PATTY: A Programmable Atom Typer and Language for Automatic Classification of Atoms in Molecular Databases

Bruce L. Bush* and Robert P. Sheridan

Merck Research Laboratories, Building 50SW-100, P.O. Box 2000, 126 E. Lincoln Avenue, Rahway, New Jersey 07065

Received June 1, 1993®

PATTY (Programmable ATom TYper) is an algorithm for assigning "atom types" to a molecule based on its connection table of atoms and bonds. Its operation is controlled entirely by a rules file created by the user. Each rule contains a "pattern", a description in linear notation of a class of chemical substructures which may contain branches or rings. Rules are of two sorts: those which define properties to be used in subsequent rules and those which assign the final atom types. As an example, we present rules for classifying atoms into seven broad types based on their physical properties (*e.g.* cation, H-bond donor, hydrophobe). This classification has been applied to large databases of three-dimensional modes of druglike compounds. PATTY is very rapid and operates on a variety of hardware platforms.

INTRODUCTION

Atoms in molecules are frequently assigned "atom types" based on their bonded environments. An empirical energy calculation may distinguish among "amide carbon", "aromatic carbon", or perhaps even "5:6 bridgehead aromatic carbon". Even finer distinctions may be required when effective atomic charge is assigned. By contrast, broad physicochemical categories such as "aromatic", "hydrogen bond donor", and "hydrophobe" are useful for describing pharmacophoric patterns, drawing analogies between compounds, or making rough estimates of binding affinity.

Many computer programs require a computational chemist to assign atom types manually. For databases of thousands of chemical structures, manual assignment is impractical, and an automatic typing procedure is needed. An additional advantage of an automatic typer, aside from speed, is that it defines atom types unambiguously. Such reproducibility is important in calibrating and applying molecular force fields, as well as in documenting database queries.

We have developed PATTY (Programmable ATom TYper) to address these issues. The program reads a molecular connection table (atom species, bonds, and integral bond orders) and assigns atom types or charges based on a set of user-defined rules. The time required is small compared with most other molecular modeling operations, for instance, single-point energy calculations. PATTY is also a language processor: the atom-typing rules are defined in a free-format text language which expresses tests for atom properties and bonding topologies. The language is designed to be easily understood and modified.

The pioneering work of Corey and Wipke[1] and of Corey, Wipke, Cramer, and Howe[2] on computer-assisted design of organic syntheses (program LHASA) also provided a method of perceiving chemical substructures in terms of a user-defined rules file. PATTY addresses this task in a conceptually similar way, with a completely new FORTRAN implementation designed for portability and ease of use. The PATTY language bears some resemblance to the SMILES linear notation[3] for chemical structures. However, PATTY uses a sequence of rules rather than a single descriptive string. Each rule defines properties that can be used in subsequent rules. This technique allows PATTY to perceive very complex combinations of properties.

In this paper, we outline the elements of the PATTY language and the organization of a rules file. We then describe the operation of the program, with brief mention of the data structures into which the language is compiled. We show, as an example, a rules file designed to assign atoms to broad physicochemical categories, and describe the application of these rules to a large database of three-dimensional coordinates. We conclude by mentioning other possible applications or extensions of PATTY.

THE PATTY LANGUAGE

Rules File. The *RULES FILE* controls the operation of the program. It is a free-format file broken up as desired for legibility by white space or newlines. Comments may appear anywhere, within a line or as separate lines, set off by curly braces "{...Text...}".

RULES are applied to the molecule in the same order as in the file. Each rule ends in a semicolon (";") and has one of two forms:

property-assigning rule:

PATTERN ? PROPERTY-LIST ;

type-setting rule:

PATTERN > ATOM-TYPE-LIST ;

A *PATTERN* consists of *ATOM_TEST_NODES* connected by *BOND_TESTS*. The program "walks" the molecule, matching the tests to the molecule in all possible ways. Whenever the complete pattern matches a fragment, the program then assigns either *PROPERTIES* or *ATOM TYPES* to the atoms of that fragment, in the order matched.

Property-assigning rules have question marks ("?") after the pattern. For example,

C (= *) - * ? sp2_carbon;

assigns the property "sp2_carbon" to any carbon, doubly bonded to (at least) one atom and singly bonded to (at least) one other. The symbol "*" means "any atom". Each

* To whom correspondence should be addressed.

® Abstract published in *Advance ACS Abstracts*, September 1, 1993.

0095-2338/93/1633-0756$04.00/0

ATOM_TEST_NODE of a pattern (here, "C" or "*") must match a different atom of the molecule.

