Molecular Query Language (MQL)—A Context-Free Grammar for Substructure Matching

Ewgenij Proschak,*[,†] Jörg K. Wegner,[‡] Andreas Schüller,[†] Gisbert Schneider,[†] and Uli Fechner[†]

Johann Wolfgang Goethe-University, Institute of Organic Chemistry and Chemical Biology, Chair for Chem- and Bioinformatics, Siesmayerstr. 70, D-60323 Frankfurt am Main, Germany, and Tibotec BVBA, Generaal De Wittelaan L11 B3, 2800 Mechelen, Belgium

Received July 19, 2006

We have developed a Java library for substructure matching that features easy-to-read syntax and extensibility. This molecular query language (MQL) is grounded on a context-free grammar, which allows for straightforward modification and extension. The formal description of MQL is provided in this paper. Molecule primitives are atoms, bonds, properties, branching, and rings. User-defined features can be added via a Java interface. In MQL, molecules are represented as graphs. Substructure matching was implemented using the Ullmann algorithm because of favorable run-time performance. The Ullmann algorithm carries out a fast subgraph isomorphism search by combining backtracking with effective forward checking. MQL software design was driven by the aim to facilitate the use of various cheminformatics toolkits. Two Java interfaces provide a bridge from our MQL package to an external toolkit: the first one provides the matching rules for every feature of a particular toolkit; the second one converts the found match from the internal format of MQL to the format of the external toolkit. We already implemented these interfaces for the Chemistry Development Toolkit.

INTRODUCTION

Substructure searching is a common task in cheminformatics. The goal of a substructure search is to find a chemical motif or structure (the “query structure”) in one or more other chemical structures (“target structures”). If a query structure is detected in a target structure, the atoms of the query structure are mapped to the atoms of the target structure and this mapping is returned to the user. Such a mapping can be visualized by highlighting the query structure within the chemical structure. A query can be defined by using either a substructure query language (text-based) or a suitable molecular drawing program (graphics-based). The first research work related to substructure searching started as early as 1960,[1] and a recent review pinpoints its enduring significance.[2] Query-matching scenarios include database searching, calculation of substructure-based descriptors, and assignment of atom types. Two such substructure query languages are widespread among the cheminformatics community: SMARTS[3] and SLN.[4] We have developed a substructure query language (“Molecular Query Language”, MQL), which is grounded on a context-free grammar. MQL originated from our need to define complicated substructure expressions for fast, automated de novo design algorithms.[5] MQL is not qualified for the definition of relations between different substructure queries like, for example, GENIE.[6] Different from ALEMBIC[7] and the Scientific Vector Language,[8] MQL is not a programming language tailored toward cheminformatics applicability. The sole focus of MQL lies in the definition of feature-rich substructure queries.

* Corresponding author tel: +49-69 798 24878; fax: +49-69 798 24880; e-mail: proschak@bioinformatik.uni-frankfurt.de.
[†] Johann Wolfgang Goethe-University.
[‡] Tibotec BVBA.

In general, any substructure query language should meet two requirements. First, from the perspective of a user, it should be easy to learn, and even elaborate query expressions should be readable. The second requirement originates from the developer’s perspective: given the inherent complexity of a substructure query language, it should be designed in a way that supports straightforward implementation. In MQL, two-dimensional molecular substructure queries are represented by a set of ASCII characters. Atom and bond symbols and the specification of branching and ring closures by brackets follows a concept first introduced by SMILES.[9] Even though MQL could be employed as a line notation to represent molecules, this particular usage was not regarded during development. Instead, we focused on the suitability of MQL as a line notation for substructure queries.

MOLECULAR GRAPHS AND QUERY GRAPHS

A molecular graph is constituted by nodes representing the atoms of a molecule and by the edges that correspond to the bonds of a molecule. Analogously, a query graph comprises nodes and edges that represent atoms and bonds of a query structure. Properties are assigned to the nodes and edges of both a molecular and a query graph. Among such properties are element symbols in the case of nodes, or bond orders in the case of edges. The difference between a molecular and a query graph is the assignment of properties to nodes and edges. Whereas the node of a molecular graph can only be assigned to a single element symbol, no such restriction is imposed upon the node of a query graph. It is totally valid to have a query structure where a particular atom may be either a carbon atom or a nitrogen atom. The properties that are assigned to nodes and edges can be subdivided into nominal and numerical properties. Nominal properties correspond to Boolean values, and numerical properties are assigned discrete or continuous numerical values.

