SMILES Extensions for Pattern Matching and Molecular Transformations: Applications in Chemoinformatics

Richard G. A. Bone, Michael A. Firth,† and Richard A. Sykes*

Proteus Molecular Design Ltd., Beechfield House, Lyme Green Business Park, Macclesfield, Cheshire, SK11 0JL, United Kingdom

Received January 15, 1999

The selection and modification of atoms or functional groups underly many of the manipulations central to molecular modeling. It has become even more important to automate these tasks with the current prevalence of work with large databases of molecules. We have devised SUPER-SMILES, a conceptually simple set of extensions to the SMILES line notation, whose key features are addition and deletion facilities, macros, atom tagging, disjunctions, and constraints. This superset of SMILES enables us to carry out transformations on individual molecular structures or across members of a database with a pattern-matching protocol. The principal advantage of SUPER-SMILES is the ability to specify chemical reactions with a very simple augmentation of the SMILES line notation. For example, in conjunction with macros, it is possible to represent the displacement of tosylate with phenoxy by the expression "(Delete Tosyl) (Add Phenoxy)". SUPER-SMILES thus represents a unified approach to molecular structure specification and modification and can easily be applied to large datasets of molecules. This functionality has been implemented within the PROMETHEUS suite of CAMD programs. We demonstrate its use in carrying out such operations as atom-type assignment, protonation of molecules, valency checking, and hydrogen addition. Further applications such as library design and construction immediately suggest themselves.

INTRODUCTION

Line notations are a popular method for representing chemical formulas and their use is now widespread.[1] They are not limited to the representation of whole molecules nor to the description of single species.

SMILES[2,3] is now the most widely used chemical line notation, though others have been proposed.[4−12] SMILES has become popular because its rule base is very small and it is not encumbered by the need to generate a unique form.[2,3] In fact, the rules which govern derivation of a formula for an organic molecule are easily mastered by a synthetic chemist. SMILES is best regarded as a language which encodes a connection table representation.[4] Therefore the key to SMILES is that it has its roots in graph theoretical concepts. But its algorithmic definition derives from computer languages[2] and thus permits enhancements which can be expressed as syntactic expressions.

Whereas it is easy for a molecular modeling program to handle most other formats (e.g., Cartesian or connection table[13]) as input, the ability to cope with a line notation such as SMILES is less straightforward and requires more time to implement. Nevertheless, in an integrated CAMD environment, a SMILES interpreter is now recognized to be one of the basic tools needed. Although SMILES was initially made available in the Daylight software,[14] many other pieces of commercially available code have taken SMILES on board (see, for example, refs 15−17), and there is good reason to adopt one's own implementation if none of these is at hand.

Inevitably, in the various implementations worldwide, deviations from the original specification have arisen, as have extensions. It is possible to devise modifications while retaining the basic tenets of SMILES itself. It is a natural extension of pattern-matching ideas that the matched atoms or groups would need to be modified in some way. Such operations can be thought of as chemical reactions and require extensions to the matching capability which enable addition and deletion of atoms or bonds. Daylight has developed SMARTS[18] for pattern matching and has also introduced SMIRKS, a new notation for describing "generic reactions", derived from SMARTS and SMILES. Daylight's Reaction Toolkit[18] provides modelers with the necessary functions for reaction database searching as well as manipulating molecules through a SMIRKS syntax. We aim to show that, during the course of implementing a SMILES interpreter, it is possible to develop it into a language for structural manipulation and substructural pattern matching as well as unambiguous representation.

This paper describes SUPER-SMILES, a "superset" of SMILES, which we have developed and integrated into the PROMETHEUS CAMD technology.[19] SUPER-SMILES aims to offer powerful pattern matching facilities and a simple mechanism for carrying out molecular transformations. It augments the SMILES alphabet with keywords, each specifying some directive, choice, or constraint. Atoms which match are accessible via an associated data structure. The incorporation of addition and deletion mechanisms into the line notation enables molecular derivatization to take place in the same way as pattern matching, and with an easily understandable expression.

* Author to whom correspondence whould be addressed.
† Current address: International Research IS, Zeneca Pharmaceuticals, Mereside, Alderley Park, Macclesfield, Cheshire, SK10 4TG, U.K.

10.1021/ci990422w CCC: $18.00
Published on Web 08/28/1999

SUPER-SMILES can be used as an aid to creativity in ligand design situations, for example, as a means to define general classes of interaction sites, for assigning force field parameters, or for classifying rotatable bonds. The basic operations of structure building and substructure matching can be carried out with SUPER-SMILES, thus enabling molecular modelers and chemists to specify substructures and to carry out molecular transformations with ease. It is also easy to apply SUPER-SMILES expressions to databases of molecules so that filtering operations associated with library selection and profiling become easily expressible.

Our choice to implement a variant of SMILES is due to our confidence in the chemist's perspective—that it is intuitively straightforward to represent molecular operations by the SMILES line notation—and that the processing time in 2D is slight. (We defer the 2D−3D conversion step to later stages in our procedures and exploit other software kits for that process.) Our reliance on driving routines to effect molecular selection and manipulations and the use of rule sets stored in files are all facilitated by use of a line notation. It is much easier for a chemist or modeler to derive or edit the SMILES pattern for a molecular substructure than it is for them to augment the portions of computer code which operate on the molecular connection table and atom types. Indeed, most other chemical structure formats lack the concision required for deft manipulations.[20]

SOFTWARE ENVIRONMENT

Whereas the external representation of molecular data is usually specified by a file format or a symbolic language such as SMILES, software machinery is required to convert it to an internal representation. This means that the molecular modeling software must have data structures for describing molecular structures. In pattern-matching situations, a molecule stored in some external form, e.g., in a database or as a SMILES string, is read into PROMETHEUS where it is converted to a standard internal representation, fit for being matched by a SUPER-SMILES pattern. Our ability to define the appropriate data structures is made easier by the fact that we have our own programming language, Global, with which we can control the data.

GLOBAL Language. Global underpins the PROMETHEUS software environment. The current version in use is a fully-fledged programming language. It retains some features of the original implementation,[21] an early scripting language, akin to Tcl.[22] Global finds its main application in the construction of driver routines for molecular design procedures, as well as in the interactive manipulation of molecules. Most operations are performed with molecule data structures, calling upon a large library of chemistry-specific functions. Global has many of the features found in other more familiar languages such as C, FORTRAN, and Perl. These include the following: block structure; function declaration; typed variables; scoping conventions; loops; conditional statements; formatted I/O. General-purpose file facilities are present, and it is possible to call FORTRAN and C subroutines and functions from Global as well as access the UNIX system interface. Additionally, aspects of object orientation such as class structures and inheritance are fully supported. The use of SUPER-SMILES requires only a small subset of these features. Principally we use the block structure, object orientation (including dynamic object creation), and the character string concatenation facilities.

Data Structures. Named data structures in Global have built-in fields and may be passed to C and FORTRAN functions which can access those fields. There are five data structures used in operations with SUPER-SMILES: *MoleculeClass* contains mixed data, including fields for the number of atoms and bonds, arrays of atom labels, element types and charges, bond connection table, and geometric coordinates for each conformer; *MoleculeSelectionClass* is an object class which allows us to define a subset of atoms of a molecule, referenced by index; *SmilesFormulaClass* defines the compiled form of a SUPER-SMILES string; *SmilesResultClass* stores the result of matching a compiled SUPER-SMILES object against a molecular structure; and SmilesStateClass retains information about the checked and matched atoms.

It is instructive to analyze the structure of the SUPER-SMILES result object because it returns information about the latest match as well as offering us some control over the scope of the matching. An object of SmilesResultClass has accessible internal fields:

```
NAtoms: Int
Atoms: Int (Self.NAtoms)
StartAtom: Int
EndAtom: Int
NNewAtoms: Int
NewAtoms: Int (Self.NNewAtoms)
NewMol: MoleculeClass
NMatches: Int
Result: Object
State: SmilesStateClass
```

The structure of this class contains information about the matched atoms and constraints which may be initialized to restrict the scope of a match. Some fields in the object are altered during the matching process. The field NAtoms returns with the number of atoms in the current match. The atom numbers of those atoms are held in the dynamically allocated array, Atoms (its dimension is automatically resized to NAtoms). Similarly, the variable NNewAtoms contains the number of atoms which have been added during the match; this is only nonzero in the case of transformations which augment the molecule structure. The NewAtoms array records the atom numbers of the newly added atoms. In the case of a matching operation which transforms the molecule, a new molecule object is appended to the result structure in field NewMol; the atom numberings in NewAtoms are with respect to NewMol. The integer NMatches records the number of matches so far that have been applied to the current result object. The result object is also used to keep track of the current state of the match for subsequent uses of the same object against the same molecule, a process which is further described later. The information is held in the field State. Finally, the field Result is itself an object which holds the atom numbers of atoms which match with particular labeled atoms (or ranges of atoms) in the SUPER-

SMILES string and which become suitably marked in the pattern. This mechanism affords a way of identifying tagged atoms subsequent to a match.

Other fields, if preset, may determine the outcome of the match. For example, StartAtom and EndAtom would, if set, limit the matching utility to starting its searching at atoms in the range (StartAtom, EndAtom) of the molecule in question.

As with other object-oriented languages, within Global the dot operator allows us to refer to the fields of an object. If an object R is of structure SmilesResultClass, then the field NAtoms, for example, is accessed as R.NAtoms.

ELEMENTS OF THE SUPER-SMILES LINE NOTATION

In this section we describe the SUPER-SMILES syntax and capabilities and indicate some of the differences between it and SMILES and SMARTS. (Our aim is that the features are concise and powerful, yet give rise to a readable syntax.) In practice, for the purpose of testing out SUPER-SMILES expressions in our custom-built GUI, and for building libraries in 2D, a proprietary algorithm for generating 2D representations is used; this will be described in detail elsewhere.[23] When 3D structures are required, commercial software such as MSI's Converter[24] is used.

