MATCH
Communications in Mathematical and in Computer Chemistry

MATCH Commun. Math. Comput. Chem. **80** (2018) 661-704

ISSN 0340 - 6253

Rule Composition in Graph Transformation Models of Chemical Reactions

Jakob L. Andersen[1–4], Christoph Flamm[4,10], Daniel Merkle[2,*], Peter F. Stadler[4–9]

[1] *Earth-Life Science Institute, Tokyo Institute of Technology, Tokyo 152-8550, Japan*

[2] *Department of Mathematics and Computer Science, University of Southern Denmark, Odense M DK-5230, Denmark*

[3] *Research group Bioinformatics and Computational Biology, Faculty of Computer Science, University of Vienna, Wien A-1090, Austria*

[4] *Institute for Theoretical Chemistry, University of Vienna, Wien A-1090, Austria*

[5] *Bioinformatics Group, Department of Computer Science, and Interdisciplinary Center for Bioinformatics, and German Centre for Integrative Biodiversity Research (iDiv) Halle-Jena-Leipzig and Competence Center for Scalable Data Services and Solutions Dresden-Leipzig and Leipzig Research Center for Civilization Diseases, University of Leipzig, Leipzig D-04107, Germany*

[6] *Max Planck Institute for Mathematics in the Sciences, Leipzig D-04103, Germany*

[7] *Fraunhofer Institute for Cell Therapy and Immunology, Leipzig D-04103, Germany*

[8] *Center for non-coding RNA in Technology and Health, University of Copenhagen, Frederiksberg C DK-1870, Denmark*

[9] *Santa Fe Institute, 1399 Hyde Park Rd, Santa Fe NM 87501, USA*

[10] *Research Network Chemistry Meets Microbiology, University of Vienna, Wien A-1090, Austria*

∗ daniel@imada.sdu.dk

(Received May 29, 2017)

Abstract

Graph transformation form a natural model for chemical reaction systems and provide a sufficient level of detail to track individual atoms. Among alternative graph transformation formalisms the Double Pushout approach, which is firmly grounded in category theory, is particularly well-suited as a model of chemistry. We explore here the formal foundations of defining composition of transformation rules using ideas from concurrency theory. In addition of a generic construction we consider several special cases that each have an intuitive chemical interpretation. We

illustrate the usefulness of these specialised operations by automatically calculating coarse-grained transformation rules for complete chemical pathways, that preserve the traces of atoms through the pathways. This type of computation has direct practical relevance for the analysis and design of isotope labelling experiments.

1 Introduction

Chemical reactions are transformations of one set of chemical substances into another that leave the atoms unchanged but change chemical bonds between them. At the level of abstraction that is most commonly employed in the field, the salient information about a chemical reaction is expressed as a transformation of structural formulas. In mathematical terms, these are labelled graphs. Chemical reactions are not at all arbitrary rearrangements of atoms. Instead, they can be classified into a limited number of reaction mechanisms that differ in the patterns of chemical bonds that are rearranged, known as *imaginary transition states* [1]. The so-called "named reactions" specify the most important classes of reactions in organic chemistry.

All chemical reactions can be understood as a sequence of one or more *elementary reactions*. These occur in a single step, with a single transition state, and kinetically follow the law of mass action. The pattern of broken and newly formed bonds forms a simple cycle for elementary reactions. Non-elementary (multi-step) reactions thus are simply a juxtaposition of elementary reactions.

Classes of chemical reactions are determined not only by their associated imaginary transition states. Typically they are restricted to particular classes of substances that are determined by functional groups in the vicinity of the reaction centre. Taken together this implies that chemical reactions can be understood as rule-based transformations of molecular graphs. The reaction rules combine two ingredients: a specific pattern of bond changes determined by the imaginary transition state, and a precondition on the structure(s) of substrate molecules. The latter can be expressed as the requirement for a particular pattern/subgraph to be present in the substrates. From the mathematical perspective, this suggests to model chemical reactions by means of graph transformations acting on the graph representations of molecules, i.e., their structural formulae [2].

Interestingly, the intuitive view of chemistry as a transformation system has inspired models of concurrent computation already in 1990s; a prime example is the "Chemical Abstract Machine" [3]. Around the same time models of artificial chemistries [4] have

appeared that use rewriting-style calculi borrowed from theoretical computer science. Abstract rule-based systems have also been used to formalise an essentially computational view of biological processes [5, 6]. Kappa [7] and BioNetGen [8] were designed to model the behaviour of mixtures of "agents", typically interacting proteins. "Membrane computation" [9] and brane calculus [10] focus on compartmentalisation and the relative placement of cellular components. The computational aspects of epigenetics have been abstracted to a term-rewriting-like model [11]. For reviews see also [12, 13].