LANGUAGE DEFINITION

The first step in the processing of a text-based substructure query is the parsing of the query. In computer science, the procedure of evaluating a sequence of characters to assess whether it adheres to a given formal grammar is called “parsing”. The computer program that carries out this task is a “parser”. A parser recognizes the internal structure of an input sequence and converts it into a data structure, usually a graph representation. We defined MQL in terms of context-free grammar (CFG) to alleviate its processing by a computer. CFGs are commonly employed in linguistics and computer science to describe formal languages.[10] A CFG is a generative grammar. Generative grammar specifies an algorithm that generates expressions in the respective language. The deployment of a CFG paves the way to a straightforward implementation of a query language parser with the use of parser-generating software. If a language is nontrivial, as it is for sure the case with a substructure query language, the hand-crafting of a parser is difficult and error-prone no matter how much care is taken. This problem can be circumvented if a language is based on a CFG. We employed the open-source Java Compiler Compiler (JavaCC)[11] to generate a parser. The CFG is written in extended Backus−Naur Form (BNF)[12,13] so that it is interpretable by JavaCC. The compilation of such a CFG file by JavaCC yields a parser written in Java. The complete language specification as a CFG is depicted in Figure 1. The automatically generated parser is then able to analyze a MQL expression step by step until the expression comprises only terminal signs.

ATOM SPECIFICATION

An atom is specified by its atomic symbol (C, N, or Li). Furthermore, aromatic nitrogen and carbons are indicated by small letters (c and n). An exclamation sign (“!”) defines a negation of an atom (for example, “!C” matches all but carbon atoms). Any atom (the wildcard atom) is specified by the character “*”. Special atom signs are “Heavy”, “Hetero”, and “Halogen”, which are defined to comprise all atoms except for hydrogen, all atoms apart from carbon and hydrogen, and halogen atoms, respectively. An atom sign may be directly followed by an identifier (C1 or N12). The so-called unselection sign “’” (an apostrophe) indicates that an atom must be matched but is not part of the returned matching (see Figure 2 for an example). An atom sign is optionally followed by atom properties within square brackets; these properties are described in the following.

BOND SPECIFICATION

A bond is specified by a bond sign where “-” represents a single bond, “=” a double bond, “#” a triple bond, “:” an aromatic bond, and “~” any bond. Corresponding to the atom sign, a bond sign is followed by an optional identifier and properties within square brackets. An exclamation sign can be used to negate the bond properties. In contrast to SMILES/SMARTS and SLN, single bonds may not be omitted but must be written explicitly.

BRANCHING AND RINGS

The grammar allows construction of query graphs with branching and ring closures. Opening and closing round brackets allow the user to specify branched queries, for example, “C(-F)(-F)-F” for trifluormethane. The specification of queries with ring systems is supported by the ring closure sign “$” (as defined with the CFG ring-variables <Ringbond> and <Ringclosure>). For example, in MQL, benzene is written as c$1:c:c:c:c:c$:1.

PROPERTIES

Properties follow atom and bond signs in square brackets and specify further features of the atom or bond. From the very start, the design of our language and its implementation was guided by the maxim to facilitate the definition of additional properties by a developer. Nevertheless, the available set of properties should be sufficient to enable typical query searches. We allowed nominal, numerical, and user-defined properties. Nominal features are recalled simply by their name. The return value is true if the atom or bond has this property and is false otherwise. Numerical properties consist of a name, a value, and a relation sign. For example, in the expression “order<=2”, “order” is the name of the numerical property,“2” is the value, and “<=” is the relation sign. Values are decimal numbers; relation signs are common algebraic relational operators (“=”, “<”, “>”, “<=”, and “>=”). The evaluation of a numerical property proceeds as follows: First, the value of the atom or bond which corresponds to the property name is interrogated. The received value is then compared with the property value according to the relation sign. The result of this procedure is true if the evaluation of the expression “‘received value’ ‘relation sign’ ‘property value’” evaluates to an algebraic true.

Nominal Atom Properties. The element symbol is a basic property of an atom. All symbols contained in the periodic table of elements are allowed. The first letter has to be a capital one, for example, “C”, “Li”, or “O”. The property aromatic is true if an atom is a member of an aromatic system as defined by Hueckel.[14] The nominal property aliphatic is just the opposite of aromatic: atoms are defined as aliphatic if they are not aromatic according to the Hueckel rule. The ring property of an atom is set to true if an atom is part of a cycle within the molecular graph. The size of the cycle is not considered by this feature. We defined the hybridization state property as nominal, denoted as “sp”, “sp2”, and “sp3”.

Nominal Bond Properties. The properties aromatic, aliphatic, and ring can also be assigned to bonds. A bond with the feature aromatic connects two aromatic atoms. The other two features aliphatic and ring are defined analogously.