Differences from SMILES. SUPER-SMILES differs principally from SMILES in that there is no chemistry involved at the interpretation stage. Therefore SUPER-SMILES does not have inherent notions of unsatisfied valencies or aromaticity, nor does it distinguish handedness at chiral centers, though rectification of this is underway. Consequently, hydrogens, if present, must be specified exactly and explicitly.

In the main, the less restrictive way in which hydrogens are specified frees us from developing and updating a complex database of "standard" atom environments and from requiring the interpreter to be able to diagnose a particular SUPER-SMILES string as representing a bad or impossible molecule because of its imprecise or ambiguous valencies. Furthermore, it means that the parsing algorithm is quite bare and easy to maintain.

In contrast with SMILES, white space is allowed within a SUPER-SMILES string without affecting the overall interpretation. White space can be helpful in making an expression more readable.

Atoms. The definition of an atom employed by us is more flexible than that seen in SMILES itself. Without the need to distinguish between lower case letters as used for aromatic atoms and the lower case letters which are part of element symbols, we do not need to reserve an "organic" set of atoms nor enclose the remainder in square brackets. Consequently, atoms in SUPER-SMILES are represented by an uppercase letter followed by zero or more lowercase letters and underscores, and/or zero or more numbers preceded by an underscore. e.g., C, Cl, Bromine, C_sp_3.

In general, each uninterrupted sequence of digits must be preceded by an underscore. Macros and labels (see later) adopt the same naming conventions, a convenience which allows us to recognize certain reserved keywords as well as atoms, labels, and macros in an internally consistent way.

Charges are denoted in the same way as with SMILES, viz., by enclosing the atom symbol in square brackets and appending "+" or "−" followed by a number, e.g., [Mg+2] and its equivalent, [Mg++]. The number has the effect of multiplying the net sum of the preceding charges. So, an atom with zero charge can be represented in either of two ways: [N+−] or [N+0].

We have not concerned ourselves with isotopic specification.

Bonds. The different bond types are represented with characters identical to those used by SMILES itself with two exceptions. An aromatic bond, denoted by a colon (:), is used between bonded atoms of aromatic ring systems; we cannot exploit the way in which such a bond order is implicit in the lower case convention. (We always have the Kekulé representation of aromatic rings as an alternative.)

Our default bond order is "any" (i.e., implied by the omission of a bond symbol), rather than single. This facilitates pattern matching.

Topology. SUPER-SMILES patterns are read from left to right, just as with normal expressions. Branches are expressed in the same way as with SMILES. Therefore, propane is correctly and completely represented as C(-H)-(-H)(-H)-C(-H)(-H)-C(-H)(-H)(-H). The square-bracket notation for hydrogens is also present, so propane can also be represented more compactly by [C-H3]-[C-H2]-[C-H3].

The description of branching that we have implemented extends the square-bracket notation to atoms other than hydrogen, so 1,1,1-trifluoroethane could be expressed as [C-F3]-[C-H3].

Likewise, rings are represented as with SMILES, by denoting closure atom pairs. Cyclopropane is thus fully specified as C(-H)(-H)-1-C(-H)(-H)-C(-H)(-H)-1. Note that, with all bonds explicit, the ring closures must be preceded by matching bonds. The square-bracket alternative is [C-H2]-1-[C-H2]-[C-H2]-1.

PATTERN MATCHING

The major differences between SUPER-SMILES and SMILES arise within the pattern-matching syntax. In the following, we introduce a number of keywords which may be used within a SUPER-SMILES expression. The capitalization of all is regular and consistent with that for atoms: each comprises an uppercase letter with all remaining letters in lowercase.

In order to exemplify pattern matching, we use molecules **1** and **2** of Figure 1. The atoms which match example SUPER-SMILES patterns for each molecule are shown in Tables 1 and 2, respectively. Hydrogens are not shown in this depiction, and it is assumed in the following that they are not explicitly part of the molecule structure. (Therefore, our SUPER-SMILES example patterns will not contain hydrogens, though later examples of molecular transformations will.) Table 3 contains some examples of SUPER-SMILES patterns of more general use in matching organic molecules. The remainder of this section should be read in conjunction with Tables 1−3.

General Specifications. Wild cards are supported in SUPER-SMILES: we represent an atom of any type by ∗, but our symbol for a bond of any type (the default) is | (contrasted with ∼ in SMILES). Some matches involving nitrogen atoms are shown in Table 1.

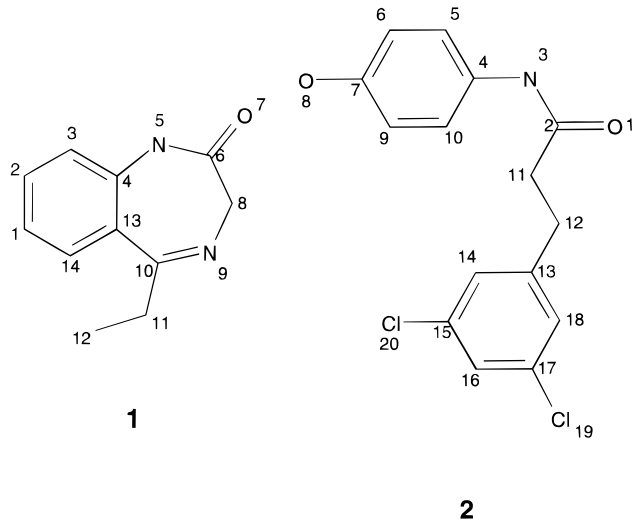


Figure 1. Example molecules **1** and **2** for discussion of SUPER-SMILES pattern matching operations.

The bracketed notation for branched atoms (e.g., the pattern [C-H3]) matches a given methyl group just once, whereas the pattern C(-H)(-H)(-H) matches the same set of atoms six times, once for each permutation of the hydrogen atoms.

In addition to branching, we permit the use of matching braces ({ and }) to denote a pattern or sequence of atoms that is within a chain, i.e., is connected at both its left and right sides. Braces may also be used wherever there is a potential ambiguity in a SUPER-SMILES expression as well as to clarify its meaning; their presence, if unnecessary, will be ignored by the interpreter. For example, the pattern C{N}C represents a three-atom chain in which the use of braces is superfluous (it is equivalent to CNC).

It is important to remember that an expression such as C1CC1 which is designed to match the skeleton of a three-membered carbon ring will also match 6 times: starting at each carbon in turn and in both senses around the ring.

Atom Attributes. The only atom-level properties that are specifically encoded for pattern-matching purposes are the formal charge (described above) and the "connectivity", denoted by @ followed by a number or an inequality. The connectivity is precisely the same as the graph theoretical quantity, "degree". It is the number of atoms bonded to the atom in question, regardless of bond type. (It should be remembered that bonds to hydrogens, when present, are counted when calculating the connectivity of an atom. This is important when describing atoms in particular hybridization states; see Table 3.)

Other atom-level attributes such as ring membership can be specified with the use of macros, as described later.

Choices and Restrictions: ?, Or, Xor and Once. An optional atom is prefixed by ? and enclosed in braces. This operator permits matching an atom or group whether or not it is present.

In order to represent a choice of atoms, we use logical expressions Or and Xor. Thus, a representation of all halogens is

F Or Cl Or Br Or I Or At

an expression which may be saved as a macro (see later).

Examples applied to molecule **1** are shown in Table 1. Note the different results which are obtained by reversing the ordering of the oxygen and nitrogen atoms in the Xor pattern.

The prefix Once has the effect of suppressing multiple matches over the subsequent set of atoms. Its most common use is in forcing single matches on ring systems. Less obvious instances include branching symmetry: a methyl group expressed as C(-H)(-H)(-H) would need to be prefixed by Once if a single match were required.

Constraints: ⟨ ⟩, Branches, Ignore, and Not. Often it is useful to specify atoms which describe a particular environment but are not included in the list of matched atoms. Such atoms are "constraints".

Table 1. Examples of SUPER-SMILES Patterns and the Atom(s) They Match on Molecule **1**

example	SUPER-SMILES	atoms matched
wild card atoms	N-∗	(5,6); (5,4); (9,8)
wild card bonds	C|N	(8,9); (10,9); (6,5); (4,5)
	C1CCCCC1N	(13,14,1,2,3,4,5); (3,2,1,14,13,4,5)
braces	C{N}C	(4,5,6); (6,5,4); (8,9,10); (10,9,8)
connectivity	C@3	4; 6; 10; 13
optional atom	C(=O){?N}C	(6,7,5,4); (6,7,8)
choices	C(=N Or =O)	(10,9); (6,7)
	C=(N Or O)	
	C={N Or O}	
	C (O Xor N)	(4,5); (6,7); (8,9); (10,9)
	C (N Xor O)	(4,5); (6,5); (8,9); (10,9)
once	{Once C1CCCCC1N}	(3,2,1,14,13,4,5)
	Once C-C(-C)=N	(11,10,13,9)
constraints	C⟨N⟩ C	(4,3); (4,13); (8,6); (6,8); (10,11); (10,13)
	C {Ignore N} C	(4,6); (6,4); (8,10); (10,8)
	C (Ignore |O)	6
	C {Ignore |O}	
	C⟨|O⟩	
	C ⟨C⟩ C-N	(3,4,5); (6,8,9); (8,6,5); (13,4,5)
	C ⟨1∗∗∗∗∗1⟩ N	(4,5)
	C ⟨∗1∗∗∗∗1⟩ N	(4,5); (10,9)
not	C(Not N)	1; 2; 3; 11; 12; 13; 14
reachability	C@3 C(=N) (...)	(13,10,9,11,12)
	CC=N (...)	(11,10,9,8,6,7,5,4,3,2,1,14,13); (13,10,9,8,6,7,5,4,3,2,1,14)
repeat	N {Repeat ∗} N	(5,6,8,9); (5,4,13,10,9); (5,4,3,2,1,14,13,10,9); each twice
	C {Repeat@3 ∗} O	(3,4,5,6,7); (10,9,8,6,7); (13,4,5,6,7)