Type-setting rules have right-arrows (">") after the pattern:

C = O > c_carbonyl o_carbonyl;

This pattern looks for any carbon and oxygen joined by a double bond and assigns them the types named "c_carbonyl" and "o_carbonyl". Symbols for types and properties may be chosen with almost complete freedom. For example, many empirical energy programs use numerals to denote atom types:

C = O > 7 4;

As shown above, the right-hand side of the rule is a list of atom types. List entries are assigned in order to the atoms which match each atom test node of the pattern. If the list is shorter than the pattern, the rest of the matched atoms are simply left alone. An asterisk in the atom-type list also means "leave this atom alone", as in

C = O > * 4;

Properties and atom types play distinct roles. A property is an intermediate quantity, visible only to PATTY. Properties assigned in one rule may be (and should be) used in later rules. By contrast, an atom type will be written to the outside world but is not used in patterns. An atom type is not used as a property, even if its name happens to be the same as a property name. Thus no conflict arises when a rules file is adapted to external naming conventions for atom types.

Any atom can have many properties at once, up to several hundred. Property assignments are simply appended to those made by earlier property-setting rules. By contrast, an atom-type setting rule overwrites any atom types set by earlier rules.

Properties and Atom Tests. The symbol "C" for "carbon" in the rules above is in fact a PATTY property, just like the user-defined "sp2_carbon". When a molecule is first read, each atom is assigned a few *pre-assigned properties* so that PATTY has something to work on. Every atom receives an initial property denoted by the standard *element name E*: "C", "O", "Ca", "Sb", and so forth. It also receives one or two *valence properties*. One symbol, of form "*zE*", specifies the number (*z*) of explicit bonded neighbors (explicit coordination number): "3C" is a carbon with exactly three bonded neighbors. If hydrogens are represented in the molecule's connection table, a second predefined symbol of form "*mEHp*" indicates how many (*m*) of the bonded neighbors are non-hydrogens and how many (*p*) are hydrogens: "2NH1" is a secondary amine, "2NH0" the deprotonated form, "1CH3" a terminal methyl, etc. These valence properties are predefined only for convenience; they can also be defined through explicit rules, as shown in the sample application. The only restriction on choice of symbols in PATTY is that the user should avoid reusing these pre-assigned property names. Since the pre-assigned symbols start with a number of an upper-case letter, it is safest to employ lower-case strings for user-defined property names.

Properties may be combined very flexibly into compound symbols called ATOM_TEST_NODES, using the operators "," (OR), "&" (AND), and "!" (NOT). A node may consist of a single test, as shown in examples above: "*", "C", or "sp2_carbon". It may give *alternatives* separated by commas:

C, N, O

matches any of the three atomic species. Sets of alternatives may be combined in turn into strings by "&" (logical AND),

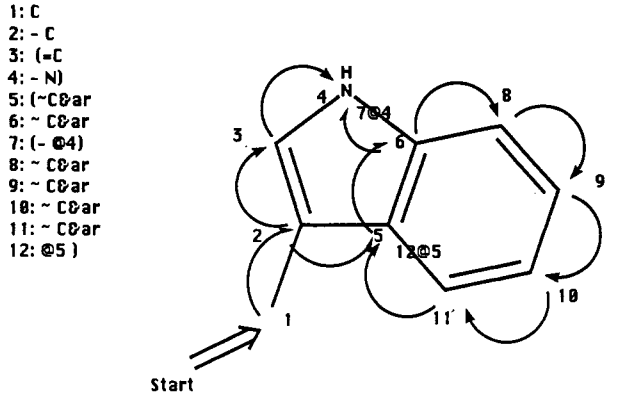

Figure 1. Schematic of how PATTY applies a pattern for the side chain of a tryptophan (Trp) residue. Numbers indicate the order of the atom tests, including the ring-closure nodes which refer back to previous nodes. The pattern is C–C(=C–N) (~C&ar~C&ar(–@4)~C&ar~C&ar~C&ar~C&ar~@5).

requiring the atom to match *each* of the sets. The node

sp2 & C

calls for a carbon (property "C") which also has property "sp2". The node

sp2 & C,N

calls for an atom that is "sp2" *and* is either "C" or "N". The node

sp2 & C,N & nohyds

imposes three necessary tests. (The order of "binding" the logical AND and OR, which is convenient for many applications, is opposite to the syntax of many query languages or computer languages.) Finally, a NOT symbol "!" may appear exactly once. The atom must match any tests before the "!" and must *not* match the tests after the "!". For example,

!H

matches any atom but hydrogen;

sp2 & N ! ring5, ring4

requires an atom that is "sp2" and is "N" (nitrogen) but is neither "ring5" nor "ring4".

The author of a rules file may prefer to break complex tests into several simpler rules. Here is an example:

sp2 & N ? n_sp2;

ring5 ? small_ring;

ring4 ? small_ring;

n_sp2 ! small_ring ? n_flat_ring;

Bonding Patterns. Atom test nodes are joined into a bonded pattern by *bond symbols* "–", "=", "#", or "~". These match single, double, triple bonds, or any bond, respectively. (A symbol "%" has been reserved for delocalized bonds, though these are not currently represented in our molecule file format.)