Numerical Atom Properties. The atomic number of an atom corresponds to the atomic numbers as defined in the periodic table of elements. For example, 6 is the atomic number for carbon, 8 for oxygen. The numerical properties *implicitHydrogens*, *explicitHydrogens*, and *allHydrogens* refer to the number of hydrogen atoms that are connected to an atom. The property *implicitHydrogens* corresponds to hydrogen atoms which are not part of the molecular graph. In contrast, “*explicitHydrogens*” have to appear in the

```
Query ::= (<Atom>|<MarcushQuery>) (<Ringclosure>|<Ringbond>)*
(<Branch>)* (<Bond> <Query>)?
Atom ::= (<NOTSIGN>)? <Symbol> (<Identifier>)? (<UNSELECTION_FLAG>)?
(<OPEN_SQUARE_BRACKET><Properties> <CLOSE_SQUARE_BRACKET>)?
Identifier ::= <DIGITS>
Properties ::=
(<PROPERTY_SIGNS>|<DOUBLE_BOND>|<NOTSIGN>|<DIGITS>|<LETTERS>)*
Symbol ::= (<SYMBOL_SIGN>|<HEAVY_ATOM>|<HETERO_ATOM>|<HALOGEN_ATOM>
|<AROMATIC_CARBON>|<AROMATIC_NITROGEN>|<ANY_ATOM>)
Bond ::= (<NOTSIGN>)? <Bondsymbol>
(<OPEN_SQUARE_BRACKET><Properties><CLOSE_SQUARE_BRACKET>)?
Bondsymbol ::=
(<ANY_BOND>|<SINGLE_BOND>|<DOUBLE_BOND>|<TRIPLE_BOND>|<AROMATIC_BOND>)
Ringbond ::= <RINGSIGN> <Bond> <Identifier>
Ringclosure ::= <RINGSIGN> <Identifier>
Branch ::= <OPEN_ROUND_BRACKET> <Bond> <Query> <CLOSE_ROUND_BRACKET>
MarcushQuery ::= (<MARCUSH_R>|<MARCUSH_X>) <DIGITS>
NOTSIGN ::= “!”
UNSELECTION_FLAG ::= “′”
OPEN_SQUARE_BRACKET ::= “[”
CLOSE_SQUARE_BRACKET ::= “]”
HEAVY_ATOM::= “Heavy”
HETERO_ATOM::= “Hetero”
HALOGEN_ATOM::= “Halogen”
AROMATIC_CARBON::= “c”
AROMATIC_NITROGEN::= “n”
ANY_ATOM::= “*”
SINGLE_BOND::= “-”
DOUBLE_BOND::= “=”
AROMATIC_BOND::= “:”
TRIPLE_BOND::= “#”
ANY_BOND::= “~”
DIGITS ::= ({“0”-“9”})+
LETTERS::= ({"a"-"z","A"-"Z"})+
PROPERTY_SIGNS::= (“{”|“}”|“&”|“|”|“+”|“.”)+
SYMBOL_SIGN::=
"Ac"|"Ag"|"Al"|"Am"|"Ar"|"As"|"At"|"Au"|"B"|"Ba"|"Be"|"Bi"|"Bh"|"Bk"|"
Br"|"C"|"Ca"|"Cd"|"Ce"|"Cf"|"Cl"|"Cm"|"Co"|"Cr"|"Cs"|"Cu"|"Db"|"Ds"|"D
y"|"Er"|"Es"|"Eu"|"F"|"Fe"|"Fm"|"Fr"|"Ga"|"Gd"|"Ge"|"H"|"He"|"Hf"|"Hg"
|"Ho"|"Hs"|"I"|"In"|"Ir"|"K"|"Kr"|"La"|"Li"|"Lr"|"Lu"|"Md"|"Mg"|"Mn"|"
Mo"|"Mt"|"N"|"Na"|"Nb"|"Nd"|"Ne"|"Ni"|"No"|"Np"|"O"|"Os"|"P"|"Pa"|"Pb"
|"Pd"|"Pm"|"Po"|"Pr"|"Pt"|"Pu"|"Ra"|"Rb"|"Re"|"Rg"|"Rh"|"Rn"|"Rt"|"Ru"
|"S"|"Sb"|"Sc"|"Se"|"Sg"|"Si"|"Sm"|"Sn"|"Sr"|"Ta"|"Tb"|"Tc"|"Te"|"Th"|
"Ti"|"Tl"|"Tm"|"U"|"V"|"W"|"Xe"|"Y"|"Yb"|"Zn"|"Zr"
RINGSIGN::= "$"
MARCUSH_R::="R"
MARCUSH_X::="X"
```

Figure 1. Definition of the context-free grammar for MQL. The following BNF symbols have been employed: “::=” for assignment, “(...)” for grouping, “|” for alternative, “(...)?” for optional singular occurrence, “(...)+” for optional repetition once and more times, “(...)*” for optional repetition zero or more times, “{...}” for ranges or sets, and “<...>” for variables. Capital letters represent character tokens; small letters stand for nonterminals.