Table 2. Examples of SUPER-SMILES Patterns and the Atom(s) They Match on Molecule **2**[a]

example	SUPER-SMILES	atoms matched
labeled atoms	N′P′ ⟨C=O⟩	3
macros	Ring_6	(4,5,6,7,9,10); (13,14,15,16,17,18)
	Ring_6 |N	(5,6,7,9,10,4,3)
	Aromatic_6	4; 5; 6; 7; 9; 10; 13; 14; 15; 16; 17; 18
	Hexring	(4,5,6,7,9,10); (13,14,15,16,17,18)
labeled macros	Once Hexring ′A′ Repeat ∗ Hexring ′B′	(5,6,7,9,10,4,3,2,11,12,13,14,15,16,17,18)
set	C(=O)N (Set ′Phenoxy′...)	(2,1,3,4,5,6,7,8,9,10)
revisit	N(Once Ring_6) {Revisit CCCCO}	(3,4,5,6,7,9,10,8) (twice)
goto	N (Once Hexring) Goto ′Hexring.d′ O	(3,4,5,6,7,9,10,8)
	C (Once Hexring) Goto ′c′ Cl Goto ′e′ Cl	(12,13,14,15,16,17,18,19,20))
start	{Once Hexring} {Once {Start Hexring}}	(4,5,6,7,9,10,13,14,15,16,17,18)
	Once {O Start O}	(1,8)
composite	{Once OC1CCCCC1} {Once Revisit Start NC1CCCCC1}	(8,7,6,5,4,9,10,3)

[a] Expansions of macros are found in the text.

Table 3. Examples of SUPER-SMILES Patterns To Describe Common situations

example	SUPER-SMILES	comments
sp^3 carbon	C@4	in a molecule with its full complement of hydrogens
sp^2 or sp^3 nitrogen	N@⟩1	nitrogen other than in a nitrile
spiropentane	C′A′CC′A′CC′A′	multiple ring closures achieved with a single label; in SMILES it would be written as C1CC12CC2
halide ions	{[F-] Or [Cl-] Or [Br-] Or [I-]}	could be used when stripping halide anions from a molecule
pyrrole macro	N′_1′-1-C′_2′C′_3′-C′_4′C′_5′-1	hydrogens have been omitted for clarity
ring formation	Pentyl N′A′	builds piperidine

We want to specify both an environment which must be true to permit a match and one which is false. The former is achieved in two ways: with angle brackets, ⟨ and ⟩, or with the keyword Ignore; and the latter with Not.

The angle brackets are a shorthand which may be used to specify atoms in a branch (i.e., any pattern which would be enclosed in parentheses). Therefore it is not meaningful to use them within a sequence of consecutively bonded atoms. Any SUPER-SMILES expression may be placed within angle brackets. A constraint may also include atoms already matched in the pattern. Therefore, the fact that atoms within a constraint can be revisited requires the use of caution, as exemplified by the expression C⟨C⟩C-N in Table 1. The Ignore keyword is more general and may be used both in branches and within braces.

Atoms within a constraint are only matched once, so it is not necessary to use Once within angle brackets. This is useful when specifying atoms which are constrained to be members of rings; see Table 1. The expression C ⟨1∗∗∗∗∗1⟩ N means a carbon in a six-membered ring (in which the five other atoms and all of the bonds are arbitrary) and which has a nitrogen substituent attached to it. This differs from the expression C ⟨∗1∗∗∗∗1⟩ N, which describes a carbon atom subject to the constraint that it is bonded directly to an atom in a six-membered ring and also bonded to a nitrogen.

The keyword Not is used to enforce something which is not true about an atom, i.e., to express negation. (Therefore it is not to be interpreted in its usual logical sense.) Negated atoms behave as if they were in angled brackets and so also are not included in the match. The expression C(Not N) means a carbon which is not bonded to a nitrogen; it does not represent a carbon bonded to an atom which is not a nitrogen.

Matching Recurring Patterns: Repeat. The keyword Repeat applied to a SUPER-SMILES expression, p, causes it to behave as follows:

p{?p{?p{?p{...}}}}

i.e., as many occurrences of p as can be found adjacent to each other are matched. This can be useful in finding chains at least one atom in length between fixed start and end points.

It is possible to restrict the length of the repeat with the @ operator. For example, the pattern in Table 1 locates all pairs of carbon and oxygens separated by three other atoms.

Reachability. We have introduced the shorthand notation, "...", to represent all atoms which are reachable from the latest atom in the pattern. If the last atom is in a branch, then the atom to which the branch is attached is the root of the search (unless the "..." is itself within the branch). The pattern reaches atoms accessible to the root atom which do not involve tracing paths over already matched atoms.

This syntax enables us to match down a long chain implicitly rather than by specifying every atom. By default, "..." on its own matches the whole molecule; it will match as many times as there are atoms. If the molecule is disconnected, then it matches each piece separately, for as many atoms as are found in each piece.

Labels. A label is a sequence of characters enclosed in single quotes. Unless a label is reused, labels do not have any effect during compilation of a SUPER-SMILES expression. During matching, each label becomes an integer field of the Result.Result object; the matched atoms are assigned to those fields. The fields themselves become identifiers. For example, the expression N′A′⟨C=O⟩ matches the amide nitrogen (atom 3) in molecule **2** and labels it "A"; the field Result.Result.A is set to 3. Applications of labels which mark particular atoms or sites of interest are described later.

Where a label is reused, its interpretation is that of ring closure. The first use of a label marks the current atom. Subsequent uses of the label are then references to that atom. The expression C′A′CCCCC′A′ represents a six-membered cycle. There is no restriction on the number of uses of a given label, allowing complex ring closures to be expressed, but also requiring the need for caution.

Naming Strings: Macros. A macro is a named SUPER-SMILES expression which has been pre-defined. When used in subsequent SMILES patterns, it must be passed to the compiler as a named field in a macro object. Macros may contain references to other macros (though recursion is not permitted), and full expansion takes place at compile time.

For example, we may set up a macro for a generalized six-membered ring:

Ring_6 = "*1*****1";

An expression for any six-membered ring bonded to a nitrogen is shown in Table 2. All the atoms specified by the macro are included in the match.

Macros may contain constraints: an atom in a six-membered aromatic ring (expressed in Kekulé form) may be defined as

Aromatic_6 = "* ⟨-1=*-*=*-*=*-1 Or
=1-*=*-*=*-*=1⟩";

Due to the use of a constraint, this pattern has the facility that it matches a single atom at a time.

Groups of elements may also be defined as macros, to express matches with choices of atoms, Table 3. It is usual to enclose such lists within braces because the environment in which they will be found is not always known ahead of time.

Atoms within a macro may be labeled, allowing them to be referred to by other parts of a SUPER-SMILES expression.

Hexring = "C′a′1C′b′C′c′C′d′C′e′C′f′1"

For example, Hexring can be used to label any six-membered carbon ring skeleton. Within a pattern which uses this macro, the atoms can be referred to as Hexring.a, Hexring.b, and so on. Additionally, the labeling feature can be used to encode an IUPAC-like numbering scheme for commonly found ring systems, e.g., pyrrole (Table 3).

Macros can themselves be labeled, so differentiating multiple instances:

Hexring ′A′ {Repeat *} Hexring ′B′

This pattern represents two six-membered carbon rings separated by any number of atoms. The labels ′A.a′, ′A.b′ etc., refer to the atoms in the first macro and ′B.a′, ′B.b′ etc., to atoms in the second. The scope of the labels ′a′, ′b′, etc., is strictly within each respective macro. This suppresses the possibility that similarly labeled atoms within separate macros could pair up with one another and form unexpected ring closures. After matching with this pattern, the Result.Result object has objects appended to it with the names "A" and "B"; each of these has a field corresponding to the labels "a", "b", etc., containing the matched atom.

Macros overcome the omission of chemical interpretation from the implementation of SUPER-SMILES. Although there are no built in macros to the SUPER-SMILES interpreter, a large library of pre-defined molecules and molecular fragments has been encoded and is accessible via the Global interface to PROMETHEUS. They may be used in matching operations just as with any other expression. Their use in molecule building is described later.

Other Features of Macros: Set, Revisit. A range of atoms in a portion of a pattern can be specified with Set followed by a label. For example, it may be required to label the phenoxy group of molecule **2**. The string C(=O)N (Set "Phenoxy" ...) will assign a field of the Result.Result object to Phenoxy and store an interval of the list Result.Atoms corresponding to those atoms which match the labeled pattern, in this case, atoms (4, 5, 6, 7, 8, 9, 10).

Atoms within a macro may be available to the rest of an expression with the use of the Revisit directive. This keyword permits the matching utility to revisit atoms which have already been matched and represents an alternative to the use of constraints, the main difference being that all visited atoms form part of the match. The whole SUPER-SMILES expression must represent a contiguous sequence of atoms, including any that are backtracked over during the revisiting process. The example in Table 2 shows how Revisit can be used in conjunction with a ring system macro if it is required to match more than one substituent. In general, Revisit finds more important application in matching disjoint parts of molecules, as described below.

Matching Disjoint Fragments: Goto and Start. Adjacency of atoms in a SUPER-SMILES string denotes a bonding relationship. It is often the case that a pattern needs to match sets of atoms which are not directly bonded to one another. The directive Goto followed by an atom label forces the interpreter to restart the match at an atom not necessarily connected to the current one in the expression. An alternative way of matching the aromatic ring and its attached nitrogen and oxygen in molecule **2** is shown in Table 2. The macro Hexring has applied atom labels to the ring carbons; the use of Once ensures that the match occurs only once. A Goto may appear any number of times in a SUPER-SMILES expression as shown in Table 2, where the other aromatic ring and both of its chlorine substituents are matched.