Concrete models of chemistry in terms of graph transformation systems have appeared only after the turn of the millennium [2], and their systematic investigation is even younger. Labelled graphs and their transformations provide a unique level of description for chemistry. It not only largely matches the chemist's intuition but also makes it easy and transparent to incorporate fundamental properties of chemical reactions, such as the conservation of mass, atom types, and charge. Several mathematically distinct constructions of graph transformation systems have been explored in the computer science literature, see e.g. [14] for a general overview and [15, 16] for details on algebraic graph transformations. We advocate the double pushout (DPO) framework as particularly suitable for modelling chemistry because it guarantees that reactions are reversible in principle and ensures that atom maps, i.e., the bijections between atoms in the educts and products, are always well defined. This is a key advantage in application to large reaction networks because it makes it possible to follow individual atoms through complex reaction networks. In contrast to network-centric approaches, the representation of chemical reactions based on transformation rules is therefore sufficiently detailed to support the analysis of isotope-label experiments [17, 18]. In particular, the consistent handling of atom maps is indispensable when model reduction techniques are to be applied to genome-scale metabolic network reconstructions [19, 20].

A key feature in rule-based calculi is the concept of rule composition. It allows, in particular, different levels of coarse graining in the description of a system's trajectories by contracting transitions between states in a principled manner, explored in some detail for Kappa in [21]. Chemical reactions can be readily composed to "overall reactions" such as the net transformation of metabolic pathways. This observation is used implicitly in flux balance analysis at the level of the stoichiometric matrix. Recently, [22] considered the composition of concrete chemical reactions, i.e., transformations of complete molecules,

as a means of reconstructing metabolic pathways. Here we take the complementary point of view: instead of asking for concrete overall reactions, we focus on the composition of the underlying *reaction mechanisms* themselves [23].

The purpose of this contribution is to provide a comprehensive formal exposition of rule composition in the DPO graph transformation framework applied to chemical reaction rules and its practical implementation in the software package MØD. Many examples, and a much less formal presentation of the ideas can be found in [23, 24]. Here we add a more detailed description of the underlying category theory as well as an exposition of the core algorithm.

We start with a brief introduction of DPO graph transformation, and proceed to discussing the concept of rule composition in depth, Sec. 3, in the general setting of DPO graph transformation. The specialisation of the formalism to the setting of chemistry is the topic of Sec. 4. In Sec. 5 we discuss considerations for a practical implementation of rules and rule composition. Finally we present two application examples in depth, and conclude with a brief overview of open questions and ongoing developments in both the underlying theory and the practical usage of rule composition.

2 The Double Pushout Approach to Graph Transformation

2.1 Introduction to Category Theory

Several concepts from category theory, and specifically pertaining to the category of graphs, are needed to formally define the DPO approach. The fundamental notion is that of a *morphism*, that is, a binary relation between graphs. Here we focus on undirected graphs without loops or parallel edges. For simplicitly of presentation we ignore here the labels vertices and edges, which in the context of chemistry correspond to atom and bond types, respectively. All definitions can trivially be extended to graphs with vertex and edge labels. Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two graphs.

Definition 2.1 (Graph Morphisms)**.** A *graph (homo)morphism* $m\colon G \to H$ is a structure-preserving mapping of vertices and edges. That is, if $e = (u, v) \in E_G$ then $m(e) = (m(u), m(v)) \in E_H$. Furthermore (see also Fig. 1)

- m is a *monomorphism* if it is injective: $\forall u, v \in V_G, u \neq v \Rightarrow m(u) \neq m(v)$. When a monomorphism exists we may simply write it as $G \subseteq H$ or in the reverse order

$H \supseteq G$.

- m is a *subgraph isomorphism* if m is a monomorphism and $(u, v) \in E_G \Leftrightarrow (m(u), m(v)) \in E_H$.

- m is an *isomorphism* if it is a subgraph isomorphism, and is a bijection of the vertices. When an isomorphism exists we say G and H are *isomorphic* and write it as $G \cong H$.

(a) A morphism. **(b)** A monomorphism. **(c)** A subgraph isomorphism. **(d)** An isomorphism.

Figure 1. Examples of the graph morphisms from Def. 2.1, visualised as explicit vertex mappings in red. a a morphism which is not a monomorphism, b a monomorphism which is not a subgraph isomorphism, c a subgraph isomorphism which is not an isomorphism, and d an isomorphism.

Morphisms form the basis of *categories*. Here we only state the definitions and aim at an intuitive understanding of the constructions in categories of graphs. For the full details we refer to [15, Appendix A].

