The Graph Grammar Library - a generic framework for chemical graph rewrite systems

Martin Mann[1], Heinz Ekker[2], and Christoph Flamm[2,3]

[1] Bioinformatics, Institut for Computer Science, University of Freiburg, 79106 Freiburg, Germany mmann@informatik.uni-freiburg.de
[2] Institute for Theoretical Chemistry, University of Vienna, Währingerstrasse 17, 1090 Vienna, Austria xtof@tbi.univie.ac.at

Abstract. Graph rewrite systems are powerful tools to model and study complex problems in various fields of research. Their successful application to chemical reaction modelling on a molecular level was shown but no appropriate and simple system is available at the moment.
The presented Graph Grammar Library (GGL) implements a generic Double Push Out approach for general graph rewrite systems. The framework focuses on a high level of modularity as well as high performance, using state-of-the-art algorithms and data structures, and comes with extensive documentation. The large GGL chemistry module enables extensive and detailed studies of chemical systems. It well meets the requirements and abilities envisioned by Yadav et al. (2004) for such chemical rewrite systems. Here, molecules are represented as undirected labeled graphs while chemical reactions are described by according graph grammar rules. Beside the graph transformation, the GGL offers advanced cheminformatics algorithms for instance to estimate energies ofmolecules or aromaticity perception. These features are illustrated using a set of reactions from polyketide chemistry a huge class of natural compounds of medical relevance.
The graph grammar based simulation of chemical reactions offered by the GGL is a powerful tool for extensive cheminformatics studies on a molecular level. The GGL already provides rewrite rules for all enzymes listed in the KEGG LIGAND database is freely available at http://www.tbi.univie.ac.at/software/GGL/.

1 Background

Graphs are powerful tools to represent and study all kind of data in any field of research. In order to generate graph structures of interest or to alter them according to some directive, graph transformations can be applied. A common approach is to formulate such transformations in terms of graph grammars or graph rewrite systems [5, 7, 37]. This enables a compact but very expressive representation of allowed alterations and allows for sound mathematical analyses of the problems [2, 19].

Here, we present our Graph Grammar Library (GGL), a fast and generic C++ framework to formulate and apply graph grammars. Beside the general

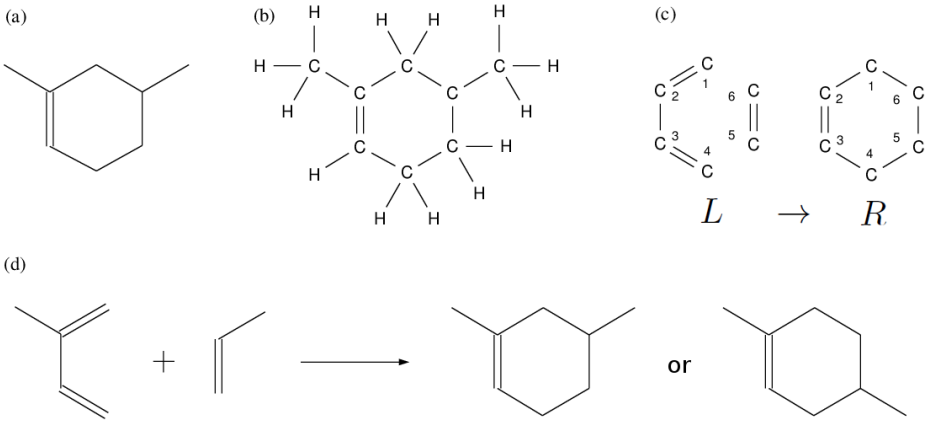


Fig. 1. Two graph representations of the same molecule at different levels of detail (a, b). Graphical depiction of the Diels-Alder reaction (d) for two molecules that enables two different result molecules. (c) The corresponding graph grammar rule $L \rightarrow R$.

graph rewrite system, we provide a specialized module to enable efficient graph grammar applications in chemical context. The power of such systems for chemical studies was highlighted by Yadav *et al.* [49] who emphasized the lack and need for an efficient implementation. The requirements and abilities sketched by Yadav *et al.* are well met by the capabilites of the GGL framework as presented within this manuscript.

To represent molecules in graph notation is well known in chemistry and found in any text book. Therein, atoms are represented by labeled nodes in the graph and the connecting chemical bonds are depicted by according edges (see Fig. 1 a) and b)). Such molecule graphs can be used to model chemical reactions by the application of graph grammar rules as shown in Fig. 1 c) and d). This small example shows already the power of graph rewrite systems, since a single graph grammar rule encodes all possible interaction combinations of the chemical reaction encoded (within the example two possible product molecules are possible, see Fig. 1 d)).

In the following, we will introduce the concept of graph rewrite systems in general and how it is implemented in detail within the GGL framework. We give insight into the techniques applied and the efforts done in order to gain an efficient and flexible framework for graph grammar applications. A major contribution of the GGL is its chemistry module tailored to fulfill the needs when representing chemical reactions via graph transformations as requested by Yadav *et al.* [49]. We show the generality of the GGL via several applications and elaborate its use for chemical studies. The latter is extended by providing a full set of predefined graph grammar rules for all enzymes listed in the KEGG LIGAND database [30] (Release 58.1 June 2011).

For simplicity, we will focus on labeled, undirected graphs in the following. Such a graph is given by a tuple (V, E, l_V, l_E) that is defined by the set of n nodes $V = \{v_1, \ldots, v_n\}$, the set of edges $E = \{\{v_i, v_j\} \mid v_i, v_j \in V\}$, and the label functions $l_V : V \to \Sigma^*$ and $l_E : E \to \Sigma^*$ that assign a label based on some alphabet Σ to each node and edge, respectively.

Graph rewrite systems are an algebraic approach to apply graph transformations [36]. It is defined by a set of rewriting rules of the form $L \to R$, with L defining the subgraph to be replaced during the transformation, the *pattern* or *left side* of the rule, and R stating the transformation result, i.e. the *replacement* or *right side* of the rule. Thus, graph rewriting requires the location of the pattern graph within the graph to transform. This *subgraph monomorphism problem* is a hard computational problem depending on the graph type [18]. Formally, given two graphs $G = (V^G, E^G, l_V^G, l_E^G)$ and $L = (V^L, E^L, l_V^L, l_E^L)$, one has to find an injective mapping $m : V^L \to V^G$ such that it holds

$$\forall_{v_i, v_j \in V^L} : m(v_i) \neq m(v_j) \quad \wedge \quad l_V^L(v_i) = l_V^G(m(v_i))$$
$$\forall_{\{v_i, v_j\} \in E^L} : \{m(v_i), m(v_j)\} \in E^G \quad \wedge \quad l_E^L(\{v_i, v_j\}) = l_E^G(\{m(v_i), m(v_j)\})$$

In the following, such a mapping m is called a *match* of the pattern L within target graph G and defines the subgraph of G isomorph to L.

When applying a graph grammar rule $L \to R$, one can follow the Double Push Out (DPO) approach [12, 36]. Therein, given a match m of L in some target graph G, the result graph G' is derived from G by (I) a relabeling of all matched nodes present in both V^L and V^R but showing different labels according to l_V^L, l_V^R, (II) the deletion of all nodes (and adjacent edges) present only in V^L but absent in V^R, and (III) the adding of all nodes exclusively present in V^R. Edges are handled accordingly based on E^L and E^R. For further details please refer to the standard literature, e.g. [12, 36].

