. Analysis of this list allows correct bond orders to be determined and permits any errors in the structural formula to be identified. Bond orders are not specified in the original formula, since they are determined as part of the graphics routine. Using built-in LISP graphics commands, the molecule is then drawn. Entry of the formula

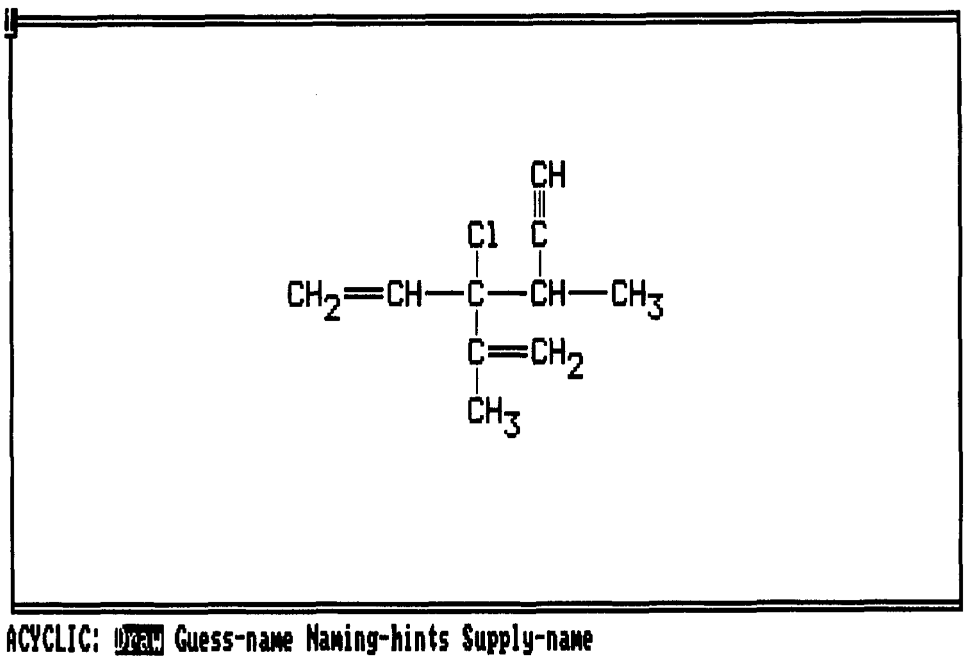


Figure 1. Graphics output for an acyclic molecule.