Definition 2.2 (Category, see [15, Definition A.1])**.** A category $\mathbf{C}$ consists of

- a class of objects $Ob(\mathbf{C})$,

- a class of morphisms $Mor(\mathbf{C})$ where each morphism $f\colon A \to B$ maps $A \in Ob(\mathbf{C})$ to $B \in Ob(\mathbf{C})$, and

- a morphism composition operator $\circ\colon Mor(\mathbf{C}) \times Mor(\mathbf{C}) \to Mor(\mathbf{C})$

such that

- the class of morphisms is closed under composition: for each pair of morphism $f\colon A \to B$, $g\colon B \to C$ there is a morphism $g \circ f\colon A \to B$,

- there is an identity morphism $\mathrm{id}_A\colon A \to A$ for each object $A \in Ob(\mathbf{C})$, and

- morphism composition is associative.

Note that there can be multiple morphisms from one object to another, e.g., if a graph G matches as a subgraph in H in different ways, then this corresponds to multiple morphisms $m_i \colon G \to H$.

In this contribution we focus on categories restricted to monomorphisms, as argued in Sec. 2.2. In Sec. 4 we further discuss chemically motivated variations, e.g., to incorporate vertex/edge labels and restrict graphs to be simple.

Concepts in category theory are often illustrated using *commutative diagrams*, which are directed graphs where the vertices represent objects and the edges represent morphisms. Composition of morphisms can then be thought of as creating a new edge representing an entire path of morphisms. Given two paths that originate and end at the same objects, if the composition along each path results in the same morphism, then the two paths are said to *commute*. If all such pairs of paths in a diagram commute the diagram it self is said to commute. For defining graph transformation and composition of rules, we need three interrelated constructions: pushouts, pullbacks, and pushout complements.

Definition 2.3 (Pushout, see [15, Definition A.17])**.** Given two morphisms $f \colon A \to B$ and $g \colon A \to C$ in some category, the morphisms $f' \colon C \to D$ and $g' \colon B \to D$ forms a pushout of f and g if and only if (see also Fig. 2)

i) $g' \circ f = f' \circ g$, i.e., the square of Fig. 2 commutes, and

ii) for all pairs of morphisms $f'' \colon C \to D'', g'' \colon B \to D''$ with $g'' \circ f = f'' \circ g$, there exists a unique morphism $d'' \colon D \to D''$ such that $f'' = d'' \circ f'$ and $g'' = d'' \circ g'$.

The object D is called the *pushout object*.

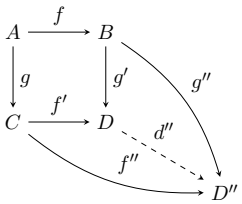


Figure 2. Illustration of the definition of a pushout (Def. 2.3). The pair of morphisms f', g' is a pushout of f, g if i) the composition $A \xrightarrow{f} B \xrightarrow{g'} D$ commutes with $A \xrightarrow{g} C \xrightarrow{f'} D$, and ii) for all morphisms f'' and g'' that similarly commute with f and g, respectively, there exists a unique morphism d'' that commutes with the other morphisms.

An interpretation of a pushout for graphs is that the graph D is the union of B and C with equality specified by common vertices and edges in A.

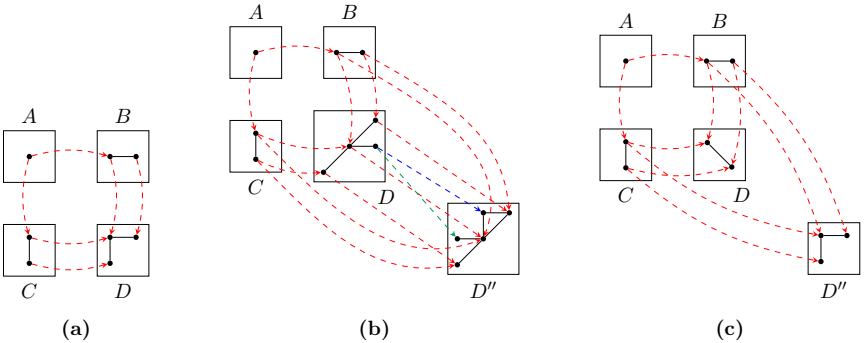


Figure 3. Illustration of a pushout and pushout candidates in the category of undirected graphs. a a pushout, which "glues" B and C along A to obtain the pushout object D. b a pushout candidate which is too large. Two commuting morphisms $D \to D''$ can be found: one using the blue mapping, and one using the green mapping. c a pushout candidate which is too small. Unrelated vertices of B and C have been merged, and with a second pushout candidate D'' we can not find any commuting morphisms $D \to D''$.

In Fig. 3 we show an example of a graph pushout, along with candidates for the pushout that satisfy the first condition of the definition, but not the second. The candidate D in Fig. 3b is not a pushout object because it is too large. The extra vertex and edge makes it possible to find an even larger candidate D'' and two different morphisms, indicated by the choice of either the blue or green mapping. The candidate in Fig. 3c is on the other hand too small, as we have mapped the extra vertex of B and C to the same vertex in D. When considering a second pushout candidate D'' we are not able to find any morphism from D to D'' such that the diagram commutes.