Such graph grammar rules have been successfully applied to model chemical reactions with molecular detail [6, 19, 35]. Therein, a graph grammar rule encodes the molecule graph transformations resulting from a chemical reaction as exemplified in Fig. 1. The figure shows the power of such an encoding: the reaction is as much abstracted from the specific molecules as possible and implicitely encodes all interaction configurations.

At a higher abstraction level, frameworks have been introduced to encode and alter metabolic and signal-transduction networks based on graph rewrite systems [7, 9, 15, 16, 26, 48]. The focus here is to encode the kinetics and interactions of chemical reactions and not the mechanisms underlying them.

All the above mentioned frameworks are either prototypical implementations not tailored for a extensive application or very specialized systems designed for a specific type of experiment/problem. Within the context of general graph rewrite systems, other implementations have been introduced. Among them are the AGG [42, 43] and the GrGen.NET [21, 28] frameworks. While the AGG implementation was shown to be not well performing for larger data sets [21],

we found the very general GrGen.NET system not well placed to meet the requirements formulated by Yadav *et al.* [49]. The compilation of the graph grammar rules into executable code including the planned search strategy used to locate the patterns and the very expressive but complicated rule encoding makes the package powerful but not well placed for its integration into tools for chemical modelling to be used by chemists.

Here, we introduce the Graph Grammar Library (GGL) as a generic graph grammar framework with a strong focus on chemical applications as sketched by Yadav *et al.*. The GGL is not as general as e.g. the GrGen.NET system, since it currently allows only for a single label attribute associated to each node and edge, but it features an easy rule encoding in combination with a flexible and efficient rule application framework. In the following, we will introduce the GGL and the applied methods in detail.

2 The Graph Grammar Library

The Graph Grammar Library (GGL) is a generic C++ programming library that enables an easy setup of graph rewrite systems for labeled graphs. During the design of the library, we have focused on the following features:

Generality The GGL is built as a generic framework to be used for graph grammars on labeled graphs. The implementation uses the generic Boost Graph Library (BGL) [39] for its core graph representations. This enables the embedding of the graph rewrite functionalities into existing BGL-based projects. Our chemistry module supports the full atom range as well as standard chemoinformatics formats and libraries [33, 46]

Modularity The object-oriented modular design of the library enables a clear separation of functional units and the straight-forward implementation and use of specific functionalities. The interconnection of the modules is defined by clear and slim interfaces to enable a high level of transparency. The separation into context-specific sublibraries enables a selective use of the operations needed for a specific task.

Performance The main goal of the library is to provide a generic framework for computationally extensive applications. Thus, the implementation is tuned to be as fast as possible while maintaining a high level of generality and modularity. We apply efficient state-of-the-art algorithms [11, 23, 24] and data structures [13, 39], as detailed later, and have profiled and improved the code for a maximal performance.

Documentation In order to make the GGL easily applicable, we provide a well documented Application Programming Interface (API) for programmers as well as tutorials for end users focusing e.g. on the graph grammar rule encoding.

In the following, we will present the core functionalities of the library and how graph grammar rules are formulated and applied. Afterwards, we focus on the chemistry module of the GGL and the provided features and functionalities for chemoinformatical applications.

2.1 Subgraph Matching

As introduced above, the identification of matches m of a given left side pattern L of a graph grammar rule $L \rightarrow R$ is the central task of rule applications. The applied algorithm to solve the subgraph isomorphism problem is thus defining the performance of the whole rewrite engine [21]. Within the GGL, we apply the efficient `VF2`-algorithm and implementation introduced by Cordella *et al.* [11], which is among the fastest available [10]. We slightly adapted and extended the fast C-implementation and provide it within the GGL v4.0 package.

We have extended the implementation in two directions. First, we have introduced the handling of wildcard labels. This is needed to specify the existence of a given node without specifying the concrete label, e.g. to define an adjacent residual group of a molecule without any details. Furthermore, we have added an advanced constraint handling that can be used to enforce additional matching constraints. Among them are degree and adjacency constraints, negative application conditions (NAC), like the non-existence of an edge, as well as advanced chemistry related confinements.

Since we focused on a high level of modularity, we use a clear interface to provide the subgraph matching functionalities within the GGL. The `VF2`-algorithm is ported via this interface for its application. This enables the use of the available efficient `VF2`-implementation as well as the replacement of the `VF2`-algorithm with other subgraph matching approaches if needed. Since it was shown that the isomorphism problem can be solved efficiently for some types of graphs [18], it might be useful to apply a dedicated matching algorithm depending on the problem at hand. Furthermore, other matching approaches can be easily integrated [17, 37, 45]. The whole subgraph matching module is encapsulated into an independent library module, the SubGraph Matching (SGM) library, which is part of the GGL distribution.

2.2 Rule Encoding

Before we give details on the graph grammar rule applications we first introduce how rules are represented and to be specified within the GGL framework. Within the library, a graph grammar rule is represented by a specific graph object that encodes both, the left (L) and right side part (R) of a rule $L \rightarrow R$. Thus, it holds for each node and edge if it is present in L or R together with the according label. Formally, it is defined by the extended graph tuple $(V, E, l_V^L, l_V^R, l_E^L, l_E^R)$, which encodes the mapping of left and right side graphs. A node $v \in V$ *not present* within L will have an empty left side node label $l_V^L(v) = \lambda$. The same holds for edges. Thus we can derive the left side pattern $L = (V^L, E^L, l_V^L, l_E^L)$ by

$$V^L = \{v \in V \mid l_V^L(v) \neq \lambda\} \text{ and } E^L = \{\, \{v_i, v_j\} \in E \mid l_E^L(\{v_i, v_j\}) \neq \lambda\}.$$

The right side graph R is derived accordingly. Based on this representation, we have all the information at hand to apply the rule for each match m of L within some target graph G, since we know exactly the corresponding nodes/edges of L and R.

```
rule [
  ruleID "Diels-Alder reaction"
  context [
    node [ id 1 label "C" ]
    node [ id 2 label "C" ]
    node [ id 3 label "C" ]
    node [ id 4 label "C" ]
    node [ id 5 label "C" ]
    node [ id 6 label "C" ]
  ]
  left [
    edge [ source 1 target 2 label "=" ]
    edge [ source 2 target 3 label "-" ]
    edge [ source 3 target 4 label "=" ]
    edge [ source 5 target 6 label "=" ]
    constrainNoEdge [ source 1 target 5 ]
    constrainNoEdge [ source 4 target 6 ]
  ]
  right [
    edge [ source 1 target 2 label "-" ]
    edge [ source 2 target 3 label "=" ]
    edge [ source 3 target 4 label "-" ]
    edge [ source 4 target 5 label "-" ]
    edge [ source 5 target 6 label "-" ]
    edge [ source 6 target 1 label "-" ]
  ]
]
```

Fig. 2. The GML rule encoding of the graph grammar rule presented in Fig. 1.

For an easy and readable specification of graph grammar rules, we use an encoding in the Graph Modelling Language (GML) format [25]. This key-value pair structured format enables a compact and human-readable encoding while it is still machine-parsable due to its simple grammar. Figure 2 shows the GML encoding of the rule presented in Fig. 1. Note, all unchanging nodes and edges part of L *and* R are defined within the `context` section (contributing to both l_V^L and l_V^R), while left/right side graph specific nodes/edges and their labels are given in the according `left`/`right` sections and will only define the according l_V^L/ l_V^R data, respectively.

If the label of some nodes/edges of the pattern is of no interest for the matching, one can specify a wildcard label (e.g. *) by adding the according key-value entry `wildcard "*"`. The subgraph matching engine will now match all nodes/edges from the left side pattern L that show the wildcard label, e.g. `node [ id 1 label "*" ]`, to any other nodes/edges in the target graph without further label comparisons.