Branched bonding patterns require parentheses. A left parenthesis "(" before the first bond of the side chain matches a right parenthesis ")" after the last atom test node along the side chain. This need not be the end of a branch in the molecule. Parentheses are optional around the last branch from a node, for example, the last "(–*)" in this pattern for four-coordinate carbon:

C (–*) (–*) (–*) (–*)

A linear fragment may be written as branched, beginning at some nonterminal atom which may be most unusual or

```
FLIPS/FLOG rules file

ATOM TYPES FOR FLIPS/FLOG 9/2/92

{1=cation, 2=anion, 3=donor, 4=acceptor, 5=polar, 6=hydrophobic,
7=none}

{PROPERTIES}

{hybridization definitions}

*=*        ?  sp2 sp2 ;

*#*        ?  sp  sp  ;

*=*=*      ?  * sp *  ;

!sp2,sp    ?  sp3     ;

*-sp2,sp   ?  conj *  ;  {any atom single bonded to sp2 or sp}

sp,sp2     ?  res     ;  {"resonant" atoms}

N&sp3&conj ?  res     ;

O&sp3&conj ?  res     ;

S&sp3&conj ?  res     ;

{number of "neighbors" definitions}

*(~*)(~*)(~*)~*    ?  x4 * * * *   ;

*(~*)(~*)~*        ?  x3temp * * * ;

x3temp!x4          ?  x3           ;

*(~*)~*            ?  x2temp * *   ;

x2temp!x4,x3       ?  x2           ;

*~*                ?  x1temp *     ;

x1temp!x4,x3,x2    ?  x1           ;

*                  ?  x0temp       ;

x0temp!x4,x3,x2,x1 ?  x0           ;

{assignment of special atom properties}

res~res~res~res~res~res~@1 ? ar6 ar6 ar6 ar6 ar6 ar6; {arom. 6-
                                                       member ring}

res~res~res~res~res~@1 ? ar5 ar5 ar5 ar5 ar5        ; {arom. 5-
                                                       member ring}

ar5,ar6 ? ar ; {any flat ring}


N,O ?    neg ; {electronegative atom}


N-C=O,S ? namide * * ; {amide nitrogen}

N-P=O,S ? namide * * ;

N-S=O,S ? namide * * ;

N-P&x4-O,S&x1    ? namide * * ; {dative representation}

N-S&x3,x4-O,S&x1 ? namide * * ;


C(=O,S)(-O,S&x1) ? cx * * ; {carboxylate carbon}

{ATOM TYPES}

{default types}

*    > 7 ;  {default}

O    > 4 ;  {set O's to acceptors}

N    > 3 ;  {set N's to donors}

C,Si > 6 ;  {set C's etc. to hydrophobic}

S,Se > 6 ;  {set S's etc. to hydrophobic}

P,As > 6 ;  {set P's etc. to hydrophobic}

F,Cl,Br,I&x1  > 6 ; {halogens to hydrophobic}

O&x1-*              > 5 * ; {hydroxide (this may be overwritten
                             later)}

{special nitrogen properties}

N&x1&sp        > 4    ; {cyano nitrogens are acceptors}

N&x2&sp2       > 4    ; {-N= are acceptors}

N&x1&sp2       > 5    ; {HN=X are polar (overwritten later by
                          quanidinium etc.)}

*=N&x2=N&x1    > * 7 4 ; {azide}

{quaternary nitrogen, P, As and exceptions}

N,P,As&x4&sp3          > 1    ; {quaternary}

N,P,As&x4&sp3-O&x1     > 7  4 ; {turns off quarternary N-oxides}

N,P,As&x4&sp3-S&x1     > 7  6 ; {turns off quarternary N-sulfides}

N,P,As&x3,x4=O         > 7  4 ; {dative P-oxide}

N,P,As&x3,x4=S         > 7  6 ; {dative P-sulfides}

{sp3 amines and exceptions}

N&x1,x2,x3&sp3!conj > 1    ; {any nonconjugated sp3 N is a basic
                              amine}

N&x1,x2&sp3!conj-N&conj > 3 *   ; {except next to a conjugated N}

N&x3&sp3!conj-N&conj    > 7 *   ;

{conjugated N's}

N&x1&sp3&conj  > 3    ; {NH2-X= are donors}

N&x2&sp3&conj  > 3    ; {-NH-=  are donors}

N&x3&sp3&conj  > 7    ; {-N(-)-X= are planar}


{these are special cases where conjugated sp3 N's are basic}

C(=N&x1,x2!ar,conj)-N&sp3!ar,namide > 7 1 1 ;     {amidine}

{guanidinium}

N&sp3!ar,namide-C(=N&x1,x2!ar,conj)-N&sp3!ar,namide > 1  7  1  1;


{trisubstituted sp2 amines are cations with exceptions}

N,P,As&x3&sp2          > 1   ; {trisubstituted aromatic amine}

N,P,As&x3&sp2-O&x1     > 7 4 ; {turns off tertiary in nitro and
                                N-oxides}

N,P,As&x3&sp2-S&x1     > 7 6 ; {turns off tertiary in N-sulfides}

N&x3&sp(=O)=O          > 7 4 4  ; {turns off tertiary N in
                                   alternative representation in
                                   dative nitro}

{disubstituted sp amines and exceptions}

N&x2&sp       > 1    ;  {diazonium}

N&x2&sp-O&x1  > 7  4 ;  {nitrile oxide}

N&x2&sp#C&x1  > 7  4 ;  {isonitrile}
```