Suppose we are in the opposite situation of a pushout, i.e., given f' and g', we want to find suitable morphisms f and g. This dual construction is a *pullback*:

Definition 2.4 (Pullback, see [15, Definition A.22])**.** Given two morphisms $f' \colon C \to D$ and $g' \colon B \to D$ in some category, the morphisms $f \colon A \to B$ and $g \colon A \to C$ form a pullback of f' and g' if and only if (see also Fig. 4)

i) $g' \circ f = f' \circ g$, i.e., the square of Fig. 4 commutes, and

ii) for all pairs of morphisms $f'' \colon A'' \to B, g'' \colon A'' \to C$ with $g' \circ f'' = f' \circ g''$, there exists a unique morphism $a'' \colon A'' \to A$ such that $f'' = f \circ a''$ and $g'' = a'' \circ g$.

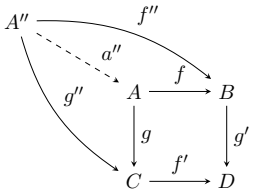


Figure 4. Illustration of the definition of a pullback (Def. 2.4), the dual of a pushout. The pair of morphisms f, g is a pullback of f', g' if i) the morphisms commute, and ii) for all morphisms f'', g'' that commute with f, g, there exists a unique morphism a'' that commutes with the rest.

For categories of graphs we can interpret a pullback as the construction of the common subgraph A of B and C as determined by the embedding of B and C in D. In Fig. 5 we have illustrated a graph pullback, and a candidate that does not fulfil both criteria.

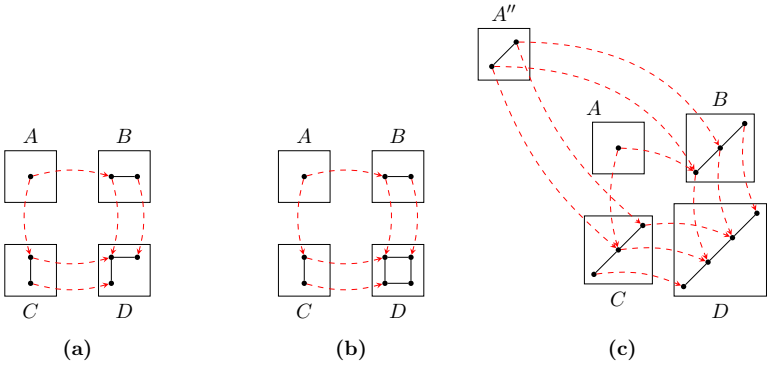


Figure 5. Illustration of two pullbacks and a pullback candidate in the category of undirected graphs. a a pullback, which intersects B with C using D to obtain the pullback object A. a a pullback, where the corresponding pushout object of $C \leftarrow A \rightarrow B$ does not yield the graph D we started from. c a pullback candidate which is too small. There is no morphism $A'' \to A$ due to the edge in A''. However, even when ignoring this edge problem the resulting morphism would not commute with the remaining morphisms.

The third variation of the situation we need is the *pushout complement*, where f and g' are given, and we wish to find g and f'.

Definition 2.5 (Pushout Complement, see [15, Definition A.20])**.** Given two morphisms $f\colon A \to B$ and $g'\colon B \to D$ in some category, the morphisms $g\colon A \to C$ and $f'\colon C \to D$ forms the pushout complement of f and g' if and only if f' and g' is the pushout of f, g.

2.2 Transformation Rules and Derivations

A graph transformation rule in the DPO formalism is a span $p = (L \xleftarrow{l} K \xrightarrow{r} R)$. The left-hand graph L plays the role of a precondition for the application of the rule, while the right-hand graph R is a post-condition. Their relation is specified by the context graph of the rule K, also called the *gluing graph*, together with a monomorphism l and a general graph morphism r. Transformation rules are "symmetric" in the sense that there is no particular difference between the left and right sides, other than the names we have given to them. Thus we can immediately define the *inverse transformation* rule $p^{-1} = (R \xleftarrow{r} K \xrightarrow{l} L)$. This is a quite useful property for the modelling of chemistry, where reactions are necessarily invertible in principle, i.e., as long as energetic considerations are disregarded. We therefore restrict r to be a monomorphism as well. If the morphisms are unimportant or clear from the context we may write a rule simply as $p = (L, K, R)$.

The transformation of a graph G using p proceeds in the following manner (see Fig. 6).

1. Find a match morphism $m\colon L \to G$, if it exists.
2. Construct D as the pushout complement of l, m, if it exists.
3. Construct H as the pushout object of d, r, if it exists.