Often, the specification of wildcard labels makes the rule too general and might enable unintended rule applications. To tackle this problem, we support the specification of additional matching constraints that have to be ensured by the matching procedure. The simplest example is to restrict the allowed labels for a node defined with wildcard label to a given set of labels:

```
constrainNode [ id 1 op = nodeLabels [ label "C" label "N" ] ]
```

The reverse constraint can be encoded by changing the operator value from "`=`" to "`!`" to disallow the given labels. Other constraints supported for general graph grammars are the restriction of edge labels or node degree, the specification of

required/forbidden adjacent nodes or edges, or the explicit prohibition of the existance of an edge between two nodes. For further details on the constraints supported and their encoding we refer to the according tutorial part of the GGL v4.0 distribution.

2.3 Rule Application

The rule application follows the Double PushOut (DPO) approach [12,36]. Given a graph grammar rule $L \rightarrow R$ encoded by a rule graph $(V, E, l_V^L, l_V^R, l_E^L, l_E^R)$ as defined above and a target graph G onto which the rule is to be applied. First, all matches m of the rule's left side pattern L within G have to be indentified. The GGL uses to this end its subgraph matching interface and (as default) the `VF2`-algorithm [11] as already introduced. For each match m, the following procedure is applied to generate the result graph G' for the current match and rule:

1. Remove from G all nodes exclusive within L

$$V^{G'} = V^G \setminus \{m(v) \mid v \in V \wedge l_V^R(v) = \lambda\}$$

   and all edges exclusive within L or adjacent to removed nodes.

$$E^{G'} = E^G \setminus \{\ \{m(v_i), m(v_j)\} \mid\ \{m(v_i), m(v_j)\} \in E \\ \wedge\ (l_E^R(\{m(v_i), m(v_j)\}) = \lambda\ \vee\ l_V^R(v_i) = \lambda \vee l_V^R(v_j) = \lambda)\ \}$$

2. Relabel within G' all nodes/edges showing different (non-empty) labels within L and R.
3. Add to G' all nodes/edges exclusive within R and label accordingly.

All resulting graphs G' are then forwarded to a provided instance of a well defined graph reporter interface. This enables application specific post-processing and storing of the graphs resulting from the graph rewrite. Within the chemistry module, described next, this is for instance used to apply sanity checks on the produced molecule graphs, to convert and store molecule graphs into canonical string formats, to gather reaction product information, or to compute the reaction rate of the specific reaction defined by the rule application. In a more algorithmic context, such a graph reporter can also trigger another recursive iteration of rule applications resulting in a depth-first search/traversal of the graph space encoded by the graph grammar. An according generic implementation is provided and applied within the GGL framework.

2.4 Chemistry Package

As discussed in the introduction, chemistry and especially chemical reactions are a well placed target for the application of graph rewrite systems [6, 35]. The GGL provides a specialized chemistry module tailored for such needs. It enables a flexible but fast and efficient implementation of graph grammar based algorithms to solve chemical reaction related problems. In the following, we will present the features provided by the module and the available data structures and algorithms.

Molecules as Graphs Figure 1 illustrate the definition of molecules in terms of graphs and the relation of chemical reactions and graph grammar rules. In detail, a molecule is represented as a graph M where the node (l_V^M) and edge labels (l_E^M) are restricted to the existing atom and bond types, respectively. As suggested by Yadav *et al.* [49] and applied in literature [6, 35], we follow the SMILES encoding of atoms and bonds [46]. Therein, bond labels are confined to $l_E^M(..) \in \{$`-,=,#,:`$\}$, encoding for single, double, triple, or aromatic bonds respectively. Atom labels l_V^M are either one of the labels known from the periodic table of elements, e.g. `C` or `Br`, an organic atom participating in an aromatic ring (encoded by lower case symbols, e.g. `c` or `n`), or a "complex" label encoding additional charge information, e.g. `O-`.

Within the GGL chemistry module we enforce an explicit encoding of hydrogen information as atoms (see Fig. 1b)), even if the SMILES notation allows for their encoding within complex node labels. This was done a) to ensure explicit and complete chemical rule encodings, b) to enable sanity checks for molecules and rules, c) to allow for the prediction of aromaticity information, and d) to ensure an efficient matching of chemical rule patterns onto molecule graphs, which is needed for their application.

Based on our internal molecule graph representation, the GGL chemistry module features a couple of functionalities needed within the context of chemical reaction problems. This includes sanity checks for molecules (e.g. correct label usage, valence constraints, etc.), the SMILES string encoding (discussed within next section), and automatic hydrogen prediction. Furthermore, we have reimplemented the efficient free energy prediction algorithm introduced by Jankowski *et al.* [29]. It enables a fast energy approximation of a given molecule graph based on a decomposition into defined atomic groups and their energy contributions. Again, we take advantage of the fast subgraph matching module part of the GGL v4.0 to allow for a performant decomposition. The implementation and use of other energy estimation approaches is easily possible due to the modular design of the library.

The estimation of a molecule's energy usually requires knowledge about (hetero) aromatic rings within the molecule [38]. Since the property of aromaticity might emerge or vanish due to chemical reactions, we provide an aromaticity prediction framework. It is based on a ring perception using an extension of the fast algorithm suggested by Hanser *et al.* [24] in combination with a support vector machine learning approach for the aromaticity prediction based on the NSPDK graph feature kernel [13].

Canonical SMILES The SMILES notation was actually introduced to enable a string representation of chemical molecules, i.e. SMILES strings are a string encoding of molecule graphs [46]. Furthermore, they allow for a canonical, i.e. unique, representation of molecules [47]. This is especially useful when storing molecule information in databases or when molecules are to be given as program input (see [49]). The GGL features a full-fledged canonical SMILES writer implementation as well as an according SMILES parser to enable SMILES as

the chemical communication language for applications. Internally, molecules are represented as graphs as given above.

Chemical Reactions Chemical reactions are special graph grammar rules since they have certain essential side constraints to be fulfilled. First and most important: conservation of mass, i.e. no atom can appear or vanish. Thus, a chemical reaction is – as a coarse grained sketch – a "rewiring" of bonds within or between molecule graphs. Technically, we can encode chemical reactions as graph grammar rules in GML notation as introduced above. Therein, atom and node labels are restricted to the encoding supported by SMILES (see above), while wildcard labels are allowed to enable more general encodings. All chemical rules can be and are checked for their correctness within the GGL chemistry framework. Among the tests are checks for mass conservation, label use, or reasonable valence changes. Another important check is to ensure that no bond is formed twice, which is done by implicitely adding according "no-edge" constraints for all bonds formed within the chemical rule.

When applying chemical rules on molecule graphs we can use the generic graph grammar rule application framework described above. No adjustments for the rule applications are needed. Since some chemical rewrites might result in non-realistic molecules, e.g. due to steric constraints not covered by the rule [49], we provide a post-application verification step. Here, the output molecules are postprocessed, e.g. to correct the molecules aromaticity, and checked for sanity. A chemical rewrite only results in a chemical reaction, if all result molecules have been shown to be valid. The canonical SMILES encoding introduced above is used to derive a compact string representation of both the resulting molecules as well as the whole reaction.