Whereas the Goto operator utilizes the label of a previously matched atom, it is possible to match noncontiguous portions of a molecule with Start, in which case all currently unvisited atoms are considered. (Therefore atoms which are bonded to the break point can still be matched with Start). The Start keyword precedes the pattern to which it applies. As a simple example, the expression {O Start O} simultaneously matches both the hydroxyl and carbonyl oxygens in molecule **2**. In Table 2 we also show how to simultaneously match both aromatic rings in molecule **2** (we have used Once to force single matches on each ring).

The keywords Revisit and Start are compatible with one another and can be combined to provide further flexibility in matching. Yet another way to match the aromatic ring in molecule **2** and both its para-bonded hetero atoms is shown in Table 2.

Further Examples. (a) Matching Peptide Sequences. Utilizing many of the above capabilities, it is possible to write compact SUPER-SMILES patterns which match useful environments such as peptide backbones. We can use a repeat construction because we do not necessarily know how long the backbone will be. Constraining the match to start at one end of the chain can be achieved with N ⟨Not C(O@1)CN⟩ CC⟨O@1⟩, whereas the repeating unit itself is NCC⟨O@1⟩.

The correct pattern is

N ⟨Not C(O@1)CN⟩ CC⟨O@1⟩
{Once ? Repeat NCC⟨O@1⟩}

We use Once to force the match to stop at the longest chain; we need the option infix, (?), to permit matching a single residue because Repeat requires at least one iteration.

Reachability finds particular application to proteins: the pattern NC(CO@1) (...) matches the backbone and side chain of an amino acid residue in a peptide sequence (though in the context of sulfur−sulfur cross-linking will match across separate chains).

We can also label protein side chains. For example, the pattern

NC(CO@1) (Set ′Side chain′ ...)

will assign a field of the Result.Result object to Sidechain and, on each match, store an interval of the list Result.Atoms corresponding to those atoms in the side chain of each residue.

(b) Representation of Peptides. One principal application of pre-defined macros is in protein structure analysis. In our library of macros we have included explicit forms for the 20 naturally occurring amino acids (Ala, Gly, etc.). This allows us to represent peptide sequences in a way similar to CHUCKLES;[25] e.g., a tripeptide could be written Ala Gly Leu. And, likewise, the amino acid macros may be mixed with regular atoms within a string for peptidic organic molecules. It is possible to handle cyclizations and branches within SUPER-SMILES with the use of labels, as described above. In a way similar to Siani et al.[25] we code specific forms of the amino acids for use in N-terminal, C-terminal, and branched situations. (The names of these are as for the amino acids themselves but suffixed with "n", "c", and "b" respectively; e.g., for cysteine, we have Cysn, Cysc, and Cysb.) The corresponding SUPER-SMILES strings are as follows (omitting hydrogens):

Cys: String = "-N-C(-C-S)-C(=O)";

Cysn: String = "-C(-C-S)-C(=O)";

Cysc: String = "-N-C(-C-S)";

Cysb: String = "-N-C(-C-S′B′)-C(=O)";

The macro Cysn can be used in a N-terminal position by putting a nitrogen or some derivative on its left-hand side.

For creating cyclic peptides, we simply add the requisite terminal atoms with the label of choice, for example:

N′Y′ Glyn Ala Leuc C′Y′=O

We use the version of glycine which is missing an N-terminus and the version of leucine which is missing a C-terminus and add each of the missing groups explicitly, labeling each "Y".

The strings for the branched residues contain an explicit label, "B". In order to utilize them in a sequence, it is necessary to label the macro itself so that the label encoded in the string can be accessed. If we were to extend the side chain of the cysteine with another carbon, we would do the following:

Gly Cysb ′X′ Leu.C′X.B′

We have labeled the cysteine X and identified its side chain atom as X.B; by using the disconnection (dot) operator, we can attach a carbon to it unambiguously.

Similarly, to bridge across a pair of side chains, we can use the following sort of construct:

N Glyn Cysb ′T′ Leu Cysb ′U′ Ala Goto
′T.B′ - ′U.B′

In this case we are adding a single bond bridge between the pair of cysteines, causing the side chain sulfurs to be bonded to one another.

Structural Transformations: Add and Delete. Changes to a molecule can only take place once an appropriate matched atom has been located. We can add atoms (and groups) to specified positions with the Add keyword. For example, in order to methylate the phenolic oxygen in molecule **2**, because no hydrogens are specified in this skeleton, we would use the pattern C-O(Add -C) which matches the oxygen unambiguously (because of its single bond to the existing carbon) and adds a carbon to it.

We are not limited to single substitutions. If we have a list of groups that we wish to experiment with in a given position, the use of Or will give rise to a sequence of distinct matches, for example:

C-O (Add {-C(=O)-C Or -C-C Or -C(-C)-C})

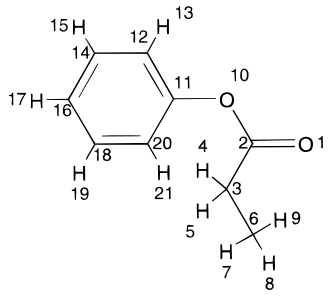
Similarly, we can delete atoms and/or bonds with a Delete keyword. Should we wish to remove the phenolic oxygen in molecule **2**, it is a simple matter of applying the following pattern: C (Delete -O); the carbon atom is returned as the successful matched atom. (Again, as there are no other carbons singly bonded to oxygens in this molecule, the pattern works unambiguously.)

In general, atom (re)numbering subsumes the internal representation. It is up to the modeler to work with the SUPER-SMILES patterns to ensure an accurate match with the target molecule.

VARIOUS TRANSFORMATIONS IN PRO_ANALOG

The foregoing pattern matching operations have shown how we can specify portions of molecules. The range of transformations that can now be described with SUPER-SMILES is large. Selected categories will be described in this section.

In an earlier publication[26] we described PRO_ANALOG, a suite of Global utilities for carrying out transformations on molecular structures with SUPER-SMILES. The utilities themselves take the form of functions which convert user-supplied arguments into the necessary SUPER-SMILES expression. In this section, we give examples of the SUPER-SMILES behind the different categories of transformation available. As described elsewhere, there is a different function within each category to cope with symmetry and presence or absence of hydrogens. In general, each function takes as its arguments the group to be deleted, the group to be added, and, where necessary, SUPER-SMILES expres-



3

Figure 2. Example molecule **3** for discussion of SUPER-SMILES transformations.

sions to specify the environment in the molecule where the change will take place.

In order to illustrate the types of transformation which are possible, we use molecule **3** (Figure 2) as our template. The SUPER-SMILES for various derivatizations of it are shown in Table 4 and the corresponding transformed molecules in Figure 3. The geometric position of the added atoms is handled with separate utilities.

Addition to and Deletion of Terminal Atoms. The ability to add and delete terminal atoms from a skeleton was described in the previous section. This set of functions has greatest use when applied to frameworks which have not been filled out with hydrogens. The basic form of the SUPER-SMILES expression is a matching pattern followed by Add or Delete, as already shown with reference to molecule **2**.

Substitution of Terminal Atoms. Substitutions are not restricted to single atoms: we can replace the methyl group in molecule **3** with trichloromethyl, as shown in Table 4.

If we wish to substitute one of the methyl hydrogens (7, 8, or 9) in molecule **3** by chlorine, we might try to achieve this with

C (-H) (-H) (Delete -H) (Add -Cl)

But this pattern may match the methyl group 6 times, giving rise to six new molecules because the transformed molecule is always returned as a new object (Result.NewMol). The use of Once will suppress this as shown in Table 4.

Replacement of Internal Atoms. In substituting terminal atoms, we utilized the branch notation. It is possible to replace multiply connected atoms by using the brace notation. In Table 4 we show how to convert molecule **3** into an amide, **3c**. Note that the full specification of the carbonyl group restricts the match to a single instance. Note also that we specify the bonds around the deleted and inserted groups.

Inserting Groups in Chains. It is only meaningful to insert a group into a bond, i.e., between two already bonded atoms. If we wish to separate the phenyl ring of molecule **3** from the ester by another methylene group (to obtain molecule **3d**), we can achieve that by deleting the bond between the ring and the oxygen and inserting the -[C-H2]- group, as shown in Table 4. We have specified which of the two possible C−O bonds by a constraint: the carbon must be a member of a six-membered ring.

Were we to be extending the chain on the other side of the carbonyl group, to form molecule **3e**, there are two ways of doing it, as shown in Table 4. Each expression uses a different way of forcing a single match, due to the symmetry involved.

Excising Groups from Chains. Similarly, it is only possible to remove groups from within a molecule that are bonded to exactly two other sites. Those two sites then become bonded to one another once the group has been removed. This operation is conceptually the reverse of insertions: we Delete a group and Add a bond.

Equivalent ways of removing the methylene group from molecule **3** and so producing the phenyl ester of ethanoic acid are shown in Table 4.

Ring Formation. In order to bridge disconnected (or noncontiguous) portions of a molecule, we can use the Start keyword. Care is necessary, however, especially when symmetry is present. As an example for discussion, we use molecule **2** and illustrate the situation of ring formation by insertion of a methylene group between the α carbon and the phenyl ring using ring carbon 10, ortho to the amide.

The following expression carries out ring formation exactly once.