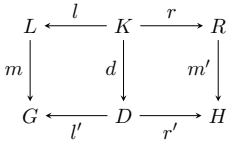


Figure 6. The diagram for a direct derivation $G \overset{p,m}{\Longrightarrow} H$ in the Double Pushout formalism of G to H using the rule p and the matching morphism m.

Such a transformation is called a *direct derivation* of H from G, using the rule p and the matching morphism m. As a shorthand we write a it as $G \overset{p,m}{\Longrightarrow} H$, or as either $G \overset{p}{\Rightarrow} H$ or $G \Rightarrow H$ if the match morphism and rule are unimportant. Assuming m is found, the existence of D can be characterised by what is called the *gluing condition*: G, p, and m

satisfy the gluing condition if both the *dangling condition* and *identification condition* are satisfied.

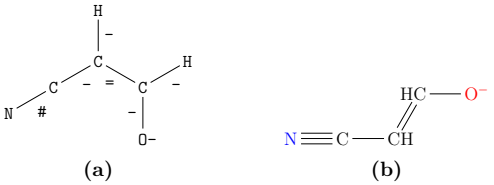
***dangling condition*:** there are no edges in $E_G \backslash m(E_L)$ incident to a vertex in $m(V_L \backslash l(V_K))$. That is, if p specifies the deletion of a vertex, it can only be applied if it also specifies the deletion of all the edges incident to the vertex.

***identification condition*:** there are no distinct vertices $u, v \in V_L$ with $m(u) = m(v)$, $u \in l(V_K)$, $v \notin l(V_K)$, and similarly for distinct edges of E_L. That is, if p specifies the deletion of a vertex or edge, but the preservation of another vertex or edge, then the matching morphism m may not identify those vertices or edges with each other.

A proof for the equivalence of the existence of the graph D and the gluing condition is given in [15, Fact 3.11]. Additionally, one can prove that whenever D exists, it is unique up to isomorphism.

We additionally restrict the matching morphism m to be a monomorphism, as atoms could otherwise be merged when matched by a rule. Note that with this restriction the identification condition is always fulfilled.

In Sec. 4 we describe additional restrictions and variations relevant for modelling of chemistry. This includes the addition of labels on vertices and edges, to describe atom and bond types. For illustrative purposes we already use such labels in examples in the next section. An example is shown in Fig. 7, where each graph is labelled on vertices with an atom type.



(a) **(b)**

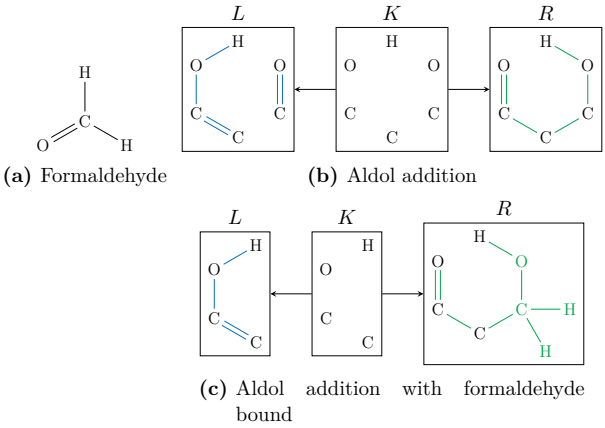
Figure 7. Example of the graph depiction scheme used in this contribution. a a labelled graph representing a molecule, depicted explicitly with labels. b the same graph visualised in style familiar from chemistry. Vertex labels encode atom types, including charges, while edge labels depict bond types. Double and triple bonds are thus encoded as one edge each, but visualised using parallel lines. In the following sections we primarily use the visualisation style from b.

The edges are similarly labelled with a bond type, though this is depicted with different

edge styles instead of with explicit labels. A single bond is shown as a single line, a double bond is shown as two parallel lines, and a triple bond as three parallel lines.

3 Composition of Transformation Rules

For ordinary mathematical functions with multiple arguments there is the concept of *partial application* [25]. For example, let $f(x, y) = x^y$ then we can define a new function $f'(x) = x^2$ by partially applying f to the number 2 in the second position, i.e., $f'(x) := f(x, 2)$. A specialised form of partial application, called *currying*, is for example an integral part of the programming language Haskell. With the chemical view on DPO rules, where they are applied not just to a single graph but to a multiset of connected graphs, we can view a rule $p = (L \leftarrow K \rightarrow R)$ with k connected components in L as a function of up to k unordered arguments. Similar to partial function application, we can then imagine that a transformation rule can be partially applied to a graph, with the result being a new rule, with fewer connected components in its left-hand graph. For example, the rule depicted in Fig. 8b have two connected components in the left-hand graph, and it can therefore be applied to either an enol and a molecule with a carbonyl group, or a single molecule with both features.