Chemical reaction networks are often subject of reaction pathway analyses. For this purpose one has to know or estimate the reaction rates to be associated to individual chemical reactions e.g. as produced by a graph grammar rule application. The GGL chemistry framework fully supports such requirements. Based on the approach by Jankowski *et al.* [29], we estimate the energy difference ΔE of the input and output molecules of a reaction. This enables the estimate of the reaction rate using the well known Arrhenius law, i.e. the reaction rate is approximated by $\exp^{-\Delta E/RT}$ for a given temperature T using the gas constant R. Other approaches, e.g. the machine learning approach by Kayala *et al.* [31], can be easily integrated and used within the GGL chemistry framework if needed due to its modular architecture.

Molecular Group Specification The specification of (bio)chemical reactions often requires the representation of large (unchanged) parts of molecules in order to make the rule as specific as the chemical reaction. A classic example is the involvement of helper molecules like ATP, NADH, etc. that are only slightly changed but have to be represented completely to avoid the application of the rule using similar molecules.

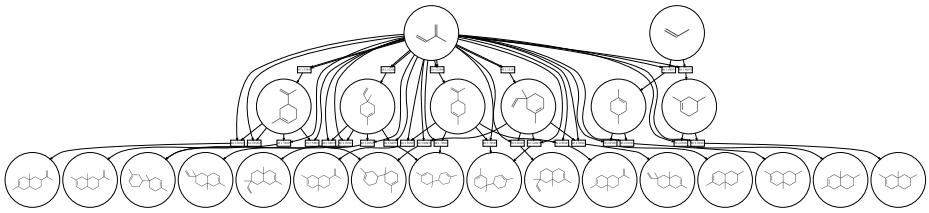


Fig. 3. The depiction of the Diels-Alder reaction network for the two input molecules depicted in Fig. 1d) (with SMILES `C=CC(C)=C` and `C=CC`) and the rule encoding from Fig. 2 using two rule application iterations.

To this end, the GGL supports the specification of molecular groups as pseudo-atoms within chemical rule definitions. They allow for a much easier and compact rule definition and avoid potential typos and mistakes. Section B of the suppl. material exemplifies the problem.

Visualization To enable an easier definition and evaluation of chemical reaction data, visualization scripts are provided. 2D-layouting of molecule graphs is done via the OpenBabel framework [33] and scalable vector graphics in SVG or PDF format are generated. For instance, given a valid GML encoding of a chemical rule, the script `chemrule2svg.pl` can be used to generate an according depiction. Figure S.1 (suppl. material) exemplifies the application for the GML rule encoding given in Fig. 2.

The GGL v4.0 package provides a reimplementation of the reaction network expansion approach presented in [6] named `toyChem`. Given a set of molecules and chemical reactions, `toyChem` expands the according reaction network via an iterative application of the reaction rules. Along the expansion, it automatically computes according reaction rates and produces a graph encoding of the reaction network. The script `printReactionNetwork.pl` visualizes the network including depictions of the molecules and rate information. For an example see Fig. 3.

Set of Enzymatic Reaction Data Provided For the application of the GGL to simulations of biochemical reactions within metabolic networks, 5133 unique GML-encoded graph grammar rules for enzymatic reactions are included in the GGL v4.0 distribution. The rules are derived from the KEGG LIGAND database [30], Release 58.1 June 2011.

For each enzyme listed in the database, a rewrite rule for each reaction annotated to that enzyme has been created automatically. To this end, an atom-to-atom correspondence between the substrate and product molecules has been determined to identify the broken and formed bonds along the given reaction. The atom mapping was generated using a greedy heuristic losely based on the Cut Successive Largest algorithm proposed in [14]. The predicted mappings follow the Principle of Minimal Chemical Distance, which states that the mech-

anism involving the least reconfigurations of valence electrons (i.e. edge additions/removals) is most likely the true reaction mechanism. Reactions that are unbalanced or contain compounds with missing or faulty molecular structure data cannot be mapped and are therefore not included. Since there is no data on reaction reversibility in the database, only the forward direction as given in the database is represented.

OpenBabel Port The GGL chemistry module is of course not the only available cheminformatics package on the market. It is neither intended nor designed to enable very sophisticated chemical informations as molecule mass or to provide specific tools like graphical depiction algorithms already available in other libraries. A powerful and freely available library is provided by the OpenBabel project [33] that was initially started to enable an easy conversion between the various chemical data formats. Nowadays, various tools and solutions are provided to tackle chemoinformatics problems.

Since we focus on a highly modular design of the GGL, we provide an easy port to convert our internal molecule graph representation into an OpenBabel object. This port can be used to get access to the full set of functionality provided by the OpenBabel library if needed [33].

2.5 Further Library Features

Beside the features introduced we want to give some further remarks on the GGL programming library. The object-oriented C++ source code is fully `ANSI`-conform and extensively documented. This enables the generation of the provided Application Programming Interface using the `doxygen` system. The dependency checking and compilation process is tailored for `Unix`-like systems (including `Cygwin` for MS Windows) and is based on the established `GNU autotools` resulting in an easy and automatic setup and installation of the libraries and tools. The GGL v4.0 comes with an extensive test set framework to ensure the correctness of the build on the used platform and to ensure the stability and maintainability of the library. In addition, we provide `Perl-5` bindings to enable the application of the GGL functionalities within fast prototypical `Perl` developments, e.g. in combination with the `PerlMol` package [44].

Last but not least, the GGL v4.0 package is freely available at:

http://www.tbi.univie.ac.at/software/GGL/

3 Application and Examples

The graph grammar library is designed to support a wide range of graph rewrite systems on any type of graph. For illustration, we have implemented a few example applications that cover different aspects of graph rewrite. All are distributed within the GGL v4.0 package and described in Sec. C of the suppl. material.

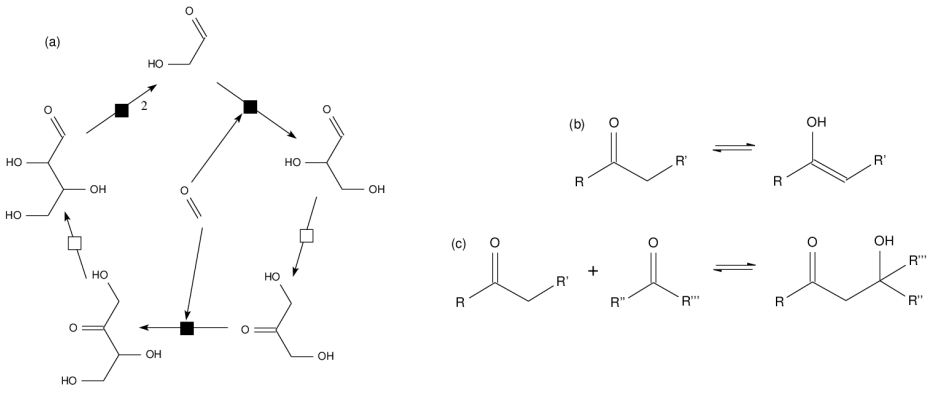


Fig. 4. Autocatalytic core network (a) of the formose process, keto-enol tautomerization of a carbonyl compounds (b), and aldol addition (c) between two carbonyl compounds. Depending on the reaction direction the aldol addition forms or breaks a `C`–`C` bond (■). In the case of α-hydroxy-carbonyl compounds (`HO`–`C`–`C`=`O`), the keto-enol tautomerization provides a mechanism for the carbonyl functionality (`C`=`O`) to shift along the backbone of the sugar (□).

Since the GGL v4.0 supports an extensive module for chemical graph rewrite systems, we will focus on chemical applications in the following.