Figure 2a.

```
{trisubstituted oxygens,sulfurs and exceptions}
O,S&x3&sp3          > 1  ;  {trisubstituted}
O,S&x2&sp2          > 1  ;  {trivalent}
S&x3,x4&sp3-O&x1    > 7 4 ; {turns off sulfoxides and sulfones}
S&x3,x4&sp3-S&x1    > 7 6 ;
S&x3,x4&sp2,sp=O    > 7 4 ; {turns off dative sulfoxides and
                             sulfones}
S&x3,x4&sp2,sp=S    > 7 6 ;

{miscellaneous anions}
S&x1-ar6                          > 2 *     ; {conjugated
                                              sulhydryls}
C&x3(=O,S)-O,S&x1                 > 7 2 2   ; {carboxylates and
                                              S equivalent}
P,As&x3,x4(~O,S&x1)~O,S&x1        > 7 2 2   ; {phosphate/arsenate
                                              monoanion and S
                                              equiv}
P,As&x4(~O,S&x1)(~O,S&x1)~O,S&x1 > 7 2 2 2 ; {phosphate/arsenate
                                              dianion and S equiv
S,Se&x3(~O,S&x1)~O,S&x1           > 7 2 2   ; {sulfite/selenite
                                              anion and S
                                              equivalent}
S,Se&x4(~O,S&x1)(~O,S&x1)~O,S&x1 > 7 2 2 2 ; {sulfate/selenite
                                              anion
                                               and S equivalent}
C&x3&sp2(~O&x1)~N&x2~O~*~*~@1  > 6 2 2 4 * * ; {anion as in
                                               muscimol}
C=N&x2-N&x2=N&x2-N&x2-@1 > 6 2 2 2 2 ; {tetrazole}
C=N&x2-N&x2-N&x2=N&x2-@1 > 6 2 2 2 2 ; {tetrazole tautomer}

{miscellaneous tautomers and borderline ionizations}
{diaminopyrimidine}
N&x2&sp2-C&x3&sp2(-N&x1)~N&x2&sp2~C&x3&sp2(-N&x1)~*~*~@1 > 5 * 3 5
* 3 * * ;
N&x2&sp3-C=N&x2-C=C-@1 > 5 * 5 * * ; {make N's in imidazole
                                      equivalent}

{stabilized sulfonamides and phosphonamides}
ar-S,P&x4(~O,S&x1)(~O,S&x1)~N&x1        > * 7 4 4 5 ;
*-S,P&x4(~O,S&x1)(~O,S&x1)-N&x2-sp2,sp  > * 7 4 4 5 * ;
ar-S,P&x4(~O,S&x1)(~O,S&x1)-N&x2-sp2,sp > * 7 2 2 2 * ;

{carbonyl-hydroxide tautomers}
O&x1-C!cx=sp2-C!cx=O              > 5 * * * 5     ;
O&x1-C&ar6-ar6~C&ar6=O&x1         > 5 * * * 5     ;
O&x1-C&ar6-ar6~ar6-ar6~C&ar6=O&x1 > 5 * * * * 5 ;

{special ionizations of adjacent keto/enols}
O&x1-C!cx=C&x2,x3-C!cx(=O)-O,N-*        > 2 7 6 7 4 * * ;
O=C!cx-C&x2,x3-C!cx(=O)-O,N-*           > 2 7 6 7 4 * * ;
O&x1-C&sp2~C&sp2-C(=O)-C&sp2~C&sp2-O&x1 > 2 7 * 7 2 * 7 2 ;
O&x1-C&sp2~C&sp2~C(-O&x1)~C&sp2-C=O     > 2 7 * 7 2 * 7 2 ;

{miscellaneous fixups for "buried" atoms"}
C&x3&sp2~neg   > 7 * ; {a flat carbon surrounded by > 1 neg atoms}
C,S&x2&sp~neg  > 7 * ; {an sp carbon or sulfur surrounded by > 1
                        neg atoms}
N&x2&sp        > 7   ; {an sp nitrogen surrounded by neighbors}
```

Figure 2. Rules for assigning non-hydrogen atoms in small molecules to one of seven physiochemical types. The first group of rules (marked by "?") assigns "properties" based on substructures, the second (">") assigns "atom types". The property-assigning rules are subdivided into three sections to establish hybridization, to establish the number of (non-hydrogen) neighbors, and to recognize special environments. An atom in a molecule may have any number of properties. However, an atom can have only one atom type, and an atom type set by one rule may be overwritten by a later rule.

characteristic. Pattern "S(=O)(=O)" matches the same fragments as "O=S=O", though in different atom order. Branches may be nested to essentially any depth:

C (–C(–H) (–H)–H) (–C(–C(–H) (–H)–H) (–H)–H)

is one pattern for a secondary butyl.