(a) Formaldehyde **(b)** Aldol addition

(c) Aldol addition with formaldehyde bound

Figure 8. An example of partial application of a graph transformation rule. Formaldehyde a can be bound to one of the components of the left-hand graph of aldol addition b. The resulting rule c represents the addition of formaldehyde to an enol.

We can now derive a new rule, Fig. 8c, where a formaldehyde molecule have been

bound to one of the components of the original rule. Suppose we bind another graph to the new rule. The last component would then be used and the resulting rule have the form $(\emptyset \leftarrow \emptyset \rightarrow H)$. Such a rule is essentially equivalent to a constant function, implying that a DPO derivation can be calculated by iterated graph binding.

A more general concept than partial application is function composition. Consider two DPO rules $p_1 = (L_1 \leftarrow K_1 \rightarrow R_1)$ and $p_2 = (L_2 \leftarrow K_2 \rightarrow R_2)$ and suppose we have a monomorphism from L_2 to R_1. Then if we first have a derivation $G_1 \overset{p_1}{\Rightarrow} G_2$, then we for sure can find a match of L_2 in G_2 and potentially a derivation $G_2 \overset{p_2}{\Rightarrow} G_3$. If we can define a composition $p_2 \circ p_1$ the two derivations can be combined into $G_1 \overset{p_2 \circ p_1}{\Longrightarrow} G_3$.

The concept of rule composition is in the area of graph transformation related to both *D-concurrency* [26] and *E-concurrency* [15, 27]. In this section we first describe the most general form of composition, similar to D-concurrency, and then several special cases with relevance to the modelling of chemistry. Secondly we describe how enumeration of compositions can be implemented in practice.

3.1 Classes of Composition

In general, the composition of two rules $p_i = (L_i \overset{l_i}{\leftarrow} K_i \overset{r_i}{\rightarrow} R_i)$ for $i = 1, 2$, depends on a common subgraph of R_1 and L_2. Formally (see also [26, Section 6.2]) we say that p_1 and p_2 are composed using a graph D with morphisms $d_1 \colon D \rightarrow R_1$, and $d_2 \colon D \rightarrow L_2$, and write $p = p_1 \bullet_D p_2$ as shorthand for the composed rule, if it exists. Note that the order of the operands is the reverse compared to the usual composition operator $\circ$. The composed rule p corresponds to first applying p_1 and then p_2.

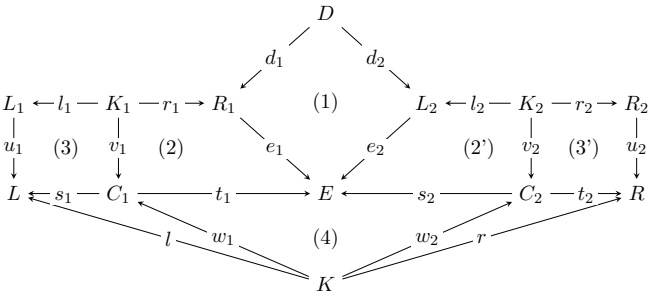


Figure 9. Commutative diagram for general rule composition [26, Section 6.2]. The two rules $p_i = (L_i \overset{l_i}{\leftarrow} K_i \overset{r_i}{\rightarrow} R_i)$ for $i = 1, 2$ are composed using the common graph D and the morphisms d_1 and d_2. The resulting rule is $p = p_1 \bullet_D p_2 = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ with $l = s_1 \circ w_1$ and $r = t_2 \circ w_2$.

The composed rule exists if and only if the diagram in Fig. 9 exists with the squares (1), (2), (2'), (3), (3') all being pushouts, and (4) being a pullback. We then have $p = p_1 \bullet_D p_2 = (L \xleftarrow{l} K \xrightarrow{r} R)$ with $l = s_1 \circ w_1$ and $r = t_2 \circ w_2$ as the resulting composition. Algorithmically we can describe the construction of p as follows:

1. Construct E as the pushout object of (1).
2. Construct C_1 and C_2 as the pushout complement objects of respectively (2) and (2').
3. Construct L and R as the pushout objects of respectively (3) and (3').
4. Construct K as the pullback object of (4).

If any of the constituent constructions are undefined, then the composition as whole is not defined.

In the following sections we describe simplified cases of composition for specific choices of the common subgraph D. These play an important role in applications to chemical systems and can be handled more efficiently in practical implementations.