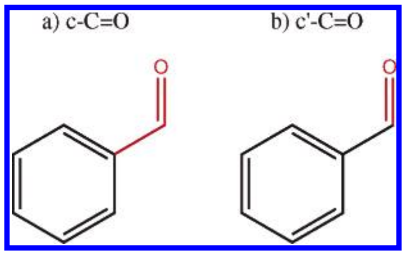


Figure 2. Exemplary demonstration of the unselection sign ‘.

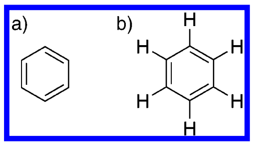


Figure 3. Depiction of the difference between molecular graphs with (a) implicit and (b) explicit hydrogens.

molecular graph, as depicted in Figure 3. “*allHydrogens*” is the sum of implicit and explicit hydrogens. The feature *explicitConnections* defines the number of edges of an atom in a molecular graph where the bond order and implicit hydrogens are not taken into account. On the contrary, the feature *valence* defines the sum of both explicit and implicit bond orders of an atom, thus corresponding to the chemical valence of the element. The number of bonds that originate from an atom can be queried with the property *totalConnections*. Bond orders are disregarded by *totalConnections*. The numerical feature *charge* refers to the formal charge of an atom. The smallest set of smallest rings (SSSR) comprises all smallest rings of a molecule (that is, all rings with the smallest number of atoms) from which all other rings of this molecule can be derived. The feature *belongsToSSSR* defines

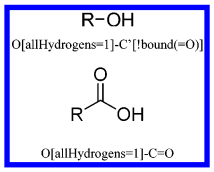


Figure 4. Example for the employment of the *bound*(...) property. Both functional groups, the alcohol group (upper structure) and carboxyl group (lower structure), contain the OH pattern. This might lead to problems if one explicitly wants to match alcohols but not carboxylic acids. In order to be able to distinguish between the two chemical groups, the chemical environment of the alcohol hydroxyl group has to be defined by the *bound*(...) property. Here, the MQL query for the alcohol group explicitly excludes a bound carbonyl group.

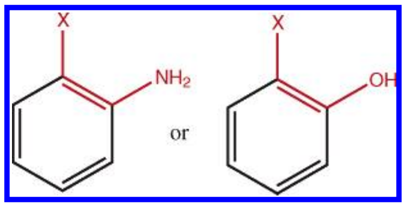


Figure 5. Depiction of the MQL query Halogen-c:c-*[{O&allHydrogens=1}|{N&allHydrogens=2}].

the number of smallest rings an atom belongs to. The size of the smallest ring an atom has membership of is defined by *smallestRingSize*. *ringConnections* corresponds to the number of bonds of an atom where the second atom that participates at this bond is member of a ring. The property *massNumber* is the mass number of the isotope.

Numerical Bond Properties. The only numerical bond property is *order*; it defines the bond order in its chemical sense.

User-Defined Properties. MQL can be extended by features that were not considered at the development stage. The Java interface *UserDefinedProperty* has to be implemented to provide access for a user-defined feature. In order to demonstrate the extendibility of MQL we have implemented the *ringSize=k* feature. If an atom is a part of a ring with size *k*, the return value of the query is true.

Atom Environments. The chemical environment of an atom can be described by the *bound*(...) atom feature. This feature accepts a complete MQL query as its argument inside the round brackets which defines the chemical environment surrounding the atom in structure and properties. An example is given in Figure 4 for distinguishing hydroxy from carboxy groups.

Combination of Properties. Multiple features can be combined by boolean conjunctions |, &, and ! and grouped with curly braces "{...}" to allow for complex queries. An example of such a combination of properties is illustrated in Figure 5.

MATCHING ALGORITHM

Graph data structures are commonly used to represent molecules in computers.[1] In MQL, the molecule as well as the query is internally represented as a graph. Thus, the matching algorithm can be regarded as the computational problem of finding the subgraph isomorphism of the query graph within the molecular graph.[15] There are different approaches to find a subgraph isomorphism: backtracking,[16] partitioning,[17] and clique detection on the association graph.[18] We decided to employ the Ullmann algorithm[19] as it has