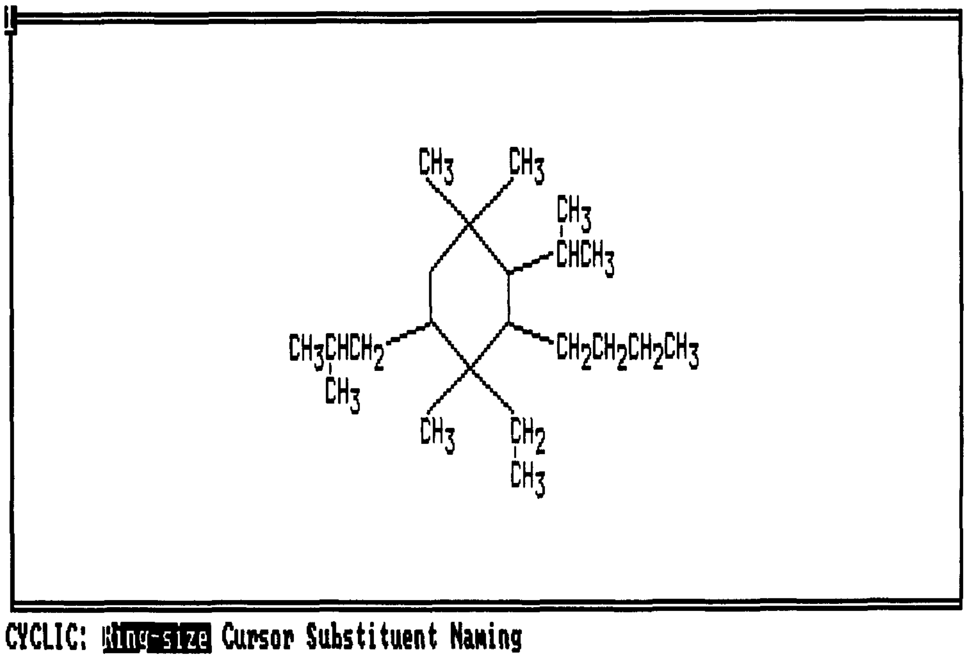


Figure 2. Graphics output for a cyclic molecule.

"$CH_2CHC(C(CH_3)CH_2)ClCH(CCH)CH_3$" produces Figure 1.

For cyclic molecules, the same basic approach is used. Once the desired ring size has been selected, movement of a cursor allows attachment of substituents. Substituents are specified by the entry of their structural formulas. Prior to drawing cyclic molecules, error trapping routines identify any structural problems that may exist, such as attachment of too many substituents to a given carbon atom and errors in the formulas of substituents. Figure 2 shows one example of the output for a cyclic molecule. The program is quite versatile in that it accepts substituents with alkyl, alkenyl, alkynyl, and halogen functionalities.

DERIVATION OF IUPAC NAMES

Once a molecule has been properly drawn, the nomenclature programs are accessed. The correct name for the molecule is determined, but supplied to the user only upon request. This protocol allows the user to make his/her own guess as to the name and receive appropriate feedback. The programming techniques used to derive names from the structures of acyclic alkanes, alkenes, alkynes, and halides are described below. Although not discussed, the programs treat cyclic systems in a similar fashion.

The molecule of choice for the following discussion has the formula $CH_3CH(CH_3)CHCH_2$. As part of the initial drawing routine, this formula is transformed into AB(C)D=E. It is this modified formula that is used for name derivation purposes. To determine the IUPAC name of an acyclic compound it is necessary to identify the longest continuous chain of carbon atoms in the molecule (the parent chain). To do so, the list that was created for purposes of drawing the molecule is used to create *bond-order*, a representation of all the covalent bonds within the molecule. For the formula AB(C)D=E, *bond-order* is assigned the value:

```
*bond-order* = ((A B 1) (B C 1) (B D 1) (D E 2))     (1)
```

Each sublist within *bond-order* specifies two interconnected atoms and the bond that holds them together. Interpretation of *bond-order* in eq 1 shows the molecule to be an alkene, since a double bond is present between atoms D and E.

The original list is again analyzed to determine the immediate neighbors of each atom:

$$\begin{aligned} A_{\text{*neighbors*}} &= (B) \\ B_{\text{*neighbors*}} &= (A\ C\ D) \\ C_{\text{*neighbors*}} &= (B) \\ D_{\text{*neighbors*}} &= (B\ E) \\ E_{\text{*neighbors*}} &= (D) \end{aligned} \qquad (2)$$

With a *neighbors* property[7,9] having been assigned to each atom, the terminal atoms of the molecule can be determined since the last atom in a chain has only one neighbor:

$$\text{*ends-of-molecule*} = (A\ C\ E) \qquad (3)$$

A best-first search technique[7,10] is then applied to identify all of the possible parent chains in the molecule. In sequence, each member of the *ends-of-molecule* list is used as a root node and each of the other members of the list as a goal of the search. Given, for example, that A and C are members of *ends-of-molecule*, that B is a neighbor of A, and that C is a neighbor of B, one possible parent chain can be designated (A B C). For the molecule described by eqs 2 and 3, the complete list of potential longest chains is

$$\begin{aligned} \text{*parents*} = (&(A\ B\ C) \\ &(A\ B\ D\ E) \\ &(C\ B\ A) \\ &(C\ B\ D\ E) \\ &(E\ D\ B\ A) \\ &(E\ D\ B\ C)) \end{aligned} \qquad (4)$$

The forward and reverse direction of each pathway is retained in *parents* for use in the later steps of the naming process.