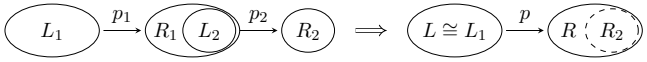
3.1.1 Parallel Composition

For D being the empty graph we can always compose the rules, and obtain a rule which combines the effect of p_1 and p_2, i.e., $p = (L_1 \cup L_2 \xleftarrow{l_1 \cup l_2} K_1 \cup K_2 \xrightarrow{r_1 \cup r_2} R_1 \cup R_2)$. Intuitively we can see this using the diagram in Fig. 9, where $D = \emptyset$ means that pushout (1) degenerates to a disjoint union, i.e., $E = R_1 \cup L_2$. The image of e_1 is thus disjoint from the copy of L_2, and the completion of pushouts (2) and (3) then propagates L_2 into C_1 and L without modification. Symmetrically R_1 is propagated into R. In short we write this merging of two rules as $p = p_1 \bullet_\emptyset p_2$ and visualise it as in Fig. 10.

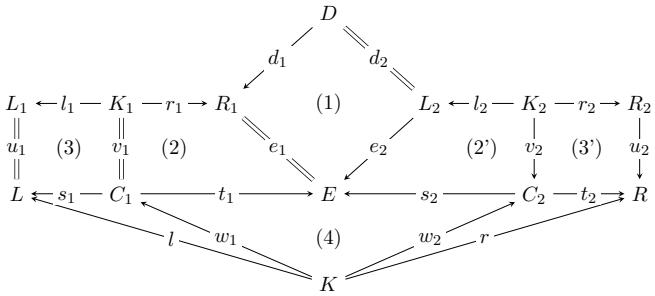
Figure 10. Composition $p = p_1 \bullet_\emptyset p_2$ with an empty common subgraph, giving a composed rule which models the parallel transformation using p_1 and p_2.

3.1.2 Full Composition

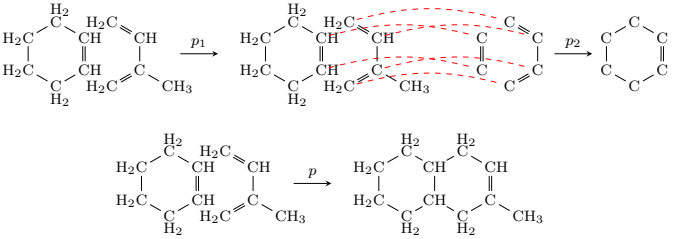
When we see DPO rules as a kind of abstract functions where the left- and right-side graphs are the pre- and postconditions, respectively, we can look at the case where the precondition of p_2 is fulfilled completely by the postcondition of p_1, as illustrated in Fig. 11a.



(a) Abstract depiction



(b) Specialisation of the composition diagram



(c) Chemical example

Figure 11. Full composition $p = p_1 \bullet_{\supseteq} p_2$ where D is a copy of L_2 and d_2 is an isomorphism. a Abstract depiction; L_2 is isomorphic to a subgraph of R_1. b Specialised commutative diagram for full composition, where double-lines represent isomorphisms. As d_2 is an isomorphism so will e_1, v_1 and u_1 be. c Chemical example; $p_1 = (G \leftarrow G \rightarrow G)$ is the identity rule for a graph G encoding the educts cyclohexene and isoprene. The second rule, p_2, is the transformation rule for the Diels-Alder reaction. The composed rule therefore encodes the overall rule of the Diels-Alder reaction on the input molecules. The context graphs and D are omitted from the drawings for simplicity, though the embedding of the common graph D in R_1 and L_2 is indicated by the red dashed lines.

We call this special case *full composition* and formally specify it as when $D \cong L_2, d_2$

$= \mathrm{id}_{L_2}$ and d_1 being a monomorphism. The effect of d_2 being the identity morphism, and thereby an isomorphism, is illustrated in Fig. 11b. Because (1) is a pushout we must have e_1 being an isomorphism, which in turn makes both v_1 and u_1 isomorphisms as well.

A chemical example of full composition is shown in Fig. 11c, where p_1 additionally is a special identity rule that requires a specific graph and does not change it. The effect of this choice of p_1 is discussed in Sec. 3.2. For full composition we note that due to the complete embedding of L_2 in R_1 we have $L \cong L_1$, meaning the resulting rule have the same precondition as p_1. As shorthand we may write $p_1 \bullet_{\supseteq} p_2$ to denote an arbitrary full composition or the enumeration of all such compositions, depending on the context.

We can additionally define the symmetric kind of full composition, $p_1 \bullet_{\subseteq} p_2$ where d_1 is an isomorphism and d_2 a monomorphism. Further we have the special case $p_1 \bullet_{\cong} p_2$ where both d_1 and d_2 are isomorphisms. However, we reserve the term *full composition* for $p_1 \bullet_{\supseteq} p_2$ which particularly is useful for a method for calculating atom traces.