A pattern need not include all of the branches from a single atom of the molecule. For example, "C–*" would match any carbon with one, two, three, or more neighbors. To distinguish them, specific valence symbols are required, for instance "1CH3–*" for a methyl.

Rings are described by a special atom test node, a *ring closure*, which points to a previous atom test node. Such a node may not contain any other tests. (Ring closure is, of course, an exception to the rule that each node in the pattern must match a different atom in the molecule.) A ring-closure node is written either as "@*n*" or "<*r*" where *n* or *r* is a positive number. The first form, "@*n*", refers to the *n*th atom test node by its absolute position in the pattern:

C(–C = C – C = C – C = C – @2)

matches a benzyl group. The second form, "<*r*", points to the *r*th preceding node ("go back *r* nodes"):

C(–C = C – C = C – C = C – <6)

Each ring-closure node should be paired with an asterisk in the assignment list on the right-hand side, since any assignment is made to the real atom to which it points. Likewise, closure nodes themselves must be counted when later nodes are numbered.

Fused rings require two or more ring closure nodes. The methylindole side chain of tryptophan will be matched by

C – C(= C – N) (~ C&ar ~ C&ar (– @4)
~ C&ar ~ C&ar ~ C&ar ~ C&ar ~ @5)

as illustrated in Figure 1. Most ring patterns can be written in several inequivalent orders. It is usually best to begin with an unusual node or to follow canonical numbering for a specific ring system.

ORGANIZATION OF THE PROGRAM

Program Flow. We now describe the PATTY program schematically. The main program is essentially a loop over molecules:

```
convert rules file to numbers and SYMBOL table;
for each molecule
    compile CONNECTIONS
        (for each atom, its bonds);
    set_predefined_PROPERTIES
        (for each atom: element, valences);
    apply_all_PATTERNs:
        for each rule
          compile symbol codes into arrays
              (instructions for FSM):
                PATTERN, JUMP, ACTION;
          apply_PATTERN: [FSM] apply rule
              (follow PATTERN, JUMP; take AC-
                TION)
          next rule;
    write the atom TYPES
next molecule.
```

Language Interpreter (Finite State Machine). The heart of the problem, embodied in routine *apply_PATTERN*, is a finite state machine (FSM).[4] Such a procedure, also called a finite state automaton, can interpret certain types of unambiguous languages like the PATTY language. The "program" for such a machine is stored as an array or table of instructions: in this case, the *PATTERN* of a rule (including associated arrays *JUMP* and *TESTS*) and the *ACTION* array. The machine operates upon external data: in this case, the *PROPERTIES* of the atoms (including the predefined properties) and the *CONNECTIONS*. At each step the FSM adopts a single defined "state" chosen from a finite set of possibilities (hence the name). The "state" of the FSM in PATTY consists of two "stacks", *PATH* and *BONDS_WALKED* (lists of atoms and bonds traversed so far in the molecule), and a pointer to the *PATTERN*.

This FSM executes a depth-first search over the molecule, comparing the bonds and atoms with tests described in PATTERN. When a test succeeds, the FSM adds to the PATH and moves forward in the PATTERN. When a test fails, the FSM first tries other bonds, then "pops" the stacks and moves back in the PATTERN. A successful atom is checked for duplication before being added to the PATH, so as to avoid self-intersecting walks. Branches in a rule are encoded in the JUMP array (actually a "column" of PATTERN). JUMP records the position of the left parenthesis associated with each right parenthesis in the rule. When a walk succeeds in matching this atom test node in PATTERN, the FSM takes the JUMP and proceeds to look at more bonds from the atom just before the left parenthesis.

Whenever the FSM machine successfully reaches the end of the PATTERN array, it takes the appropriate ACTION. For each atom in the PATH, it will set bits in PROPERTIES or place a value into TYPES. Having taken the action, the FSM then removes the *CURRENT_ATOM* from the end of PATH and continues the walk. For this reason, symmetrical patterns will be satisfied several times by the same fragment in the molecule, each time with the atoms in different order. For example, isopentane (tetramethylmethane) will match a pattern "C(–C) (–C) (–C) (–C)" 24 times. In practice, time spent making redundant matches has not been a significant problem.

Thus the PATH grows and shrinks, triggering the ACTION repeatedly. After the first entry in PATTERN has been applied to every atom of the molecule, apply_PATTERN has exhausted the rule, leaving behind a modified set of atom PROPERTIES. PATTY then expands the next rule into the working arrays and proceeds to apply it to the same molecule.