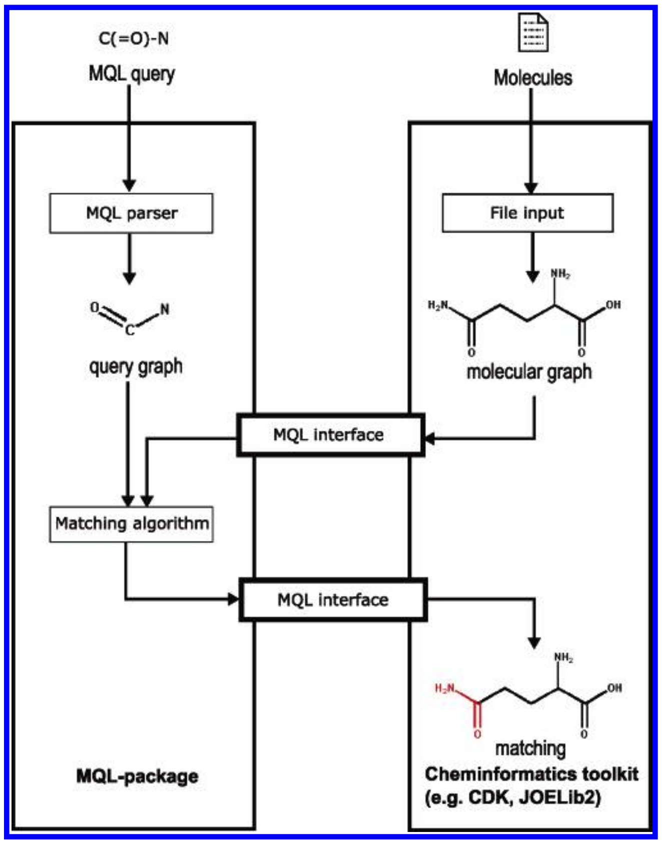


Figure 6. Workflow of a substructure search with MQL. The text-based MQL query is processed by the parser to yield the query graph (MQL parser); the cheminformatics toolkit processes the molecules to obtain their molecular graphs (File input). The matching routine of the MQL package is granted access to the molecule data structure of the cheminformatics toolkit by implementation of an interface. The Ullmann algorithm carries out the matching (Matching algorithm) and returns found matches to the cheminformatics toolkit via a second interface.

been shown to be suitable in terms of the required run time for this task.[20] The Ullmann algorithm performs a subgraph isomorphism search by combining backtracking with effective forward checking. The computation time of the Ullmann algorithm is in the best case bounded by $O(nm)$, in the worst case by $O(m^n n^2)$, assuming that m is the vertex number of the molecular graph and n is the vertex number of the query graph.[18] We have implemented this algorithm in Java as a part of our MQL package.

IMPLEMENTATION AND WORKFLOW

The MQL library was implemented in Java to allow for use on a variety of operating systems. Its software design was driven by the aim to facilitate the employment of different cheminformatics toolkits. Two Java interfaces offer the bridge from our MQL package to the cheminformatics toolkit of choice: the first one provides the matching rules for every feature of a particular toolkit; the second one converts the found match from the internal format of MQL to the format of the cheminformatics toolkit. We already implemented these interfaces for the Chemistry Development Toolkit.[21] Figure 6 shows the general workflow of our MQL package and highlights the steps where bridges to a cheminformatics toolkit play a role.

To demonstrate the features of our substructure query language, we have implemented a demo application that