N-C-C {Add -C-} {Start C-C(=O)}

We have used the Kekulé representation of the ring so that

Table 4. Examples of SUPER-SMILES Patterns To Effect Various Transformations on Molecule **3**

transformation	SUPER-SMILES	resulting molecule
terminal substitutions	[C-H2](Delete -[C-H3])(Add -[C-Cl3])	3a
	Once C(-H)(-H)(Delete -H)(Add -Cl)	3b
replacements	C {Delete -O-} {Add -N(-H)-} C(=O)	3c
insertions	C⟨1*****1⟩ {Delete -} {Add -[C-H2]-} O	3d
	Once C(=O) {Delete -} {Add -[C-H2]-} C(-H)(-H)	
	C(=O) {Delete -} {Add -[C-H2]-} [C-H2]	3e
excisions	{Once C(=O) {Delete -[C-H2]-} {Add -} C(-H)(-H)(-H)}	
	C(=O) {Delete -[C-H2]-} {Add -} [C-H3]	3f
ring formation	{Once O-C1CCCCC1(Delete -H) {Add -[C-H2]-} {Start C(Delete -H)-C(=O)}}	3 g
aromatic substitution	{Once Phenyl} {Goto 'c' ⟨*⟩ (Delete -H) (Add -[C-H3])}	3h
	{Once Phenyl} {Goto 'a' ⟨*⟩ (Delete -H) (Add -[C-H3])}	
	{Goto 'e' ⟨*⟩ (Delete -H) (Add -[C-H3])}	3i
	{Once Phenyl} {Goto 'a' ⟨*⟩ (Delete -H) (Add -[C-H3])} Or	
	{Goto 'e' ⟨*⟩ (Delete -H) (Add -[C-H3])}	3j (twice)
	{Once Phenyl} {Goto 'a' ⟨*⟩ (Delete -H) (Add -[C-H3])} Xor	
	{Goto 'e' ⟨*⟩ (Delete -H) (Add -[C-H3])}	3j
reduction	C(-C) (Delete =O) (Add -O-H) (Add -H)	3k
dehydrogenation	{Once C(-H)(Delete -H) {Delete -} {Add =} C(-H)(Delete -H)}	3l

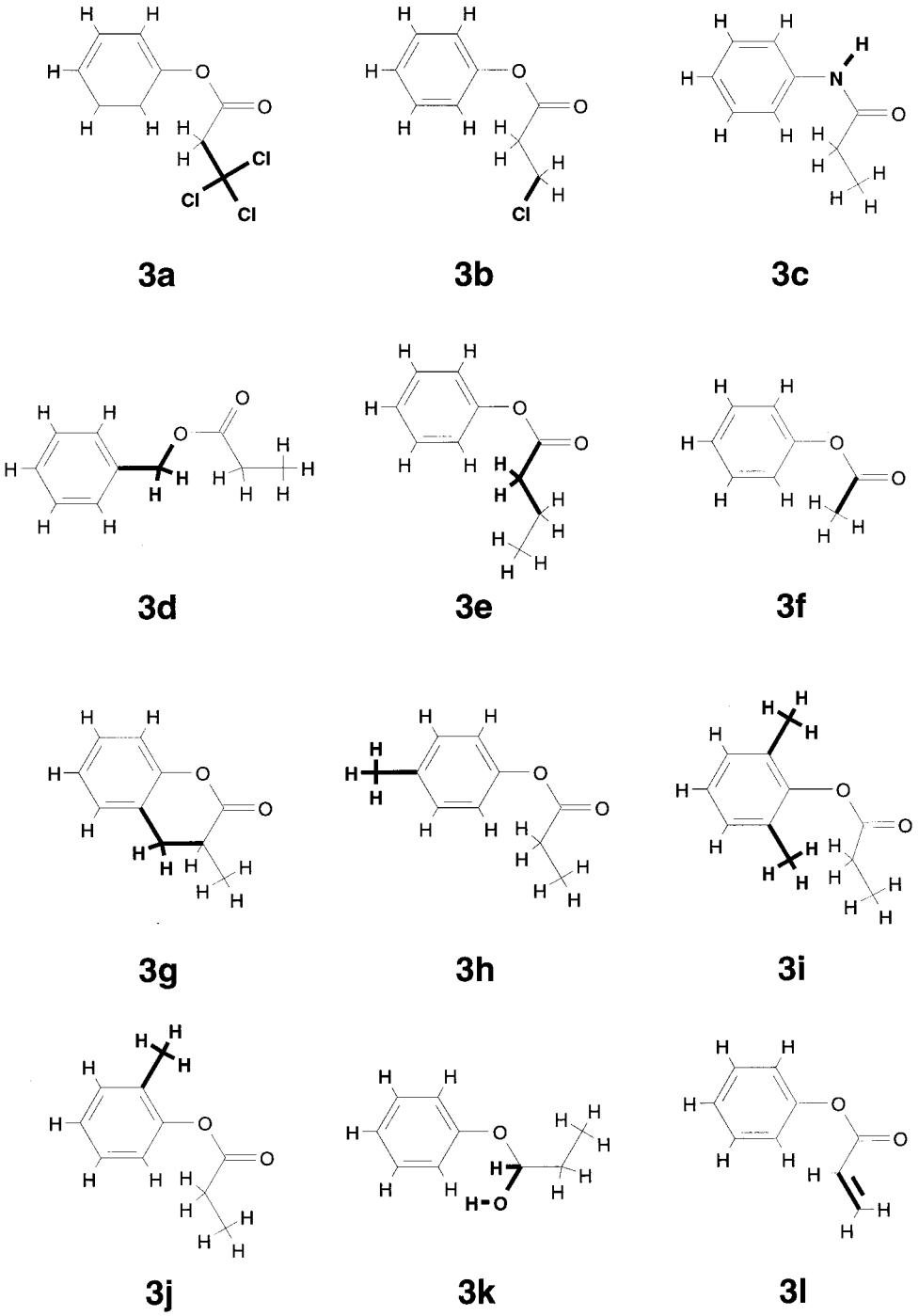


Figure 3. Molecules derived from molecule **3** by SUPER-SMILES transformations.

the portion N-C-C matches only one set of ring atoms. Similarly, the α carbon is unambiguously specified.

There are circumstances when it is necessary to specify the same atoms in each portion of the expression, in which case we can use the Revisit keyword. Although it is not necessary in molecule **2**, the expression

C(=O)-N-C-C{Add -C-} {Start Revisit C-C(=O)}

illustrates this to achieve the same transformation given above. It is the carbonyl group which is used twice.

Returning to molecule **3** we may carry out a similar ring formation between its α carbon and ring carbon 20, ortho to the ester oxygen. Because there are hydrogens present, not only is it necessary to suppress multiple matches, but the hydrogens must be deleted at each attachment point. The expression is shown in Table 4. Without the use of Once, the ring formation would be carried out 4 times because the phenyl ring carbon can be matched twice (corresponding to each direction around the ring) and because the α carbon has two attached hydrogens.

Aromatic Substitution. One of the types of transformation that is widespread in organic and medicinal chemistry is aromatic substitution. We can show how to combine several aspects of SUPER-SMILES to derive expressions for different substitution patterns on aromatic ring systems. The overall approach is to devise a macro which covers each of the possible representations of an aromatic ring and to label each of the ring atoms. We can then use the Goto keyword

to locate the actual substitution site. With these general principles, it is also possible to set up expressions for various substitution patterns and choices.

The basic macro for the phenyl ring is

```
Phenyl = "(C-1=C'a'-C'b'C'c'-C'd'=C'e'-1) Xor
          (C=1-C'a'=C'b'-C'c'C'd'-C'e'=1) Xor
          (C:1:C'a':C'b':C'c':C'd':C'e':1)";
```

We have used Xor to separate the different forms because we do not wish multiple matches to occur. (It might be possible for each of the two Kekulé forms to match.) This macro assumes that an attachment point of the aromatic ring to the rest of the molecule is specified at the leftmost carbon, which is therefore unlabeled.

In order to achieve the actual substitution, by group "X", say, in the ortho position, we could use a pattern of the form

```
Goto 'a' {⟨*⟩  (Delete -H)  (Add -X)  Xor  (Not *)
                                          (Add -X)}
```

This expression copes with the situations both where explicit hydrogens are present and where they are not. In the first case, the hydrogen is implicit in the constraint that ring atom "a" has a branch, (⟨*⟩). This expression cannot substitute group X onto a ring which already has a non-hydrogen substituent at position a. In the case of a hydrogen suppressed molecule (where atom a has no substituents at all), the group X is simply added.

The expression for substituting a methyl group para to the ester oxygen in molecule **3** is shown in Table 4. Each position can be specified unambiguously in this way.

Should we wish to substitute more than one position, there are several possible approaches. For example, we might wish to substitute both ortho positions simultaneously, each of the ortho positions separately (in cases where the two positions are symmetrically inequivalent), or just any one of the ortho positions. The first of these, simultaneous substitution, is achieved with separate uses of Goto in the same expression. Separate substitutions, i.e., returning separate molecules for separate matches, require use of Or between each Goto. And, similarly, in order to obtain substitutions at any one ortho position (but not both), we use Xor between each Goto.

PRO_ANALOG contains a library of Global functions which return the SUPER-SMILES for many common patterns of aromatic substitution.[26]

Generalized Molecular Functionalizations. Having seen the range of transformations that can be effected with SUPER-SMILES, we show how to describe certain types of well-known reaction. Several are shown in Table 4. The first example is the simple reduction of a carbonyl group to an alcohol. Transformations which change bond orders, such as dehydrogenations, are more complicated but can still be represented. Typically, the reverse of each of these reactions can be effected with the same portion of SUPER-SMILES but with each Add replaced by a Delete and vice versa.

Intramolecular reactions can also be described. For example, an intramolecular esterification could be written as

```
[C-H2]-O (Delete -H)  {Add -}
                      {Start C(=O)} {Delete O(...)}
```

We have explicitly deleted the hydroxyl hydrogen; the alcohol "leaving group" is compactly expressed using the (...) notation.

COMPUTATIONAL IMPLEMENTATION

We have implemented our SMILES matcher using standard compiler technology: the SUPER-SMILES patterns are treated as source code and compiled into a byte-code object language which is interpreted by a small stack-based interpreter during matching. This interpreter accumulates information in a state indexed by the instructions of the object code.

Essentially, our implementation uses fundamentals of language parsing based on a "Finite State Machine".[27] The first stage is to convert the SUPER-SMILES pattern into tokens using Lex,[28] which recognizes atom names, keywords, regular expressions, macros, numbers, and special characters. The tokens must be parsed (much as a piece of compilable code) in order to generate the object code. The grammar of SUPER-SMILES requires no more than one token of lookahead. Consequently, the parser can be implemented in yacc,[29] which is a LALR(1) ("Look Ahead Left Recursive") parser generator.