The graph grammar approach to chemical transformation gives a very compact description of a whole "chemical universe", i.e. the language of all chemical graphs, which are reachable from an initial set of chemical "starting graphs" by iterative application of the reaction rules. The iterative expansion of a particular graph grammar yields a directed, potentially infinit reaction network, where the chemical graphs are connected by hyperedges representing the reaction rules. These reaction networks can be further analysed statistically or with methodologies from network theory to uncover unexpected relations between network nodes and their properties [22, 41]. The `toyChem` utility distributed with the GGL implements only simple exhaustive iterative expansion of a chemical universe. For the efficient exploration of a chemical universe a sofisticated strategy framework is required (see [1]) to avoid combinatorial explosion. The GGL is a major improvement over the prototypical implementation presented in [6]. Besides the extention to the full chemical atom set GGL implements an energy increment system as well as a rate calculation approach for chemical reactions. Furthermore GGL provides an interface to the OpenBabel library and therefore the whole functionality of this important open source cheminformatics library can be harnessed from within the GGL.

On the basis of the formose process [8], we illustrate, that the graph grammar approach is indeed a sensible model of chemical reaction networks and nicely captures the algebraic properties of chemistry itself. The formose process condenses formaldehyde, the simplest possible sugar, into a combinatorial complex

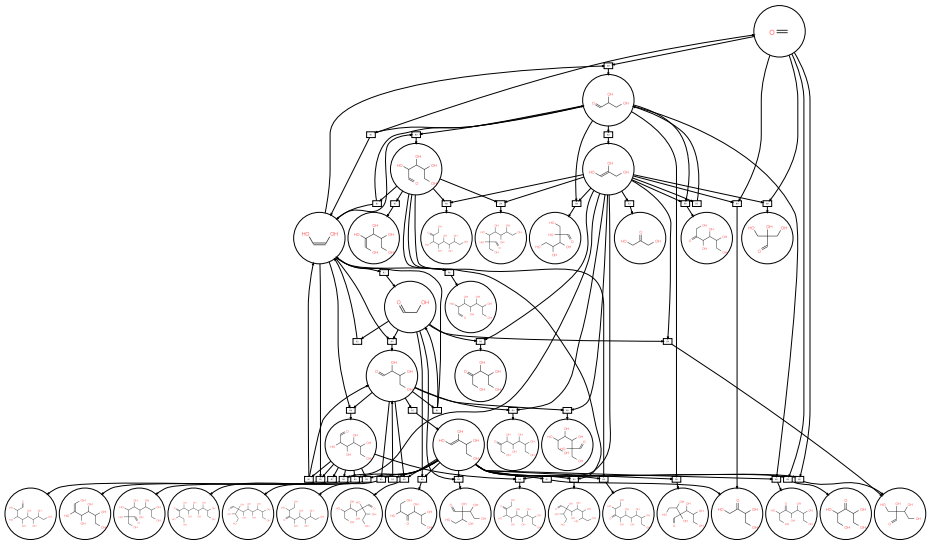


Fig. 5. Hypergraph of the resulting reaction network after 4 iterations of the formose process grammar.

mixture of higher sugars by repeatedly involving only two reversible reactions, the aldol reaction and the keto-enol tautomerization (see Fig. 4 b, c). The corresponding graph grammar rule encodings are depicted in Fig. S.5 and S.6 (suppl. material). An autocatalytic loop, which produces glycolaldehyd and consumes formaldehyde, is located at the core of the formose process (see Fig. 4 a). Figure 5 depicts the growing reaction network of the formose process.

The following table 1 shows the exponential explosion of the molecular space for the formose process and the according runtime of our `toyChem` tool. Note, the vast majority of the runtime is consumed by canonical SMILES generation for the molecule graphs resulting from graph grammar rule applications. Still this step is essential to distinguish new from already known molecules and it is much faster than graph isomorphism based comparisons.

Iteration	1	2	3	4	5	6
Molecules	3	5	9	37	302	10,572
Time	0	0	0	0	$0.2s$	$10.5s$

Table 1. Exponential explosion of the molecular space for the formose process starting from `OCC=O` and `C=O` along with the runtimes of `toyChem`.

The second example shows the reaction network for the enzyme mechanism of β-lactamase (see Fig. S.7) as found in the MACiE database [27] entry M0002. Specificity of the amino acide side chains of Lys, Ser and Glu in the active site of the enzyme EC 3.5.2.6 was achieved by marking the C_α atom labels of the aminoacides with a SMILES class flag (see Fig. S.8 suppl. material). In that way reactions between amino acide side-chains are suppressed within the graph grammar rule application. Figure S.9 (suppl. material) depicts the first step within the enzyme mechanism.

4 Summary

The Graph Grammar Library (GGL) is a powerful framework for cheminformatics applications based on graph rewrite. It meets very well the mandatory requirements for such studies as discussed by Yadav *et al.* [49] and comes with a powerful chemistry module providing essential algorithms. With the advent of genome-scale metabolic networks, formalisms such as the GGL to handle chemical transformation will become an important factor for the analysis, interpretation, and manipulation of these networks.

Acknowledgements This work was supported in part by the Vienna Science and Technology Fund (WWTF) proj. no. MA07-030, the Austrian GEN-AU project "Bioinformatics Integration Network III", and the COST Action CM0703 "Systems Chemistry". Furthermore, we thank the users of the GGL framework for their ongoing contributions, bug reports and suggestions.

References

1. Andersen, J., Flamm, C., Merkle, D., Stadler, P.F.: Generic strategies for chemical space exploration. In: 24th International Conference on Rewriting Techniques and Applications. Eindhoven, The Netherlands (2013), submitted to RTA 2013
2. Andersen, J.L., Flamm, C., Merkle, D., Stadler, P.F.: Maximizing output and recognizing autocatalysis in chemical reaction networks is np-complete. J. Syst. Chem. (2011)
3. Archdeacon, D., Colbourn, C.J., Gitler, I., Provan, J.S.: Four-terminal reducibility and projective-planar wye-delta-wye-reducible graphs. J. Graph Theory 33, 83–93 (1998)
4. Baran, M., Staton, E.: Distribution transformer models for branch current based feeder analysis. Power Systems, IEEE Transactions on 12(2), 698–703 (1997)
5. Beck, M., Benkö, G., Eble, G., Flamm, C., Müller, S., Stadler, P.F.: Graph grammars as models for the evolution of developmental pathways. In: Proc. of GWAL'04. pp. 8–15 (2004)
6. Benkö, G., Flamm, C., Stadler, P.F.: A graph-based toy model of chemistry. J Chem. Inf. and Comp. Sci. 43(4), 1085–1093 (2003)
7. Blinov, M., Yang, J., Faeder, J., Hlavacek, W.: Graph theory for rule-based modeling of biochemical networks. In: Transactions on Computational Systems Biology VII, LNCS, vol. 4230, pp. 89–106. Springer (2006)