Fortran Implementation. PATTY is a stand-alone Fortran program which has been ported to a variety of hosts and operating systems: VAX VMS, Silicon Graphics Iris Unix (Irix), IBM mainframe MVS, and Cray UNICOS. All important functions are performed by routines which can be embedded in other programs. The data are passed explicitly between routines (not through COMMON). The rules file is read through an explicitly coded free-format input package. All operators have predefined numerical codes; symbols are assigned codes as they appear in the rules file. Strict coding practices, such as structured execution flow and explicit declaration of all variables and externals, make the program essentially a structured "C" program expressed in portable Fortran.

The current implementation of PATTY employs the following limits and capacities. All of them are Fortran PARAMETERS included from specification files.

Characters per user-defined symbol: 15
No. of symbols = max width of bit masks: 500
No. of symbols in all rules: 30000
No. of bits per integer: depends on computer
Atoms per molecule: 2000
Bonds from any atom: 8
No. of atom nodes in any one pattern: 250
Average masks per node: 5 (5 × 250 per pattern)

CONNECTION has one row for each atom, containing all bonds from that atom. Each bond is thus represented twice. PROPERTIES is stored as a bit array for each atom, declared as an array of integers and manipulated by explicit, portable Fortran functions. The FSM uses only as many bits as there are symbols mentioned in the rules file. Thus, no space or compute time is wasted on properties that will never be used.

To save space, the entire rules file is first checked for syntax and converted to a compact array of symbol numbers. Each rule in turn is then expanded "on the fly" into these working arrays, which need only be large enough to accommodate a single rule. The time consumed appears not to be an issue.

Each entry or "row" of PATTERN specifies a bond test and a set of atom tests called an ATOM_TEST_NODE. These atom nodes are of two types. A ring closure test is simply stored in PATTERN as the location of the associated real atom in PATH. No properties test is needed. Other atom tests are performed by a routine *TEST_ATOM* which takes care of all computer-specific details.

TEST_ATOM refers to the PATTERN table to get two pointers to a separate array, *TESTS*. In TESTS are found two sets of bit masks, the YES and NO masks, which correspond to properties named before and after the NOT symbol. (YES and NO masks are stored at the top and bottom of the TESTS array, growing toward the middle.) Each "AND" requires an additional mask. A mask, like the PROPERTIES of the actual atoms, is stored as a long array of 1 and 0 bits, one for each possible property. Routine TEST_ATOM tests the PROPERTIES of the candidate atom against each mask (using logical bitwise "AND"). The result must be nonzero for each YES mask and must be zero for each NO mask.

APPLICATION AND RESULTS

As a sample application we present a library of rules developed for the FLIPS/FLOG system (to be described elsewhere).[5] Program FLIPS is designed to select, from a three-dimensional database of small molecules, compounds which are similar to an ensemble of superimposed small

molecules. FLOG looks for compounds complementary to a receptor of known three-dimensional structure. To expand the notions of similarity and complementary beyond simple "lock and key" steric fit, without limiting the search to specific chemical functions, we decided to classify non-hydrogen atoms into seven types, depending on their state at near-neutral pH, as follows: (1) cations; (2) anions; (3) donors; (4) acceptors; (5) polar; (6) hydrophobe; (7) other. Except for the addition of type 7, these are similar to the six categories defined by Jiang and Kim[6] for atoms in proteins. The rules we developed to assign these seven atom types are listed in Figure 2. Note that they treat hydrogens implicitly, because the databases contain only non-hydrogen atoms.

Types 3 and 4 include uncharged hydrogen bond donors and acceptors, respectively. Type 5, "polar", includes atoms that are simultaneously donors and acceptors, *e.g.* hydroxyl oxygen, and atoms which can be donors or acceptors via tautomerization but not both simultaneously, *e.g.* nitrogens of an imidazole ring. This type also includes other atoms for which the pK_a is near neutrality, *e.g.* the N in aromatic sulfonamides. Nonpolar atoms in a nonpolar bonded environment are classified as type 6, "hydrophobe". Finally, polar atoms (*e.g.* N) which do not have a proton or lone pair, nonpolar atoms (*e.g.* C) which have a polar environment, or the central atom of sulfones, phosphates, etc., are assigned to type 7, "other".

We applied the rules to the MINDEX database, which contains 57 531 conformations of 7636 chemical structures. Coordinates for MINDEX were derived from the connection tables in the 11th edition of the *Merck Index*.[7] Assignment of atom types, with each conformation being typed independently, took 10 CPU min on our Cray Y/MP. Atom types of selected structures from that database are shown in Figure 3.