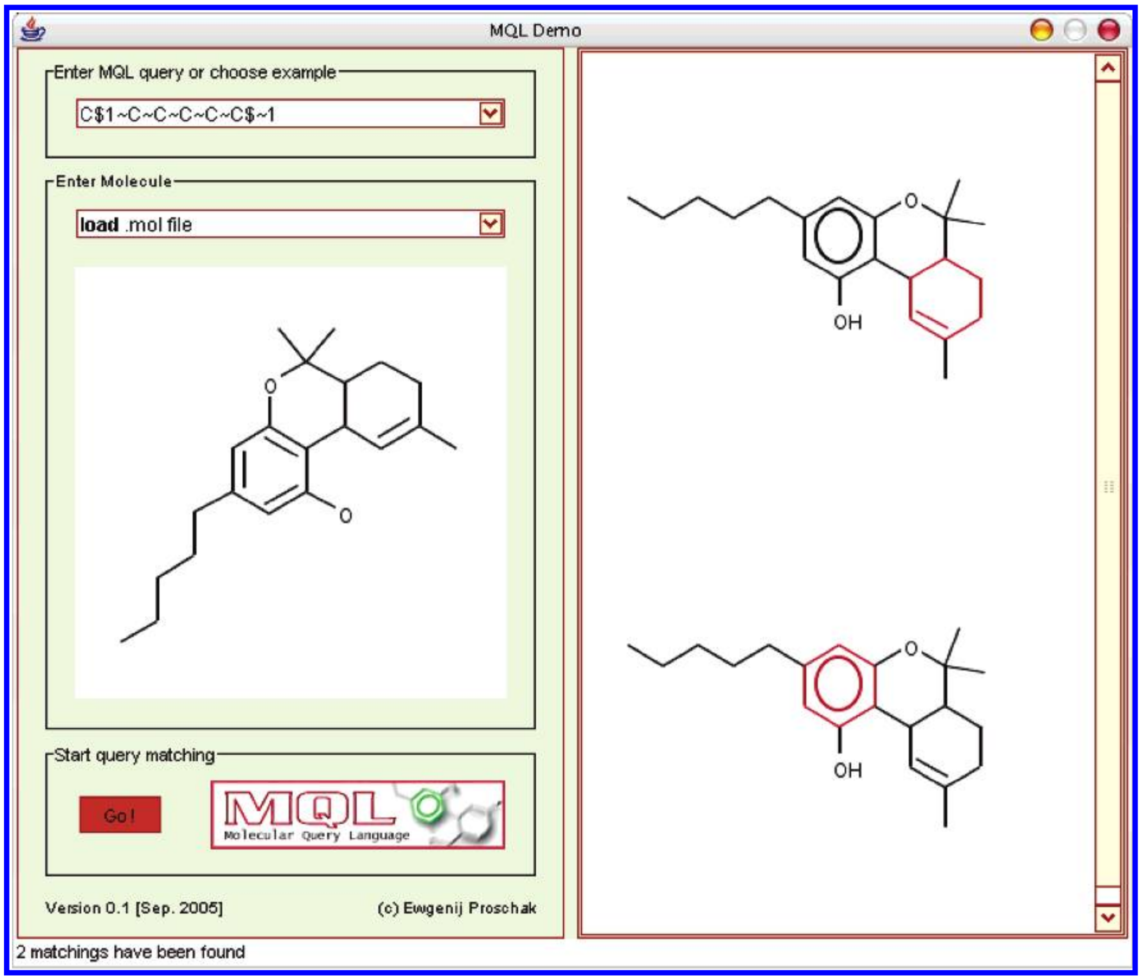


Figure 7. Screenshot of the MQL demo application. The MQL query specifies a substructure search for any six-member ring that consists of carbon atoms; tetrahydrocannabinol (THC) is the target structure. The two resulting matches are highlighted on the right side of the GUI.

comes with a graphical user interface (GUI). The user can enter a query and a target structure. The target structure can be specified by means of SMILES, a structure editor (JChemPaint[22]), and an MDL mol file.[23] Then, an attempt is made to locate the query in the target structure, and the matches—if any—are displayed by highlighting the query structure within the target structure. A screenshot of a demo application is shown in Figure 7. Both the demo application and MQL are accessible via the software link on our Web server at the URL http://www.modlab.de.

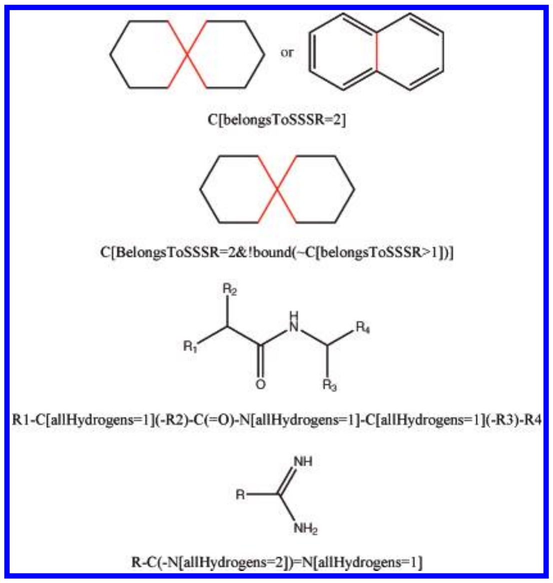
DISCUSSION

MQL does not underlie restrictions with respect to nominal atom and bond features; therefore, it opens a wide field of potential applications. For example, MQL can be extended for usage with reduced graph features, like atom-collapsed-donor and -acceptor properties. In this case, the usage is not even restricted to atom-based molecular information, but extension to an abstract pharmacophore concept is also feasible. The following list gives an overview of potential applications, which might benefit from the general property handling in MQL.

Atom Typing. One of the actual problems in cheminformatics is a missing common standard for chemical expert systems.[24] One reason for this problem is that the well-known PATTY algorithm is restricted to the capabilities of the query language used.[25] When no topological information is provided, then usually hard-coded algorithms are used to determine the hybridization state of atoms from their 3D coordinates. MQL does not have this restriction, since a bond-length property can be added to the list of valid numerical properties. By defining query triangles, all possible geometries can be covered. The main benefit lies in the high verbosity of this approach and the potentially straightforward exchange of query definitions. In this way, chemical expert systems could collaborate on the same query definition files by avoiding library-specific hard-coded algorithms.

Pharmacophore Mapping. In analogy to bond distances, feature distances on reduced graphs can be defined, which allows MQL to cover pharmacophore queries equally well.