We have made both the lexer and the parser reentrant in order to compile macro definitions by a recursive call as they occur in the SUPER-SMILES pattern. This makes for a much cleaner implementation than the more usual preprocessing pass textually substituting macro bodies for names.

The object language into which the SUPER-SMILES expression is compiled is a simple one address byte-coded language, comprising 46 codes in total for matches of atoms and bonds, Boolean operations, and control flow and for accumulation of information in the match result structure.

The parser generates code linearly, with no requirement for back-patching. Jump addresses, for instructions which require them, are calculated at run time and stored in the instruction state.

A matching operation involves interpreting the object code with a molecule object and a starting state. The SUPER-SMILES object is interpreted by a code interpreter which maintains a program counter and has an internal state containing information about each successfully executed instruction. In particular, matching instructions retain a reference to the atom or bonds which match.

Success is indicated by completing the program; failure, by backtracking all the way before the first instruction.

(Backtracking is done by backing up a single instruction at a time, giving a depth-first algorithm. The state for each instruction contains the number of tries for the instruction so far. Instructions which match an atom may be tried only once before failure; instructions such as those to match a bond may be tried as many times as there are untried bonds on the current atom before failure. On failure the number of tries for the failing instruction is reset to zero.)

After a successful execution, the current state of the program is processed to produce the final match result, i.e., the atoms and bonds which match. The match result may be used for a subsequent execution of the object code: its starting state will be used to initialize the interpreter.

OPERATIONAL ASPECTS

SUPER-SMILES Utility Functions. There are four functions (directly callable from Global or C) for carrying out the principal manipulations of SUPER-SMILES expressions: encoding them into binary form; decoding molecular structures into SUPER-SMILES expressions; and two for matching of SUPER-SMILES expressions against molecular structures. These functions have been written in ANSI C.

(a) SmilesCompile. This function takes a String argument specifying a SUPER-SMILES pattern, and a Global Object containing macro definitions. It returns a compiled form (of type SmilesFormulaClass) for use in subsequent matching operations.

(b) SmilesMatch. This Boolean function takes three arguments: a previously compiled pattern (of type SmilesFormulaClass), a molecule-object (or Molecule Selection), and an object of type SmilesResultClass in which the results of successive matchings are stored. While matches can be found, the function returns True; if there is no match or no remaining matches, it returns False.

(c) SmilesUniqueMatch. This is just like SmilesMatch, but each successful match returns a different set of atoms.

(d) SmilesCanon. This generates a unique (canonicalized) SUPER-SMILES string from a molecule, based on Weininger's algorithm.[30] We have the option to give the function flags which specify whether or not a square-bracket notation is to be generated at branch points and also to specify a starting atom from which to make the string.

Typical Matching Protocol. The typical way in which matching with SUPER-SMILES is carried out is exemplified by the following portion (10 executable lines) of Global which creates a SUPER-SMILES object for an amide group and matches it against a molecule read in from a file test.mol, printing the result structure for each match. Comments follow the ## symbols.

```
SmilesString:  String;
                        ## Declare string variable
Mol:  MoleculeClass;
                        ## Declare molecule object
Macros:  Object;
                       ## Declare object to store macros
Pattern:  SmilesFormulaClass;
                        ## Declare Smiles Object
.Result:  SmilesResultClass;
                        ## Allocate SmilesResult Object
Mol = ReadMolecule ("test.mol");
SmilesString = "C(=O)-N(-H)";
Pattern = SmilesCompile (SmilesString, Macros);
While (SmilesMatch (Pattern, Mol, Result));
  Result.Print;
```

The first five executable lines of this extract merely declare the data types of the variables involved. The sixth line compiles SmilesString into an object that can be matched against the molecule. The remaining lines carry out the matching within a loop which exits when no further matches can be found. In practice, these lines of code would be embedded in some wrapper or other readily accessible utility.

The function SmilesUniqueMatch will return separate matches for each distinct set of atoms which can match the pattern. It can therefore be useful for single matches of ring systems but differs from the use of Once inside the pattern. For example, the pattern C1CCCCC1 will match each ring in molecule **2** once when using SmilesUniqueMatch; whereas the pattern {Once C1CCCCC1} used with the function SmilesMatch will match just the ring which is found first.

MISCELLANEOUS FACILITIES

Molecule Building. When using the SmilesMatch utility to effect a transformation, the new structure is contained in the NewMol field of the Result object. The implementation of SmilesMatch allows matching of patterns against a null (empty) molecule structure, in which case the result is the creation of a new molecule (or molecules) from the pattern. This turns out to be a powerful way of converting SUPER-SMILES strings into molecule data structures.

In the case that a single molecule is completely specified by the pattern, a single molecule is created in Result.NewMol. It is also possible to specify several molecules with a variable pattern, in which case embedding the match in a loop, as in

```
While (SmilesMatch (Pattern, Mol, Result));
  Print Result.NewMol;
```

will return a new molecule for each instance of the match. For example, the pattern

```
C:1(-H):C(-H):{N Or C(-H)}:C(-H):C(-H):C:1-H
```

when matched in that way against a null molecule, builds two separate aromatic rings, pyridine and benzene.

Creation of SUPER-SMILES Patterns with Global. In the interactive environment of PROMETHEUS the starting point of any matching operation is the generation of a SUPER-SMILES string. Global allows construction of patterns by simple assignment to string variables and also string concatenation with the "+" operator.

For example, we can build the string for butanoic acid, as follows:

```
String Chain_3 = "[C-H3]-[C-H2]-[C-H2]";
String Carboxy = "C(=O)-O-H";
String Butylacid = Chain_3 + "-" + Carboxy;
```

The result of this is that we have string variables for commonly used functional groups that can be reused. Of course, the molecule structure corresponding to butanoic acid can then be created by compiling Butylacid into a SUPER-SMILES pattern and matching it against a null molecule structure.

In practice, a store of commonly used functional groups and molecules has been created and, on initialization, permits those structures to be used as Global string variables as well as pre-defined macro names that can be used in SUPER-SMILES strings. There are too many groups to be listed here, but examples of the categories present include Alkyls; CycloAlkyls; CycloHeteroAlkyls; HeteroAromatics; Halo-

Table 5. Examples of SUPER-SMILES Expressions Stored in Rule Bases Used by GLOBAL Protocols

rule base	item name	SUPER-SMILES
ionization	Carboxyl	O′M′ (Delete -H) -C=O
	PrimaryAmine	N@3′P′(-H)(-H)(Add -H)←C@4⟩
	Amidine	N@2′P′ (-H Or C@4) (Add -H) ⟨=C-N@3⟩
	Sulfonamide	N′M′(Delete -H) ←C⟩ -[S=O2]-C
hydrogen addition	Acetyl	C@1 ⟨#*⟩ (Add -H)
	Deloc	C@2 ⟨(-*) =*⟩ (Add -H)
	Arom	C@2 ⟨(:*):*⟩ (Add -H)
	Vinyl	C@1 ⟨=*⟩ (Add -H) (Add -H)
	Terminal	C@1 ←*⟩ (Add -H) (Add -H) (Add -H)
	Secondary	C@2 ⟨(-*) -*⟩ (Add -H) (Add -H)
	Tertiary	C@3 ⟨(-*) (-*) -*⟩ (Add -H)
standardization	Match_nitro	*-N′A′(=O)=O′B′
	Change_nitro	O={Delete N(=O)} {Add =[N′A′+](-[O′B′-]-)}-C
	Match_alkali	Na′A′-O′B′
	Split_alkali	* (Delete -O-Na)(Add -[O′A′-].[Na′B′+])
valency checking	Alkyne	C@2 ⟨(-*) #*⟩
	Arom	C@3 ⟨(:*) (-*):*⟩
	Vinyl	C@3 ⟨(-*) (-*) =*⟩
	Tetrahedral	C@4 ⟨(-*) (-*) (-*) -*⟩
	Nitrile	N@1 ⟨#*⟩
	Narom	N@2 ⟨(:*):*⟩
	Imine	N@2 ⟨(-*) =*⟩
	Nitro	N@3 ⟨(-*) (=O) =O⟩
	Amine	N@3 ⟨(-*) (-*) -*⟩
	Ammonium	[N@4+] ⟨(-*) (-*) (-*) -*⟩
atom typing	Type_05	C ⟨ (-Hetero) (-H) (-H) -H⟩
	Type_47	H ←C_31 Or -C_20⟩
	Type_58	O@1 ⟨=*⟩
	Type_106	S@2 ⟨ (-H) -C⟩

Carbons; Amides; AminoAcids. Such macros allow us to express reactions in a very clear way. For example, a nucleophilic substitution reaction displacing bromide by ethoxide could be written

C (Delete -Br) (Add Ethoxy)

where Ethoxy has been predefined as -O-[C-H2]-[C-H3]. Most macros have been set up so that they bond on the left and enable a molecule expression to be built from left to right. A list of the pre-defined macros may be browsed through a graphical user interface. And, of course, this list may be extended as more examples are thought to be useful.

APPLICATIONS IN CHEMOINFORMATICS

With the functionality of SUPER-SMILES and the Global programming environment, we can construct utilities to achieve many of the basic manipulations which are frequently carried out during molecular modeling applications. Embedding these operations within a control structure facilitates automation and is especially useful in database filtering and virtual combinatorial library design situations. Most of these utilities take the form of generalized atom-matching protocols.

We have devised procedures for systematically matching SUPER-SMILES patterns against each atom in a molecule. The precise algorithm differs according to the context, but common to all is a rule base of executable statements that expresses the appropriate SUPER-SMILES transformations as named rules. Examples of the rules discussed here are shown in Table 5.

Ionization Utility. In the preparation of molecules for a ligand design experiment it is often necessary to ensure that their protonation states are correct; molecules which are obtained from database searching and/or which have come from a 3D generation package are not usually ionized for physiological pH. Accordingly, we have set up rule bases for acidic and basic groups, respectively. We can use a matching protocol to protonate or deprotonate the appropriate groups.