DISCUSSION

The PATTY language can describe any bonded topology, including branches, rings, and fusions. The depth-first strategy employed by PATTY to recognize such patterns in a molecule may be contrasted with the concentric patterns ("FRELs") employed by the DARC program system.[8]

PATTY patterns may be very specific and indeed can be used to define a specific molecule or functional group. They may be very general, with numerous tests and alternatives at each atom site. User-defined chemical concepts such as "aromatic", "sp2", or "halogen" may be included.

The one major restriction is that a pattern always describes one fixed bonding pattern. Thus PATTY cannot look for a "chain of any length" or a "ring of any size", classify bonds as "acyclic" or "cyclic", or match a ring substituent "anywhere on the ring". This restriction greatly simplifies the operation of the program and the development of a rules file. However, it means that alternative tautomeric structures or alternative dative representations (*e.g.* O=N=O *vs* O=N$^+$—O$^-$ in a nitro group) must be written as separate rules. Similarly, no completely general pattern can define aromaticity; each distinct Kekule structure requires its own pattern of explicit double and single bonds. In practice, we define a few special cases of aromaticity and then use the intermediate atom properties "aromatic" and "sp2" connected by "any bond".

The physiochemical atom-type classification requires rules to extend over rather large fragments (*e.g.* tetrazoles, aromatic sulfonamides). Such rules can be easily expressed in the PATTY language, and it is easy for the user to understand, modify, and "debug" them. A method that depends on "hard-coded" substructures, as was used, for instance, by Rusinko *et al.*[9] for their three-dimensional database, may be limited to very small substructures.

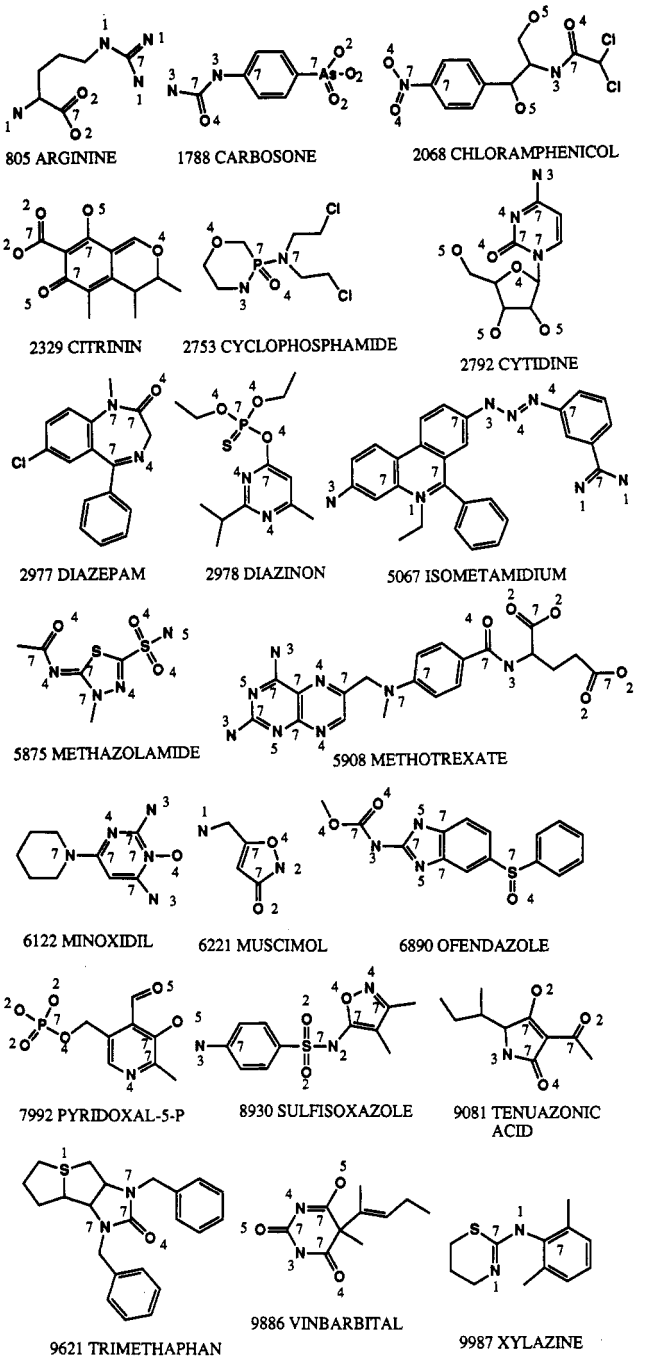

Figure 3. Selected chemical structures from the MINDEX database with their atom assignments. Atom types are as follows: 1 = cation, 2 = anion, 3 = H-bond donor, 4 = H-bond acceptor, 5 = polar, 6 = hydrophobic, 7 = none of the above. In this figure, unlabeled atoms are type 6.

The PATTY language was designed for generality and implemented with a view toward reliability, not speed. Nonetheless, the program has proved fast enough to be routinely used on databases with a half-million conformations.