Query Mining. Actual mining methods on molecules are typically restricted to multiple labeled nodes with nominal atom and bond features.[26,27] Typically the problem is even restricted to reduced graphs of single labeled atoms,[28] for example, element, atom, or pharmacophore types. Since it is unknown a priori which atom or feature reduction scheme might be useful for the mining problem at hand, the mapping of features is a critical step for any mining method. MQL does not restrict the mining space. Possible extensions of existing algorithms might include partial charges or polarizability properties. The only restriction is given by the run



```
C[belongsToSSSR=2]
C[BelongsToSSSR=2&!bound(~C[belongsToSSSR>1])]
R1-C[allHydrogens=1](-R2)-C(=O)-N[allHydrogens=1]-C[allHydrogens=1](-R3)-R4
R-C(-N[allHydrogens=2])=N[allHydrogens=1]
```

Figure 8. Examples of MQL queries and the corresponding matches indicated in the molecular graphs.

time for getting general and the most discriminatory query patterns.

Inductive Logic Programming. MQL could also be used for association-learning approaches using the efficient level-wise version space algorithm of Kramer and de Raedt.[27,29]

Query−Query Matching. Especially for reaction and query mining, MQL could be extended to find maximum common substructures in query definitions. This can be done by replacing the fast Ullmann subgraph isomorphism algorithm by a slower maximum common subgraph isomorphism algorithm, like finding cliques in the product graph[18] of two query graphs.[30] The areas of application as well as the run time are then only restricted by the heuristics used in the clique-detection algorithm as investigated by Gardiner and co-workers.[31]

CONCLUSION

We have developed and implemented a new substructure query language termed molecular query language. We offer open access to the specifications and an implementation in the programming language Java for noncommercial use. This is meant to support innovation through the open-source concept.[24,32] We wish to stress that MQL was not designed to substitute existing fingerprint-based similarity searching. Its primary purpose is to provide a language which facilitates unusual or complicated substructure mapping. Several such examples are shown in Figure 8. The language can be extended to support requirements that were not considered during development. Future work shall be directed toward support for Markush *R* groups and stereochemistry features. Comprehensive benchmarking will be essential for testing the usefulness of MQL. Speed improvements may be achieved by optimizing the MQL query. We intend to implement extensions to the MQL toolkit for approving relations between queries and virtual reaction schemes.

ACKNOWLEDGMENT

We are most grateful to Dr. John Bradshaw for valuable comments on the manuscript and pointing out alternative implementations to us. Yusuf Tanrikulu and Michael Meissner are thanked for their support in designing illustrations and the graphical user interface. U.F. is grateful for a fellowship granted by Sanofi-Aventis Pharma Deutschland GmbH and Goethe-University, Frankfurt am Main. This research was supported by the Beilstein-Institut zur Förderung der Chemischen Wissenschaften, Frankfurt am Main.

Note Added after ASAP Publication: This article was released ASAP on January 11, 2007, with minor errors in the references and in the section entitled “Branching and Rings”. The correct version was posted on January 18, 2007.