8. Butlerov, A.M.: Einiges über die chemische structur der körper. Zeitschrift für Chemie 4, 549–560 (1861)
9. Colvin, J., Monine, M., Gutenkunst, R., Hlavacek, W., Von Hoff, D., Posner, R.: RuleMonkey: software for stochastic simulation of rule-based models. BMC Bioinformatics 11(1), 404 (2010)
10. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: Performance evaluation of the VF graph matching algorithm. In: Proc. of ICIAP'99. p. 1172 (1999)
11. Cordella, L., Foggia, P., Sansone, C., Vento, M.: A (sub)graph isomorphism algorithm for matching large graphs. Pattern Analysis and Machine Intelligence, IEEE Transactions on 26(10), 1367–1372 (2004)
12. Corradini, A., , Montanari, U., , Rossi, F., Ehrig, H., Heckel, R., Loewe, M.: Algebraic approaches to graph transformation part I: Basic concepts and double pushout approach. In: Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations. vol. 97, pp. 163–245 (1997)
13. Costa, F., De Grave, K.: Fast neighborhood subgraph pairwise distance kernel. In: Proc. of ICML'10. pp. 255–262 (2010)
14. Crabtree, J.D., Mehta, D.P.: Automated reaction mapping. J. Exp. Algo. 13, 1.15–1.29 (2009)
15. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Rule-based modelling of cellular signalling. In: Proc. of Concurrency Theory (CONCUR). pp. 17–41 (2007)
16. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Rule-based modelling, symmetries, refinements. In: Proc. of Formal Methods in Systems Biology (FMSB). pp. 103–122 (2008)
17. Dörr, H.: Efficient Graph Rewriting and its Implementation. LNCS, Springer (1995)
18. Eppstein, D.: Subgraph isomorphism in planar graphs and related problems. In: Proc. of SODA'95. pp. 632–640 (1995)
19. Flamm, C., Ullrich, A., Ekker, H., Mann, M., Hoegerl, D., Rohrschneider, M., Sauer, S., Scheuermann, G., Klemm, K., Hofacker, I.L., Stadler, P.F.: Evolution of metabolic networks: A computational framework. J Syst. Chem. 1(1), 4 (2010)
20. Gardner, M.: The fantastic combinations of John Conway's new solitaire game "life". Scientific American 223, 120–123 (1970)
21. Geiss, R., Batz, G., Grund, D., Hack, S., Szalkowski, A.: GrGen: A fast SPO-based graph rewriting tool. In: Proc. of ICGT'06), LNCS, vol. 4178, pp. 383–397. Springer (2006)
22. Grzybowski, B.A., Bishop, K.J.M., Kowalczyk, B., Wilmer, C.E.: The ẃiredúniverse of organic chemistry. Nature Chemistry 1, 31–36 (2009)
23. Guzman, J.d., Nuffer, D.: The Spirit Library: Inline parsing in C++. C/C++ Users Journal 21(9), 22 (2003)
24. Hanser, T., Jauffret, P., Kaufmann, G.: A new algorithm for exhaustive ring perception in a molecular graph. J. Chem. Inf. Comp. Sci. 36(6), 1146–1152 (1996)
25. Himsolt, M.: GML: A portable graph file format. Tech. rep., University of Passau, Germany (1999)
26. Hlavacek, W.S., Faeder, J.R., Blinov, M.L., Posner, R.G., Hucka, M., Fontana, W.: Rules for modeling signal-transduction systems. Sci. STKE 2006(344), re6 (2006)
27. Holliday, G.L., Andreinj, C., Fischer, J., Rahman, S.A., Almonacid, D.E., Williams, S.T., Pearson, W.R.: MACiE: exploring the diversity of biochemical reactions. Nuc. Acids Res. 40, D783–D789 (2012)
28. Jakumeit, E., Buchwald, S., Kroll, M.: GrGen.NET. STTT J. 12(3), 263–271 (2010)

29. Jankowski, M.D., Henry, C.S., Broadbelt, L.J., Hatzimanikatis, V.: Group contribution method for thermodynamic analysis of complex metabolic networks. Biophys. J. 95(3), 1487–1499 (2008)
30. Kanehisa, M., Goto, S., Sato, Y., Furumichi, M., Tanabe, M.: KEGG for integration and interpretation of large-scale molecular data sets. Nuc. Acids Res. (2011)
31. Kayala, M.A., Azencott, C.A., Chen, J.H., Baldi, P.: Learning to predict chemical reactions. J. Chem. Inf. and Modeling 51(9), 2209–2222 (2011)
32. Lynce, I., Ouaknine, J.: Sudoku as a SAT problem. In: Proc. of AIMATH'06. p. 9 (2006)
33. O'Boyle, N.M., Banck, M., James, C.A., Morley, C., Vandermeersch, T., Hutchison, G.R.: Open Babel: An open chemical toolbox. J. Cheminf. 3(1), 33+ (2011)
34. Roos-Kozel, B.L., Jorgensen, W.L.: Computer-assisted mechanistic evaluation of organic reactions. 2. perception of rings, aromaticity, and tautomers. J. Chem. Inf. Comp. Sci. 21, 101–111 (1981)
35. Rosselló, F., Valiente, G.: Chemical graphs, chemical reaction graphs, and chemical graph transformation. Electron. Notes Theor. Comput. Sci. 127, 157–166 (2005)
36. Rozenberg, G. (ed.): Handbook of graph grammars and computing by graph transformation: volume I. foundations. World Scientific Publishing Co., Inc. (1997)
37. Rudolf, M.: Utilizing constraint satisfaction techniques for efficient graph pattern matching. In: Theory and Application of Graph Transformations, LNCS, vol. 1764, pp. 381–394. Springer (2000)
38. Schleyer, P.v.R., Jiao, H.: What is aromaticity? Pure and Applied Chem. 68(2), 209–218 (1996)
39. Siek, J.G., Lee, L.Q., Lumsdaine, A.: The Boost Graph Library: User Guide and Reference Manual (C++ In-Depth Series). Addison-Wesley Professional (2001)
40. Simonis, H.: Sudoku as a constraint problem. In: CP Workshop on Modeling and Reformulating CSPs. pp. 13–27 (2005), http://www.icparc.ic.ac.uk/∼hs/sudoku.pdf
41. Soh, S., Wei, Y., Kowalczyk, B., Gothard, C.M., Baytekin, B., Gothard, N., Grzybowski, B.A.: Estimating chemical reactivity and cross-influence from collective chemical knowledge. Chemical Science 3(5), 1497–1502 (2012)
42. Taentzer, G.: AGG: A tool environment for algebraic graph transformation. In: Applications of Graph Transformations with Industrial Relevance, LNCS, vol. 1779, pp. 333–341. Springer (2000)
43. Taentzer, G.: AGG: A graph transformation environment for modeling and validation of software. In: Applications of Graph Transformations with Industrial Relevance, LNCS, vol. 3062, pp. 446–453. Springer (2004)
44. Tubert-Brohman, I.: Perl and chemistry. The Perl Journal 8, 3–5 (2004)
45. Varró, G., Friedl, K., Varró, D.: Adaptive graph pattern matching for model transformations using model-sensitive search plans. Electron. Notes Theor. Comput. Sci. 152, 191–205 (2006)
46. Weininger, D.: SMILES, a chemical language and information system. 1. introduction to methodology and encoding rules. J. Chem. Inf. Comp. Sci. 28(1), 31–36 (1988)
47. Weininger, D.: SMILES, a chemical language and information system. 2. algorithm for generation of unique SMILES notation. J. Chem. Inf. Comp. Sci. 29(2), 97–101 (1989)
48. Xu, W., Smith, A.M., Faeder, J.R., Marai, G.E.: RuleBender: a visual interface for rule-based modeling. Bioinformatics 27(12), 1721–1722 (2011)
49. Yadav, M.K., Kelley, B.P., Silverman, S.M.: The potential of a chemical graph transformation system. In: ICGT. LNCS, vol. 3256, pp. 83–95. Springer (2004)

Supplementary Material

A Visualization

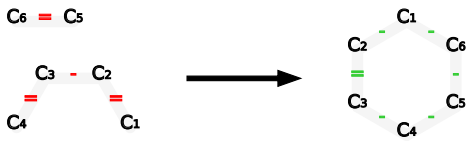


Fig. S.1. The depiction of the Diels-Alder GML graph grammar rule encoding using the provided visualization script `chemrule2svg.pl`. The presentation highlights the altered bonds within the left and right side graph of the reaction in red and green, resp., to enable the identification of the underlying reaction mechanism.