As the databases have migrated between computer systems, PATTY has been "ported" from VAX/VMS to an IBM OS/VS/MVS mainframe to the Cray system currently in use. This required only small modifications to the input–output statements (Fortran OPEN), and perhaps changes to some of the parameters in the specifications files. In particular, the bit masks and their associated routines can be accommodated

to varying word lengths and storage orders. The decision to store only one rule at a time in expanded arrays has proved to be a good one, as the Cray architecture does not provide virtual memory or compact representations of logical (Boolean) data. The explicit coding of all operations in portable Fortran has also proved robust.

PATTY has also been incorporated as a subroutine into commands within the general-purpose interactive modeling system AMF.[10] Adapting the stand-alone program to this environment required only changes to the main program. File input and output are replaced by the passing of data to and from the modeling system's data structures through Fortran argument lists.

Other applications of the pattern language are easily envisioned. With slight modification, the stand-alone program has been used in our laboratories to attach canonical ring numbering to a set of piperazylpurines, as a preliminary to tabulating the substituents (R. Nachbar, unpublished work). In this application, PATTY becomes a programmable "atom NAMER" rather than "atom TYPER". With some modification, the program might generate an unambiguous description of each pendant group. To generate a standard IUPAC-compliant name for an entire molecule, as is done by program AUTONOM,[11] entails selecting a parent structure and building a "name tree". Such tasks lie beyond the scope of PATTY.

Instead of attaching information to the atoms of the input structure, PATTY might be modified to retrieve the atom list for each fragment or derive its geometry, much as program QUEST[12] analyzes geometries of crystal structures in the Cambridge Structural Database. For such applications, the fact that PATTY re-traverses symmetric patterns in every possible order could become a problem. An extension of the PATTY language might distinguish a single traversal of each pattern, by imposing additional tests on the internal storage order of chemically equivalent nodes.

CONCLUSION

PATTY, with its associated language for describing classes of chemical structures, has been applied to databases containing a wide variety of pharmaceutically interesting chemical structures. The language is simple and intuitive, and was employed successfully by a researcher not familiar with the internal operation of the program. The routine operates on a variety of hardware platforms. It is fast enough to serve as a preprocessor of a very large database or to set up individual calculations in the course of an interactive modeling study.

The program PATTY will be submitted to QCPE, the Quantum Chemistry Program Exchange, Indiana University, Bloomington, IN.

ACKNOWLEDGMENT

It is a pleasure to acknowledge stimulating discussions with Drs. Graham M. Smith and Joseph D. Andose regarding strategies for detecting fragments.

REFERENCES AND NOTES

(1) Corey, E.; Wipke, W. T. Computer-assisted design of complex organic syntheses. *Science* **1969**, *166*, 178–192.
(2) Corey, E. J.; Wipke, W. T.; Cramer, R. D., III; Howe, W. J. Techniques for perception by a computer of synthetically significant structural features in complex molecules. *J. Amer. Chem. Soc.* **1972**, *94*, 431–439.
(3) Weininger, D. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci.* **1988**, *28*, 31–36.
(4) Hopcroft, J. E.; Ullman, J. *Introduction to Automata Theory, Languages, and Computation*; Addison-Wesley: Reading, MA, 1979.
(5) Miller, M. D.; Kearsley, S. K.; Underwood, D. J.; Sheridan, R. P. FLOG: a system to find "quasi-flexible" ligands complementary to a receptor of known three-dimensional structure. *J. Comput.-Aided Molec. Des.*, submitted for publication.
(6) Jiang, F.; Kim, S.-H. Soft docking: matching of molecular surface cubes. *J. Mol. Biol.* **1991**, *219*, 79–102.
(7) The *Merck Index*, 11th ed.; Budavari, S., et al., Eds.; Merck & Co., Inc.: Rahway, NJ, 1989.
(8) DuBois, J.-E.; Carrier, G.; Panaye, A. DARC topological descriptors for pattern recognition in molecular database management systems and design. *J. Chem. Inf. Comput. Sci.* **1991**, *31*, 574–578.
(9) Rusinko, A., III; Sheridan, R. P.; Nilakantan, R.; Haraki, K. S.; Bauman, N.; Venkataraghavan, R. Using CONCORD to construct a large database of three-dimensional coordinates from connection tables. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 251–255.
(10) Andose, J. D.; Blevins, R. A.; Fluder, E. M.; Halgren, T. A.; Kearsley, S. K.; Sallamack, S.; Shpungin, J. AMF-The Advanced Modeling Facility, Version 1.12, January 1991; Merck Research Laboratories: Rahway, NJ.
(11) CSD Software System. 1993; Software Development Group, Crystallographic Data Centre, 12 Union Rd., CB2 1EZ Cambridge, England.
(12) Wisniewski, J. L. AUTONOM: System for computer translation of structural diagrams into IUPAC-compatible names. *J. Chem. Inf. Comput. Sci.* **1990**, *30*, 324–332.