In accordance with the IUPAC rules, the *parents* list for alkanes is sorted by length and all but the longest chains are discarded. If the molecule is an alkene or alkyne, *parents* is sorted so as to keep those pathways with the most multiple bonds, with further distinctions being made following IUPAC rule A-3.[2] In the current example, a revised *parents* list results:

$$\begin{aligned} \text{*parents*} = (&(A\ B\ D\ E) \\ &(C\ B\ D\ E) \\ &(E\ D\ B\ A) \\ &(E\ D\ B\ C)) \end{aligned} \qquad (5)$$

Reducing the *parents* list in eq 4 to only those chains containing a double bond was accomplished by comparing each sublist of *parents* to *bond-order*. Only those chains containing atoms D and E have a double bond. Since the four remaining chains are of the same length, all are included in the revised *parents* list (eq 5). All members of *parents* are candidates for the parent chain, and all are of the same length, so the prefix for the final parent chain, *but* in this case, may be assigned.

Further distinction between the remaining members of *parents* is accomplished by determining locants for the substituents and/or multiple bonds. To determine the substituents, each member of *parents* is examined separately, atom by atom. The list that describes each individual parent is compared to the *neighbors* property of each atom in the chain. Any member of *neighbors* that is not a member of the parent chain represents a point of attachment for a substituent. Consider the list (A B D E), the first member of *parents*. From the initial parsing of the structural formula, it was determined that $A_{\text{*neighbors*}}$ equals (B). Since B is a member of the pathway (A B D E), the atom A has no substituents. In a similar fashion, B, whose neighbors are A, C, and D, does have a substituent because C is a neighbor that is not contained in the parent. Once a branch point has been located, the structure of the substituent is determined by searching the branch to its end(s). Substituent names are assigned based on their formulas.

While testing each member of *parents* for attachments, a new property, that of *substituents*, is defined for each atom in the molecule. Each position in the *substituents* list corresponds to the identical position in *parents*. Atom B, for example, is assigned the property list:

$$B_{\text{*substituents*}} = ((C)\ (A)\ (C)\ (A)) \qquad (6)$$

The first item in $B_{\text{*substituents*}}$ corresponds to the first member of the *parents* list. Specifically, C is a substituent of B when B is in the chain A-B-D-E. Similarly, A is a substituent of B in the parent chain C-B-D-E, the second member of *parents*. In this example, the value of *substituents* for A, C, D, and E is (nil nil nil nil), since these atoms carry substituents in none of the four chains.

After derivation of the *substituents* property for each atom, the *parents* expression is analyzed in terms of both the *bond-order* and *substituents* lists. The result is a new expression that contains the position of each member of *parents*, the total number and position of substituents attached to that parent, and the type and location of any multiple bonds. In this instance the new list, *arranger*, has the value:

$$\begin{aligned} \text{*arranger*} = (&(1\ (2)\ (2\ 3)) \\ &(2\ (2)\ (2\ 3)) \\ &(3\ (3)\ (2\ 1)) \\ &(4\ (3)\ (2\ 1))) \end{aligned} \qquad (7)$$

The first member of *arranger*, "(1 (2) (2 3))" is interpreted in the following manner: The element "1" denotes the first sublist in *parents*, (A B D E). The second item, "(2)", indicates that the chain ABDE has a substituent on the second atom. The third element, "(2 3)", shows that a double bond begins on the third atom of the chain.