B Molecular Groups

One field of application for the definition and use of molecular groups is the specification of molecules that differ only in a few atoms or bonds. In such cases, it can be convenient to specify only the dissimilar parts of the molecules and to use group placeholders for the equal parts. That way, the similarity becomes easy to see and the SMILES easier to read.

As an example, we use the molecules NADH and NAD^+ depicted in Fig. S.2 sporting 66 and 65 atoms, respectively. The difference basically comprises only an additional proton within NADH and a charge change within the lower ring while the rest of the molecules are identically. Note, these two changes alter the ring from non-aromatic (NADH) to aromatic (NAD^+).

Minimal SMILES encodings of the molecules (highlighting the differing ring in red) are

```
NC(=O)C1[CH2]C=CN(C=1)C2OC(COP(O)(=O)O..
    ..P(O)(=O)OCC3OC(C(O)C3O)n4cnc5c(N)ncnc54)C(O)C2O
```

for NADH and

```
NC(=O)c1ccc[n+](c1)C2OC(COP(O)(=O)O..
    ..P(O)(=O)OCC3OC(C(O)C3O)n4cnc5c(N)ncnc54)C(O)C2O
```

for NAD^+.

In contrast, when using group declarations for the identical parts, namely the CONH2 group and the ribo-adenosine, the SMILES shrinks to

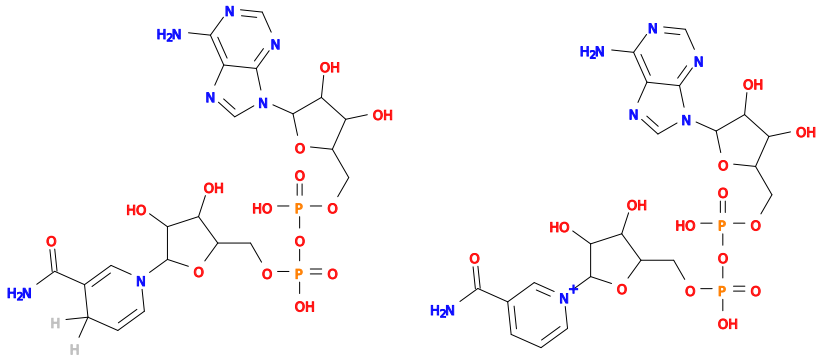


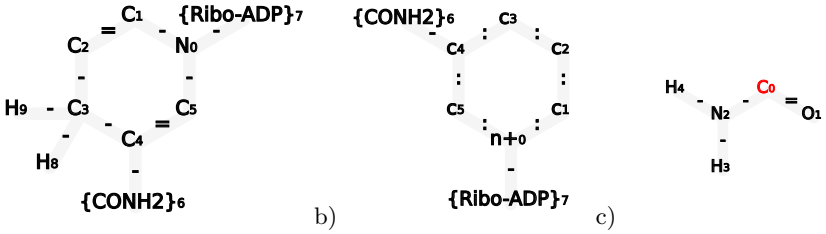
Fig. S.2. The molecules NADH (left) and NAD^+ (right).

`[{CONH2}]C1[CH2]C=CN(C=1)[{Ribo-ADP}]`

for NADH and

`[{CONH2}]c1ccc[n+](c1)[{Ribo-ADP}]`

for NAD^+, both depicted in Fig. S.3a) and b).



a) b) c)

Fig. S.3. The molecules NADH (a) and NAD^+ (b) described using group identifiers. Note, only for C_3 within NADH explicit proton information is given; all other carbons have one not depicted adjacent proton. (c) Depiction of the {CONH2} group using the script `molcomp2svg.pl`. Note, the proxy node is highlighted in red.

C General Graph Rewrite Examples

C.1 Game of Life

In the 60th, John Conway created a cellular automaton named *Game of Life* [20] that reflects the basic principles of birth, death, and survival within populations.

Each cell represents an individual that is either alive or dead depending on the state of the neighbored individuals.

The problem can be represented as a graph relabeling problem where each node represents a cell and edges connect neighbored cells. The formulation of the birth, death, and survival rules defined by Conway as graph grammar rules is straight forwardly described by graph grammar rules. Each of the three rules resembles a single node and according adjacency constraints for the matching and the according recoloring of the node in match case. In combination with an exhaustive application of the rules, this results in a complete Game of Life solver based on a graph rewrite system distributed with the package.

C.2 Sudoku

Sudoku is a combinatorial problem where a given 9×9 grid has to be filled with numbers from 1 to 9, such that each number appears only once in each row, column, and defined 3×3 sub-grids. The task is to fill a given partially filled grid such that all constraints are fullfilled [32, 40].

Similar to Game of Life, one can encode the Sudoku grid within a graph where each cell is represented by a node. Each node is connected to all other cells that have to have a different label. Thus, it resembles the dependency graph of the problem. In order to find a solution, a Depths-First-Search (DFS) of the exponential search space can be applied, where each search step is defined by the valid application of a graph grammar rule. Each rule does the assignment of a number to a non-assigned node while respecting all constraints.

The GGL v4.0 package features a generic DFS implementation for such purposes. Given a set of graph grammar rules and a start graph, a DFS exploration of the search space is done and solutions are identified.

C.3 Ring Perception

The enumeration of all rings within a graph can be done using the algorithm proposed by Hanser *et al.* [24]. Such ring perceptions are important e.g. in chemistry to do structure classifications or aromaticity identification [34]. The algorithm by Hanser *et al.* creates an image of the studied graph that represents the node adjacency within its edge labels, a so called *path graph*. This path graph is progressively collapsed in a way that the final path graph contains only loops, each representing a ring from the original graph.

We have implemented this general approach for ring perception based on a small set of graph grammar rules and a simple rule application iteration. This shows once more the expressivity of graph rewrite systems. A rule-based example as well as an efficient native C++ implementation of the ring perception algorithm is part of GGL v4.0 package.

C.4 Y-Δ-equivalence problem

The Y-Δ-equivalence problem, also known by name wye-delta or delta-wye, star-delta, star-mesh, or T-Π, is an important problem from graph theory [3] with

application in electrical resistor network optimization [4]. In short, two graphs are defined to be Y-Δ-equivalent if and only if they can be transformed into each other by applying a series of Y-Δ-transformations. These transformations are either to convert a Δ triangle subgraph into a `Y`-like subgraph (by adding a new central node in combination with the necessary rewiring) or the reverse operation, i.e. transforming a `Y`-subgraph into a triangle (via center deletion and rewiring). For instance, this property is fulfill by the Peterson graph family forming a Y-Δ-equivalence class [3].

Using two simple graph grammar rules that encode the allowed Y-Δ-transformations, we can easily setup a search engine to check if two graphs are Y-Δ-equivalent or not. To this end, one starts two independent Breadth-First-Searches (BFS) starting from the two graphs of interest. Within each BFS all graphs are generated that can be obtained from the start graph by applying Y-Δ-transformations. The Y-Δ-equivalence is proven as soon as the two sets of graphs produced by the independent BFS intersect. The according graph grammar rules in GML notation are given in Fig. S.4.

```
rule [
  ruleID "wye to delta"
  wildcard "*"
  context [
    node [ id 1 label "*" ]
    node [ id 2 label "*" ]
    node [ id 3 label "*" ]
  ]
  left [
    node [ id 4 label "*" ]
    edge [ source 4 target 1 label "*" ]
    edge [ source 4 target 2 label "*" ]
    edge [ source 4 target 3 label "*" ]
    constrainAdj [
      id 4 op = count 3
      edgeLabels [ label "*" ]
      nodeLabels [ label "*" ]
    ]
    constrainNoEdge [ source 1 target 2 ]
    constrainNoEdge [ source 1 target 3 ]
    constrainNoEdge [ source 2 target 3 ]
  ]
  right [
    edge [ source 1 target 2 label "*" ]
    edge [ source 1 target 3 label "*" ]
    edge [ source 2 target 3 label "*" ]
  ]
]
```

Fig. S.4. The GML rule encoding "wye to delta" graph grammar rule used for Y-Δ transformations. The reverse "delta to wye" rule can be easily obtained by exchanging the `left` and `right` content of the rule, while all constraints can be omitted.