In this example, we make use of labelings in SUPER-SMILES, because it is necessary to update the charge of the protonated/deprotonated atoms in the molecule data structure, for which we need to know their indices. Examples of ionization rules in our rule base are shown in Table 5.

We have consistently labeled atoms which will attain a negative charge by ′M′ (minus) and those which will become positively charged, ′P′ (plus). They appear in the Result.Result object, as fields named P and M. It is only necessary to place the hydrogens in geometrically reasonable positions when they have been added; i.e., there is an atom labeled ′P′ in the result, something which is easily tested for.

Hydrogen Addition. It is frequently necessary to add hydrogens to the atoms of molecules (often from 2D databases) which are represented only as heavy atom skeletons. Our SUPER-SMILES method of carrying this out is an example of use of the MoleculeSelection data structure.

We first count and store the numbers of atoms of each element type in the molecule. Then, looping over the list of organic elements, for each one that is present in the molecule, we form a MoleculeSelection of the atoms of that type. We match the hydrogenation rules for that type with the molecule selection. For each element type, there are a number of different rules, depending on hybridization state. For example, those for carbon are shown in Table 5. Each rule has defined the atoms to which the carbon is already bonded to be within a constraint. The carbon atom itself is required to have a particular connectivity.

```
Patt1 = SmilesCompile(Rule1);
Patt2 = SmilesCompile(Rule2);

While (SmilesMatch(Patt1,Mol,Res))

    Aatom = Res.Result.A;
    Batom = Res.Result.B;
    XA = Mol.Xcoord(Aatom);   XB = Mol.Xcoord(Batom);
    YA = Mol.Ycoord(Aatom);   YB = Mol.Ycoord(Batom);
    ZA = Mol.Zcoord(Aatom);   ZB = Mol.Zcoord(Batom);

    ~.Res;          ## Reinitialise the Result structure

    SmilesMatch(Patt2,Mol,Res);
    Mol = Res.NewMol;

    Aatom = Res.Result.A;
    Batom = Res.Result.B;
    Mol.Xcoord(Aatom) = XA;   Mol.Xcoord(Batom) = XB;
    Mol.Ycoord(Aatom) = YA;   Mol.Ycoord(Batom) = YB;
    Mol.Zcoord(Aatom) = ZA;   Mol.Zcoord(Batom) = ZB;

    ~.Res;          ## Reinitialise the Result structure

EndWhile;
```

Figure 4. Schematic algorithm to standardize functional groups.

The use of a molecule selection speeds up the process: for example, if there is only one nitrogen atom in a given organic molecule, it is pointless wasting time matching the whole nitrogen rule base against every atom in the molecule. In practice, a counter keeps track of the number of matches, and once all the atoms of a particular type have been matched, the function exits.

Standardization of Database Structures/Preprocessing. It is common for a small but significant fraction of molecules stored in commercial databases to contain valency errors or functional groups in nonstandard representations. It is useful to apply a series of preprocessing rules to these databases in order to standardize them for future use. The overall procedure will be similar to that for the protonation rules, described above, but a two-stage procedure is necessary because the geometry of the groups will need to be preserved. There needs to be a rule to identify the group and the atoms which will be changed, as well as a rule to carry out the change.

For example, the rule pair for converting a nitro group containing a hypervalent nitrogen into a valence bond representation is shown in Table 5, as are the rules for separating a sodium ion from its counterion.

Correspondingly, the algorithm to achieve this is shown schematically in Figure 4. The variables Rule1 and Rule2 are a pair of SUPER-SMILES strings to carry out the standardization for a particular functional group; Patt1 and Patt2 are the respective patterns compiled from them; the variable Res is of type SmilesResultClass.

We identify the atoms to be changed in the first match and save their coordinates. After the second match, the atoms which replace them are assigned the coordinates of the original atoms. The result structure must be reinitialized after the first match because its internal state will preclude matching the same atoms again. It must also be reinitialized after the second match because atoms may have been added or deleted, thus changing the overall atom numbering in the structure.

Valency Checking. An ancillary role that atom matching can play is in valency checking. Just as databases often contain more than one representation of certain functional groups, so there may be badly drawn structures. It is often useful to identify errors such as pentavalent carbons. It is not reasonable to try to apply a rule which corrects these atoms, because there could be more than one solution to the error and human inspection of the context is likely to be more reliable. Consequently a valency checking utility should return the number of bad atoms or a MoleculeSelection of the bad atoms.

The general procedure is similar to that for hydrogen addition. Again, we only attempt to match on element types that are present. Examples of the rules in the rule base are shown in Table 5. We have used constraints on the atoms to which each target atom is bonded in order to allow the specific atom to be easily identified when reporting an error.

Atom-Typing. SUPER-SMILES can also be used to identify and assign atom types that are used in parametrized methods. In this respect, our application is rather similar to that of Bush and Sheridan, who devised PATTY,[31] an atom-typer which uses a rule base and parser language.

One published method of calculating log *P* and the molar refractivity[32] involves assigning parameters to 120 atom types. We have implemented a protocol very similar to the valency checker, except that the function returns the atom type of the atom as an integer between 1 and 120, corresponding to the definitions in ref 32. In practice, we encode some commonly used atom assignments into macro expressions, in accord with the scheme of the authors of ref 32, for example:

Halogen = "{F Or Cl Or Br Or I}";

Hetero = "{O Or N Or S Or P Or Se Or Halogen}";

C_31 = "C@4 ⟨(-H Or -C) (-H Or -C) (-H Or -C) -Hetero⟩";

C_20 = "{C@3 ⟨(-H Or -C) (-H Or -C) =C⟩ Or C@3 ⟨(-H Or -C) (:C) :C⟩}";

The third of these is an sp^3 hybridized carbon atom in the +1 oxidation state; the last is an sp^2 hybridized carbon in the zero oxidation state. Example atom types for log *P* calculation are shown in Table 5.

In our experience, the SUPER-SMILES approach to assigning these atom types has been more reliable than an earlier FORTRAN implementation of the same properties. Certainly, the SUPER-SMILES definitions are easier to debug and change than is FORTRAN code. The main caveat is that the SUPER-SMILES method is substantially slower than a FORTRAN implementation. Therefore, when profiling large databases of compounds, it is expedient to use the compiled code and only apply SUPER-SMILES rules to molecules which fail in some way. In this fashion, the SUPER-SMILES method has assisted us in correcting and updating the FORTRAN version.

SUPER-SMILES has also found application in force-field parameter assignment, particularly in identifying rotatable bonds according to whether the constituent atoms are sp^2 or sp^3 hybridized.

DISCUSSION

Our aim has been to implement a line-notation interpreter and generator in our CAMD suite, PROMETHEUS, which

would afford a simple and single mode of use in molecular structure matching, derivatization, and construction. The main interest has been in expressing chemical transformations as simply derivable rules which can be systematically applied to libraries of molecules. For all the reasons described above, SMILES[2,3] is the language of choice to exploit. A fresh implementation can be customized for one's own environment, according to one's own priorities and the local availability of other software packages. While it is always hard to justify a reimplementation of methodology which is otherwise commercially available, for our own purposes, the additional effort is outweighed by having the tools readily available within a homogeneous CAMD suite along with customizable flexibility. Although we have utilized the core features and overall philosophy of SMILES, our implementation differs in several key respects: principally, the absence of built-in chemical rules, our syntax for pattern matching, and the way in which we encode transformations. Most other SMILES implementations deviate in small ways from the standard. It should be clear that others who choose to develop a SMILES interpreter could incorporate the features described here.

The decision to overlook aromatic assignment at the interpretation and compilation stage is justifiable because of the overheads involved. As has been seen, it allows us greater flexibility in the use of lowercase letters in SUPER-SMILES strings, though the penalty is the need for macros to express the various aromatic representations. Nevertheless, the way in which aromatic systems are assigned in commercial molecular modeling packages has been far from standardized: there remain problems both with a common agreement on which ring systems (beyond the most obvious) are to be regarded as aromatic, and the import and export of molecular data with or without aromatic bonds are not consistently handled between packages. It has been our view that aromaticity can be adequately treated with explicit bond types rather than by a specific set of atom types and is best left to the user to control.

In molecular modeling, it is typically the case that the 2D and 3D worlds are separate. It is possible to work with SUPER-SMILES strings which contain no hydrogens and encounter no difficulties in the sphere of 2D database searching, whereas a representation with fully specified hydrogens is necessary for applications such as generation of structures for molecular docking. (Alternatively, hydrogen addition may be carried out with external software.) This distinction is in line with our historical commitment to structure based ligand design, in which we are mainly dealing with 3D molecular structures with fully satisfied valencies; but it does not preclude the use of strings without hydrogens for applications such as virtual screening.

While the language of chemistry[33,34] and that of reactions is large, the graph theoretical description of the underlying processes may be expressed compactly. Although many representations of reactions include some notion of electron counting, at the connection table level we can reduce most transformations to additions and deletions of atoms, accompanied by changes in bond orders where necessary. This scheme will therefore be subject to the same limitations that undermine the connection table, as has recently been pointed out by Gasteiger.[35] It is our belief that by restricting our attention to the starting and finishing forms of structures, we can represent most of the derivatizations used in medicinal chemistry with operations on the connection table.

The largest set of SMILES extensions that currently exists is SMARTS, whose original implementation is available in the Daylight software;[14] a slightly different version is available within MOE.[15] SMARTS enables substructure searching and introduces variable expressions for matching alternative or conditional environments. SMARTS simplifies pictorial substructure matching (as might be achieved with a drawing utility) to operations with character strings. As with SMILES, the elements of SMARTS (atom constraints, bond types, and restrictions thereon) all correspond with built-in pieces of a molecule data structure.