Arranger is useful in that it facilitates the sorting of the *parents* list and the assignment of locants. If the molecule is an alkane, *arranger* is sorted based upon the position of substituents. If it is an alkene or alkyne, the sorting is based on the position of the multiple bonds (the third element of each item in *arranger*) and, if necessary, substituent position. In the example above, the molecule is an alkene, so the best choice should give the lowest number possible to the double bond. After sorting, *arranger* is reduced to

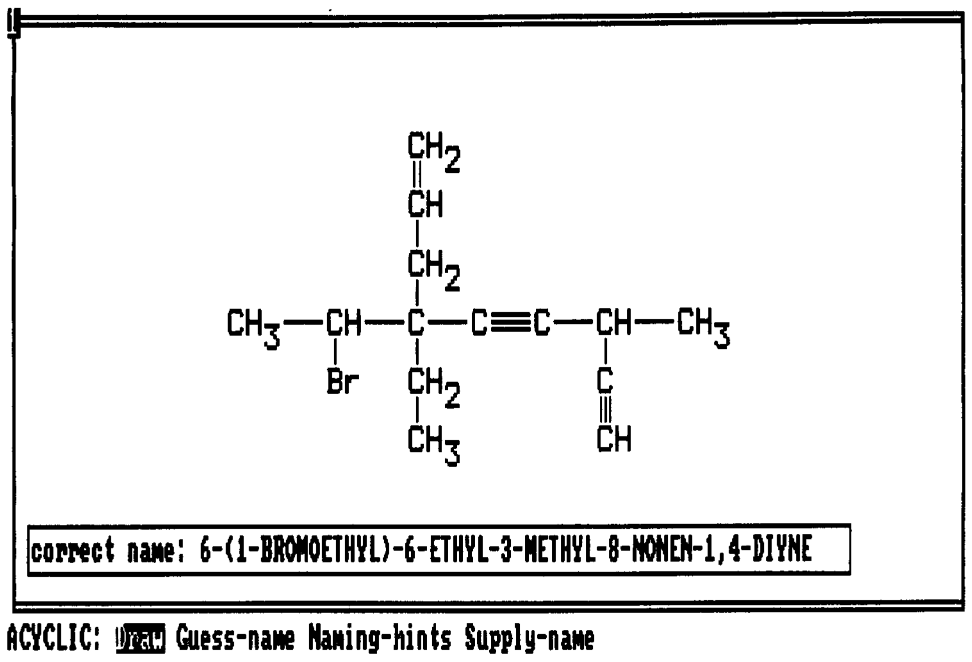
$$\begin{aligned} \text{*arranger*} = (&(3\ (3)\ (2\ 1)) \\ &(4\ (3)\ (2\ 1))) \end{aligned} \qquad (8)$$

In this example, there being no further distinctions to be made, the first sublist of *arranger* is selected as the parent. In this parent chain, the double bond is located on the first atom and the substituent on the third atom.

Once the individual components of the name have been determined, they must be correctly assembled. As specified by the IUPAC rules, substituents are sorted alphabetically and locants are arranged in numerical order. If present, the locants for multiple bonds are inserted into the proper location preceding the suffix or prefix. Any substituents or multiple bonds appearing more than once are indicated by using the proper multipliers. Finally, commas and hyphens are inserted in their respective locations. For the example above, "3-methyl-1-butene" is the name that would be derived. Figure 3 shows the program output for a more complex example.