REFERENCES AND NOTES

(1) Gasteiger, J. *Handbook of Chemoinformatics*; Wiley-VCH: Weinheim, Germany, 2003; p 8.
(2) Willett, P. Searching Techniques for Databases of Two- and Three-Dimensional Chemical Structures. *J. Med. Chem.* **2005**, *48*, 4183−4199.
(3) Daylight Theory Manual. http://www.daylight.com/dayhtml/doc/theory/theory.toc.html (accessed Jan 2006).
(4) Ash, S.; Cline, M. A.; Homer, R. W.; Hurst, T.; Smith, G. B. SYBYL Line Notation (SLN): A Versatile Language for Chemical Structure Representation. *J. Chem. Inf. Comput. Sci.* **1997**, *37*, 71−79.
(5) Schneider, G.; Fechner, U. Computer-Based *de Novo* Design of Drug-Like Molecules. *Nat. Rev. Drug Discovery* **2005**, *4*, 649−663.
(6) Van, Drie, J. H.; Weininger, D.; Martin, Y. C. ALADDIN - An Integrated Tool for Computer-Assisted Molecular Design and Pharmacophore Recognition from Geometric, Steric, and Substructure Searching of Three-Dimensional Molecular Structures. *J. Comput.-Aided Mol. Des.* **1989**, *3*, 225−251.
(7) Maliski, E. G.; Latour, K.; Bradshaw, J. The Whole Molecular Design Approach to Drug Discovery. *Drug Des. Discovery* **1992**, *9*, 1−9.
(8) *MOE, Molecular Operating Environment*, version 2005.06; Chemical Computing Group Inc.: Montreal, Canada. http://www.chemcomp.com (accessed Dec 2006).
(9) 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.
(10) Hopcroft, J.; Motwani, R.; Ullmann, J. D. *Introduction to Automata Theory, Languages and Computation*; Addison Wesley Publishing Company: Boston, MA, 2001; p 169.
(11) Kodaganallur, V. Incorporating Language Processing into Java Applications: A JavaCC Tutorial. *IEEE Software* **2004**, *21*, 70−77.
(12) Knuth, D. Backus Normal Form vs Backus Naur Form. *Commun. ACM* **1964**, *7*, 735−736.
(13) Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation 04 February 2004. http://www.w3.org/TR/REC-xml/#sec-notation (accessed Jan 2006).
(14) Vollhardt, K. P. C.; Schore, N. E. *Organic Chemistry*, 4th ed.; Palgrave Macmillan: New York, 2002; p 6ff.
(15) Barnard, J. Substructure Searching Methods - Old and New. *J. Chem. Inf. Comput. Sci.* **1993**, *33*, 532−538.
(16) Dengler, A.; Ugi, I. A Central Atom Based Algorithm and Computer Program for Substructure Search. *Comput. Chem.* **1991**, *15*, 103−107.
(17) Sussenguth, E. H. A Graph-Theoretic Algorithm for Matching Chemical Structures. *J. Chem. Doc.* **1965**, *5*, 36−43.
(18) Messmer, B. Efficient Graph Matching Algorithms. Ph.D. Thesis, University of Bern, Bern, Switzerland, 1995; p 11ff.
(19) Ullmann, J. R. An Algorithm for Subgraph Isomorphism. *J. ACM* **1976**, *23*, 31−42.
(20) De Santo, M.; Foggia, P.; Sansone, C.; Vento, M. A Large Database of Graphs and Its Use for Benchmarking Graph Isomorphism Algorithms. *Pattern Recognit. Lett.* **2003**, *24*, 1067−1079.
(21) Steinbeck, C.; Han, Y. Q.; Kuhn, S.; Horlacher, O.; Luttmann, E.; Willighagen, E. The Chemistry Development Kit (CDK): An Open-Source Java Library for Chemo- and Bioinformatics. *J. Chem. Inf. Comput. Sci.* **2003**, *43*, 493−500.
(22) Krause, S.; Willighagen, E.; Steinbeck, C. JChemPaint - Using the Collaborative Forces of the Internet to Develop a Free Editor for 2D Chemical Structures. *Molecules* **2000**, *5*, 93−98.
(23) *CT File Formats*; Elsevier MDL: San Leandro, CA. http://www.mdl.com/downloads/public/ctfile/ctfile.pdf (accessed Jan 2006).
(24) Guha, R.; Howard, M.; Hutchison, G.; Murray-Rust, P.; Rzepa, H.; Steinbeck, C.; Wegner, J. K.; Willighagen, E. L. The Blue Obelisk−Interoperability in Chemical Informatics. *J. Chem. Inf. Model.* **2005**, *46*, 991−998.

(25) 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.

(26) Wegner, J. K.; Fröhlich, H.; Mielenz, H.; Zell, A. Data and Graph Mining in Chemical Space for ADME and Activity Data Sets. *QSAR Comb. Sci.* **2006**, *25*, 205−220.

(27) Helma, C.; Cramer, T.; Kramer, S.; De Raedt, L. Data Mining and Machine Learning Techniques for the Identification of Mutagenicity Inducing Substructures and Structure Activity Relationships of Noncongeneric Compounds. *J. Chem. Inf. Comput. Sci.* **2004**, *44*, 1402−1411.

(28) Bender, A.; Mussa, H. Y.; Glen, R. C.; Reiling, S. Molecular Similarity Searching Using Atom Environments, Information-Based Feature Selection, and a Naïve Bayesian Classifier. *J. Chem. Inf. Comput. Sci.* **2004**, *44*, 170−178.

(29) De Raedt, L.; Kramer, S. The Levelwise Version Space Algorithm and its Application to Molecular Fragment Finding. *Proc. 17th Int. J. Conf. Art. Intel., IJCAI 2001*; Nebel, B., Ed.; Morgan Kaufmann: San Francisco, CA, 2001; pp 853−862.

(30) Bron, C.; Kerbosch, J. Finding All Cliques of an Undirected Graph. *Commun. ACM* **1973**, *16*, 575−577.

(31) Gardiner, E. J.; Holliday, J. D.; Willett, P.; Wilton, D. J.; Artymiuk, P. J. Selection of Reagents for Combinatorial Synthesis Using Clique Detection. *Quant. Struct-Act. Relat.* **1998**, *17*, 232−236.

(32) McDonald, C. J.; Schadow, G.; Barnes, M.; Dexter, P.; Overhage, J. M.; Mamlin, B.; McCoy, J. M. Open Source Software In Medical Informatics − Why, How and What. *Int. J. Med. Inf.* **2003**, *71*, 175−184.

CI600305H