Daylight has also introduced SMIRKS,[18] a new notation for describing "generic reactions", derived from SMARTS and SMILES. The notation relies on explicit separation of "reactants", "agents", and "products"; atom mapping labels keep track of specified sites in the transformation from reactant to product. Consequently although there is the advantage of a compact syntax, the syntax itself cannot actually carry out a modification of a molecule structure without the aid of external functions: its application to matching is therefore restricted to searching reaction databases. Important properties of SMIRKS include the fact that its expressions are not limited to bona fide chemical reactions and that, to describe a class of functionalizations which may occur in many different environments, it is only necessary to detail the atoms or bonds which actually change. What appears to be a difficulty with the syntax is the way in which hydrogens are handled: there are acknowledged inconsistencies with both SMILES and SMARTS.[18]

On the other hand, Daylight's Reaction Toolkit does allow description of reactions with the line notation itself and offers additional programs for utilizing that description. It rests on a new set of data structures and introduces new paradigms for reaction specification. Part of the motivation for that approach is connected with the storage of reaction information in databases: representing structural changes alone is insufficient; for completeness it has been necessary to record conditions, solvent, and (in earlier releases) "agents" of reaction such as catalysts, all of which can be placed in ancillary database fields. The growing need to search reaction databases has imposed additional demands on the data structure employed. If one's interest is mainly in less ambitious molecular manipulations or library building, a simpler approach is possible, one which circumvents the need to devise complex programs and data structures for the purpose. Indeed, our implementation of SUPER-SMILES admits no place for an agent.

There have been other ways reported of transforming SMILES expressions. Martin and van Drie[36] demonstrated one of the earliest applications of SMILES-string manipulation. Using the ALADDIN Control Language, functions (referred to collectively as MODSMI) were implemented to carry out various types of transformation. Key molecular environments were identified by SMILES substructures, target atoms within them were labeled, and operations such as substitution and deletion were performed on those atoms.

Ho and Marshall described DBMAKER,[37] which generates SMILES strings with string splicing operations and creates molecular databases from them. Lists of SMILES functional groups are string-catenated in random combinations; forma-

tion of ring assemblies is achieved with utilities which systematically generate "masks" containing pairs of integers representing ring fusions. More complex structures are generated with "crossover" techniques (akin to those used in genetic algorithms) wherein pairs of strings are spliced together.

While both the MODSMI and DBMAKER programs illustrate how new molecular structures can be built from SMILES strings, neither involves any extensions to the SMILES language itself; all the new functionality is implemented as external processing programs. Although string processing is easy with current computer software, the inherent disadvantages are 2-fold: that the external codes require a new and distinct set of paradigms for expressing the structural operations, and that a SMILES expression is more than a string. SMILES is a language, and it should be possible to represent structural changes with that language more concisely than with external methods.

Our presentation of SUPER-SMILES has also demonstrated that a line notation can find a variety of applications in a molecular modeling environment. Many tasks, both complex and frequently employed, can be expressed in SUPER-SMILES and applied by standard protocols. Our sets of extensions enable a single language and suite of interpretation functions to operate successfully in a wide variety of contexts.

ACKNOWLEDGMENT

The molecular modelers at Proteus Molecular Design, in particular John Liebeschuetz and Chris Murray, are thanked for continuous input and suggestions.

REFERENCES AND NOTES

(1) Wiswesser, W. J. Historic Development of Chemical Notations *J. Chem. Inf. Comput. Sci.* **1985**, *25*, 258−263.
(2) Weininger, D.; Weininger, A.; Weininger, J. L. SMILES, A Modern Chemical Language and Information System. *Chem. Des. Autom. News* **1986**, *1*, 2−15.
(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) Read, R. C. A New System for the Designation of Chemical Compounds. 1. Theoretical Preliminaries and the Coding of Acyclic Compounds. *J. Chem. Inf. Comput. Sci.* **1983**, *23*, 135−149.
(5) Wiswesser, W. J. *A Line-Formula Chemical Notation*; Crowell: New York, 1954.
(6) Bremser, W. HOSE-A Novel Substructure Code. *Anal. Chim. Acta* **1978**, *103*, 355−365.
(7) Read, R. C. A New System for the Designation of Chemical Compounds. 2. Coding of Cyclic Compounds. *J. Chem. Inf. Comput. Sci.* **1985**, *25*, 116−128.
(8) Herndon, W. C. Canonical Labelling and Linear Notation for Chemical Graphs. In *Chemical Applications of Topology and Graph Theory*; King, R. B., Ed.; Studies in Physical and Theoretical Chemistry; Elsevier: Amsterdam, 1983; Vol. 28, pp 231−242.
(9) Herndon, W. C.; Bertz, S. H. Linear Notations and Molecular Graph Similarity *J. Comput. Chem.* **1987**, *4*, 367−374.
(10) Barnard, J. M.; Jochum, C. J.; Welford, S. M. *ROSDAL: A universal structure/substructure representation for PC-host communication and standards*, ACS Symposium Series 400; American Chemical Society: Washington, D.C., 1989; pp 76−81.
(11) Huixiao, H.; Xinquan, X. ESSESA: An Expert System for Structure Elucication from Spectra. 3. LNSCS for Chemical Knowledge Representation *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 116−120.
(12) *SYBYL molecular modeling package*; Tripos Associates, Inc.: St. Louis MO.
(13) Weininger, D.; Weininger, A. GEMINI: A Generalised Connection Table Language and Interpreter. In *Chemical Structures 2*; Warr, W., Ed.; Springer: Berlin, 1993; pp 195−206.
(14) *SMILES Toolkit*; Daylight Chemical Information Systems, Inc.: Santa Fe, NM.
(15) *Molecular Operating Environment*; Chemical Computing Group, Inc.: Montreal, Canada.
(16) CACTVS: http://www2.ccc.uni-erlangen.de/software/cactvs/papers.html.
(17) Pearlman, R. S. *CONCORD*; Tripos Associates, Inc.: St. Louis MO.
(18) *Daylight Theory Manual 4.61*; Daylight Chemical Information Systems, Inc.: Santa Fe, NM, 1997; http://www.daylight.com/dayhtml/doc/theory.
(19) Clark, D. E.; Frenkel, A. D.; Levy, S. A.; Li, J.; Murray, C. W.; Robson, B.; Waszkowycz, B.; Westhead, D. R. *J. Comput. Aided Mol. Des.* **1994**, *9*, 13.
(20) Barnard, J. M. Structure Representation and Searching. In *Chemical Structure Systems*; Ash, J. E., Warr, W. A., Willet, P., Eds.; Ellis Harwood: Chichester, U.K., 1991; pp 9−56.
(21) Ball, J.; Fishleigh, R. B.; Greaney, P.; Li, J.; Marsden, A.; Platt, E.; Pool, J. L.; Robson, B. In *Chemical Structure Information Systems: Beyond the Structure Diagram*; Bawden, D., Mitchell, E. M., Eds.; Ellis Horwood: Chichester, U.K., 1990; p 107.
(22) Ousterhout, J. *Tcl and the Tk Toolkit*; Addison-Wesley: Reading, MA.
(23) Bone, R. G. A. Manuscript in preparation.
(24) *Converter*, Version 2.3; MSI: San Diego, CA.
(25) Siani, M. A.; Weininger, D.; Blaney, J. M. CHUCKLES: A Method for Representing and Searching Peptide and Peptoid Sequences on Both Monomer and Atomic Levels. *J. Chem. Inf. Comput. Sci.* **1994**, *34*, 588−593.
(26) Bone, R. G. A.; Firth, M. A.; Sykes, R. A.; Murray, C. W.; Li, J. PRO_ANALOG: Automated Analog Building According to Principles of Medicinal Chemistry. *Rational Drug Design*; ACS Symposium Series; American Chemical Society: Washington, D.C., in press.
(27) *Introduction to Automata Theory, Languages, and Computation*; Hopcroft, J. E., Ullman, J. D., Eds.; Addison-Wesley: Reading, MA, 1979.
(28) *Lex-A Lexical Analyzer Generator*; Lesk, M. E., Eds.; Computer Science Technical Report No. 39; Bell Laboratories: October 1975.
(29) *Yacc-Yet Another Compiler-Compiler*; Johnson, S. C., Ed.; Computer Science Technical Report No. 32; Bell Laboratories: July 1975.
(30) Weininger, D.; Weininger, A.; Weininger, J. L. SMILES. 2. Algorithm for Generation of Unique SMILES Notation. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 97−101.
(31) Bush, B. L.; Sheridan, R. P. PATTY: A Programmable Atom Typer and Language for Automatic Classification of Atoms in Molecular Databases. *J. Chem. Inf. Comput. Sci.* **1993**, *33*, 756−762.
(32) Viswanadhan, V. N.; Ghose, A. K.; Revankar, G. R.; Robins, R. K. Physicochemical Parameters for Three-Dimensional Structure Directed Quantitative Structure−Activity Relationships. 4. Additional Parameters for Hydrophobic and Dispersive Interactions and Their Application for an Automated Superposition of Certain Naturally Occurring Nucleoside Antibiotics. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 163−172.
(33) Gordon, J. E.; Brockwell, J. C. Chemical Inference. 1. Formalization of the Language of Organic Chemistry: Generic Structural Formulas. *J. Chem. Inf. Comput. Sci.* **1983**, *23*, 117−134.
(34) Gordon, J. E. Chemical Inference. 2. Formalization of the Language of Organic Chemistry: Generic Systematic Nomenclature. *J. Chem. Inf. Comput. Sci.* **1984**, *24*, 81-92.
(35) Bauerschmidt, S.; Gasteiger, J. Overcoming the Limitations of a Connection Table Description: A Universal Representation of Chemical Species. *J. Chem. Inf. Comput. Sci.* **1997**, *37*, 705−714.
(36) Martin, Y. C.; van Drie, J. H. Identifying Unique Core Molecules from the Output of a 3-D Database Search. In *Chemical Structures 2*; Warr, W., Ed.; Springer: Berlin, 1993; pp 315−326.
(37) Ho, C. M. W.; Marshall, G. R. DBMAKER: A set of programs to generate three-dimensional databases based upon user-specified criteria. *J. Comp.-Aided Mol. Des.* **1995**, *9*, 65-86.

CI990422W