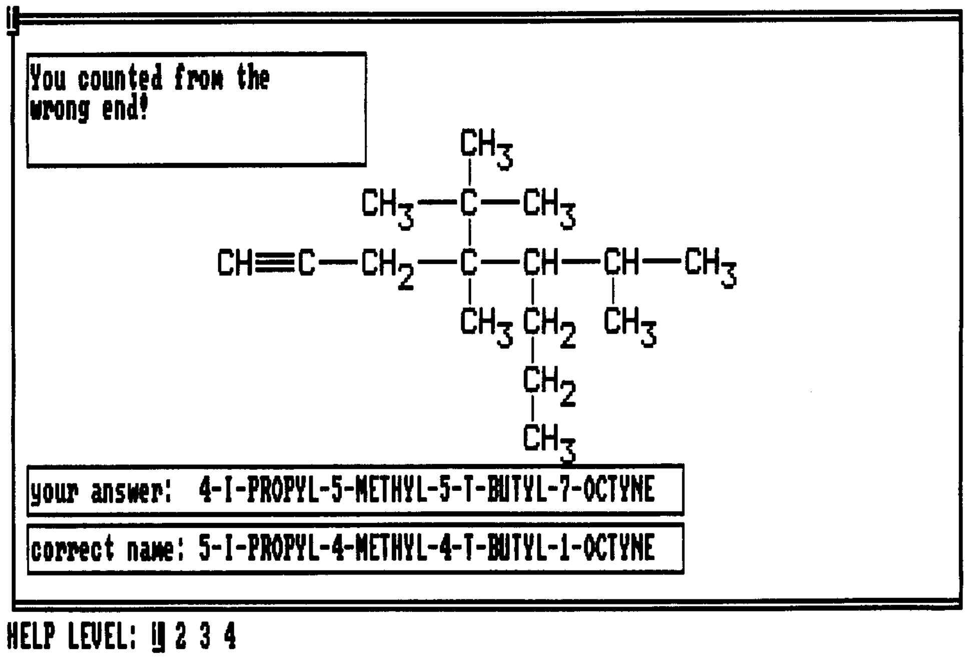
COMPARISON OF IUPAC NAMES

The ability to carry out name comparisons is a significant feature of the software. Having drawn a molecule, one option



```
correct name: 6-(1-BROMOETHYL)-6-ETHYL-3-METHYL-8-NONEN-1,4-DIYNE
```

Figure 3. Example of an IUPAC name determined from a structural formula.



```
You counted from the
wrong end!

your answer: 4-I-PROPYL-5-METHYL-5-T-BUTYL-7-OCTYNE
correct name: 5-I-PROPYL-4-METHYL-4-T-BUTYL-1-OCTYNE
```

Figure 4. Response to an incorrect name entered by a user.

available to the user is that of guessing its name. The name entered by the user is compared to the previously determined correct name, and errors are identified. To initiate comparison of the correct and user-input names, they are both converted into an easily interpretable format. Each name is analyzed in reverse, character by character. For each new character, the string that has been compiled to that point is examined for the presence of morphemes that denote suffixes, prefixes, substituents, multipliers, and locants. When matches are made, they are inserted into a *name* list in a location reserved for each specific component. The IUPAC name "5-ethyl-4,5-dimethyl-1,3-heptadiene" would be converted into the list shown in eq 9. The first element in this list, "e", is

```
*name* = (e
          ((en adi 1 3))
          hept
          ((ethyl nil 5) (methyl di 4 5)))        (9)
```

the terminal morpheme in the name of the molecule. This element is shared by several classes of organic compounds. The second element, the sublist "((en adi 1 3))", specifies the bond type, multiplier, and locant(s) for each multiple bond. The numerical prefix for the parent chain, "hept", occupies the third position of *name*. The last position is held by the substituents. They are listed by identity, multiplier, and locant(s). The absence of multipliers is indicated by the use of "nil".

If no correct morpheme is found, the unknown string of characters is placed in *name* in the position reserved for the missing item. If, for example, the name used to generate the list in eq 9 ended as "-1,3-heptadiWXYZe", instead of "-1,3-heptadiene", the second element in *name* would be "-((WXYZ adi 1 3))". This feature allows construction of informative error messages.

Naming errors are identified by using two different techniques. The first is to examine the name for compliance with the IUPAC rules. In the *name* list, substituents and suffixes should appear in the proper order, locants should appear in numerical order, and multiple locants should be indicated by using the correct multiplier. The parent, whose length is designated by the prefix, should be numbered from the correct end. Finally, the prefix, suffix, substituents, and multipliers should all have valid names.