D Chemical Reactions as Graph Grammars

D.1 Formose Process

```
rule [
  ruleID "Keto-Enol Isomerization"
  context [
    node [ id 1 label "C" ]
    node [ id 2 label "C" ]
    node [ id 3 label "O" ]
    node [ id 4 label "H" ]
  ]
  left [
    edge [ source 1 target 4 label "-" ]
    edge [ source 1 target 2 label "-" ]
    edge [ source 2 target 3 label "=" ]
    constrainAdj [ id 2 op = count 1 nodeLabels [ label "O" ] ]
  ]
  right [
    edge [ source 1 target 2 label "=" ]
    edge [ source 2 target 3 label "-" ]
    edge [ source 3 target 4 label "-" ]
  ]
]
```

Fig. S.5. The GML rule encoding the "keto-enol tautomerization" graph grammar rule used. The reverse rule can be easily obtained by exchanging the `left` and `right` content of the rule, while all constraint can be omitted.

```
rule [
  ruleID "Aldol Condensation"
  context [
    node [ id 1 label "C" ]
    node [ id 2 label "C" ]
    node [ id 3 label "O" ]
    node [ id 4 label "H" ]
    node [ id 5 label "O" ]
    node [ id 6 label "C" ]
  ]
  left [
    edge [ source 1 target 2 label "=" ]
    edge [ source 2 target 3 label "-" ]
    edge [ source 3 target 4 label "-" ]
    edge [ source 5 target 6 label "=" ]
    constrainAdj [ id 2 op = count 1 nodeLabels [ label "O" ] ]
    constrainAdj [ id 6 op = count 1 nodeLabels [ label "O" ] ]
  ]
  right [
    edge [ source 1 target 2 label "-" ]
    edge [ source 2 target 3 label "=" ]
     edge [ source 5 target 6 label "-" ]
    edge [ source 4 target 5 label "-" ]
    edge [ source 6 target 1 label "-" ]
  ]
]
```

Fig. S.6. The GML rule encoding the "aldol condensation" graph grammar rule. The reverse rule for the retro-aldol can be easily obtained by exchanging the `left` and `right` content of the rule, while all constraint can be omitted.

D.2 β-Lactamase Enzyme Mechanism

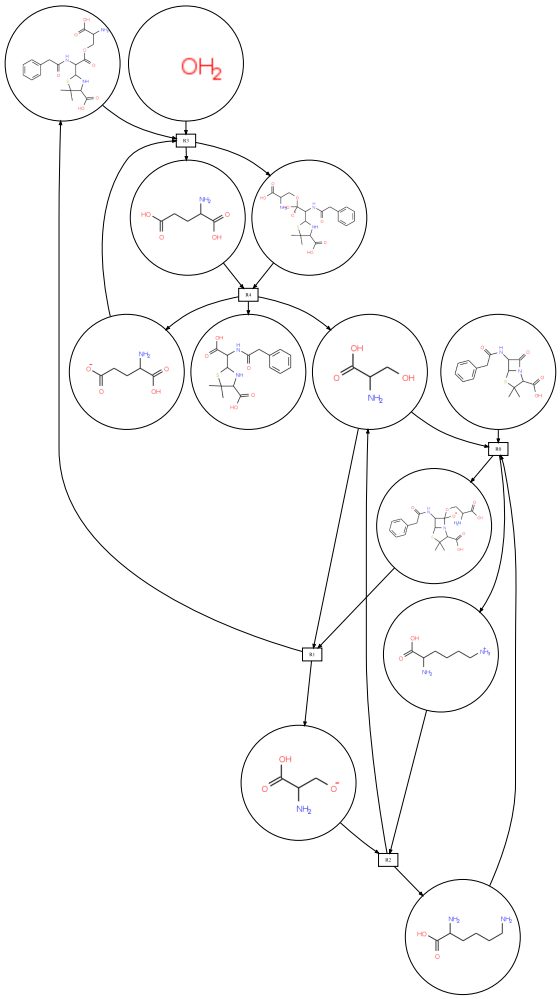


Fig. S.7. Hypergraph of the enzyme mechanism for β-lactamase (EC 3.5.2.6).

```
# Beta-lactamase (class A)
# MACiE version: 3
# MACiE-entry: M0002, 3.5.2.6, Step 01
rule [
 ruleID "3.5.2.6-M0002-S01"
 left [
  node [ id 8 label "N" ]
  node [ id 5 label "O" ]
  edge [ source 1 target 5 label "=" ]
  edge [ source 6 target 7 label "-" ]
 ]
 context [
  node [ id 1 label "C" ]
  node [ id 2 label "C" ]
  node [ id 3 label "C" ]
  node [ id 4 label "N" ]
  node [ id 6 label "O" ]
  node [ id 7 label "H" ]
  node [ id 9 label "H" ]
  node [ id 10 label "H" ]
  node [ id 100 label "*"]
  node [ id 101 label "*"]
  edge [ source 1 target 2 label "-" ]
  edge [ source 2 target 3 label "-" ]
  edge [ source 3 target 4 label "-" ]
  edge [ source 4 target 1 label "-" ]
  edge [ source 8 target 9 label "-" ]
  edge [ source 8 target 10 label "-" ]
  edge [ source 6 target 100 label "-" ]
  edge [ source 8 target 101 label "-" ]
  constrainNode [ id 100 op = nodeLabels [ label "C:1" ] ]
  constrainAdj [ id 100 op = count 1 nodeLabels [ label "O" ] ]
  constrainNode [ id 101 op = nodeLabels [ label "C:1" ] ]
 ]
 right [
   node [ id 8 label "N+" ]
   node [ id 5 label "O-" ]
   edge [ source 1 target 5 label "-" ]
   edge [ source 1 target 6 label "-" ]
   edge [ source 8 target 7 label "-" ]
 ]
]
```

Fig. S.8. Rewrite Rule in the enzyme mechanism of β-lactamase (EC 3.5.2.6). The distinction between enzyme and the substrate(s) is achieved by using `SMILES` class number for atom labels belonging to amino acid side chains of the enzyme.

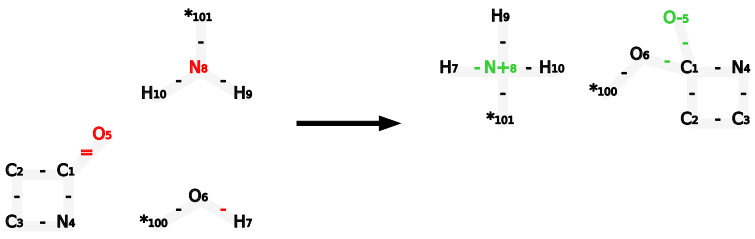


Fig. S.9. First step in the enzyme mechanism of β-lactamase (EC 3.5.2.6)