The sublist of eq 9 that reads "((ethyl nil 5) (methyl di 4 5))" will be used to illustrate the approach. If listed correctly in the original name, ethyl will appear before methyl and the locants will appear in increasing numerical order. If the multiplier "nil" appears, only one number should follow it, if "di" appears, two numbers should follow it, and so on. If none of the above is true, specific errors messages such as "Use the multiplier di instead of tri" and "List ethyl before methyl" may be generated. In this manner, a wide range of errors in IUPAC usage may be presented to the user.

If the user-supplied name passes all of the tests for agreement with the IUPAC rules, it is then compared to the correct name. If, for example, the name "5,5-dimethyl-1,3-heptadiyne" was entered for the molecule described by eq 9, the following *name* list would result:

```
*name* = (e
          ((yn adi 1 3))
          hept
          ((methyl di 5 5)))          (10)
```

This name is valid in terms of the IUPAC rules, but is still incorrect. Error messages created by comparing eqs 9 and 10 could include: "The molecule is an alkene, not an alkyne", "The bond locants are correct, but the bond type is not", "There is a missing ethyl group", and "One of the methyl groups is numbered incorrectly".

Figure 4 shows the computer output resulting from an error in a user-input name. For illustrative purposes, both the correct and user-entered names have been shown. The error message provided is but one of four levels of assistance that is available for a given naming error.

CONCLUSION

The software described above has a number of features that should prove useful to those learning the IUPAC nomenclature rules. The graphics interface draws cyclic and acyclic molecules from user-provided formulas and identifies any errors. A library of structural formulas is also provided. User-entered names are compared to program-derived names. Error trapping routines identify mistakes and supply detailed information as to the nature of those errors.

In its current stage of development, the software is effective for alkanes, alkenes, alkynes, and their corresponding halides. Cyclic molecules ranging in size from three- to six-membered rings are allowed. For acyclic compounds, moderately branched chains of up to 10 carbon atoms in length are accepted. The software is currently being expanded to include additional organic functional groups.

The software runs on PC-compatible machines. Using a Zenith personal computer (12 MHz, 80286 processor), Figures 1, 3, and 4 were drawn in under 5 s. The correct names for the molecules in Figures 3 and 4 were determined in 2 and 4 s, respectively. The name comparison routines for Figure 4 were carried out in less than 1 s.

REFERENCES AND NOTES

(1) The program runs on IBM PC's and compatible machines under mu-LISP-87.
(2) International Union of Pure and Applied Chemistry. *Nomenclature of Organic Chemistry, Sections A, B, C, D, E, F, and H*; Pergamon Press: New York, 1979; pp 5–18.
(3) Davidson, S. An Improved IUPAC-Based Method for Identifying Alkanes. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 151–155.
(4) Cooke-Fox, D. I.; Kirby, G. H.; Rayner, J. D. Computer Translation of IUPAC Systematic Organic Chemical Nomenclature. 1. Introduction and Background to a Grammar-Based Approach. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 101–105.
(5) Cooke-Fox, D. I.; Kirby, G. H.; Rayner, J. D. Computer Translation of IUPAC Systematic Organic Chemical Nomenclature. 2. Development of a Formal Grammar. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 106–112.
(6) Cooke-Fox, D. I.; Kirby, G. H.; Rayner, J. D. Computer Translation of IUPAC Systematic Organic Chemical Nomenclature. 3. Syntax Analysis and Semantic Processing. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 112–118.
(7) Winston, P. H.; Horn, B. K. P. *LISP*, 2nd ed.; Addison-Wesley: Reading, MA, 1984; pp 3–14, 96, 169–178.
(8) Wilensky, R. *Common LISPcraft*; Norton: New York, 1986; pp 1–17.
(9) Brooks, R. A. *Programming in Common LISP*; Wiley and Sons: New York, 1985; pp 1–25.
(10) Winston, P. H. *Artificial Intelligence*; Addison-Wesley: Reading, MA, 1984; pp 87–132.