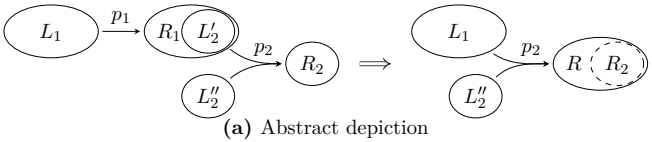
3.1.3 Partial Composition

For chemical graph transformation we have the perspective that graphs can be interpreted as multisets of their connected components, and if the rule p_2 models the merging of two molecules then L_2 will have two connected components. In a full composition we had both of those components embedded in R_1, but we can generalise to the case where a subset of the components are embedded in R_1, as visualised in Fig. 12a.

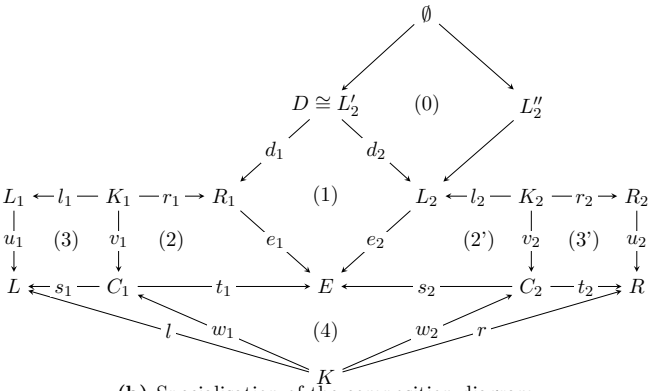
Formally we can describe this type of composition by requiring L_2 to be the disjoint union of two graphs L_2' and L_2'', and letting $D \cong L_2'$. This is illustrated in Fig. 12b, where we denote the disjoint union by a pushout from the empty graph. As for full composition we require d_1 to be a monomorphism. Note that this specification generalises both parallel and full composition, but we however refer to parallel composition explicitly and do not include it in partial composition, i.e., we require $D \ncong \emptyset$. As shorthand we write $p_1 \bullet^c_{\supseteq} p_2$ for an arbitrary partial composition, or the enumeration of them, due to the splitting of L_2 into connected components and requiring $R_1 \supseteq D$.

Fig. 12c shows a chemical example with partial composition, where L_2 has two connected components. The smaller component is used as the common subgraph D is thus not a precondition in the resulting rule, while the other component is preserved in L.

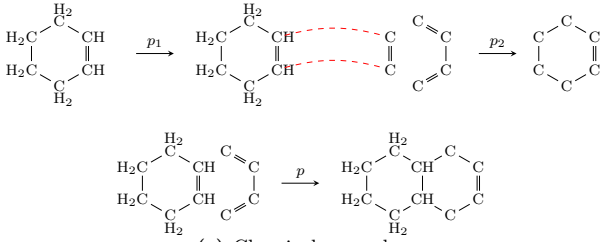
In the symmetric case of partial rule composition we can instead require components of R_1 to be a subgraph of L_2 and write $p_1 \bullet^c_{\subseteq} p_2$.



(a) Abstract depiction



(b) Specialisation of the composition diagram

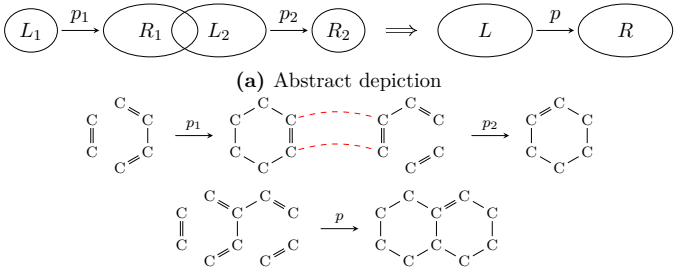


(c) Chemical example

Figure 12. Partial composition $p = p_1 \bullet^{c}_{\supseteq} p_2$ where D is a copy of a non-empty subset of the connected components of L_2, and d_2 is the inclusion morphism back into L_2. a Abstract depiction; connected components of L_2 are either completely matched into R_1 or not at all. b Specialised commutative diagram for partial composition. The selection of connected components of L_2 to form D is specified by pushout (0), that degenerates to a disjoint union. To exclude parallel composition from partial composition we require $D \ncong \emptyset$. c Chemical example; p_1 is the identity rule for cyclohexene and p_2 is the Diels-Alder transformation rule. The composed rule encodes the partial application of the Diels-Alder reaction to the molecules, leaving the diene to be instantiated at a later stage. The context graphs and D are omitted for simplicity, though the embedding of the common graph D in R_1 and L_2 is indicated by the red dashed lines.

3.1.4 General Composition

We now briefly return to the most general case of composition where D simply must be a common subgraph of R_1 and L_2. As shorthand notation we use $p_1 \bullet_\cap p_2$, and visualise the relation abstractly as in Fig. 13a, while a chemical example is shown in Fig. 13b.



(a) Abstract depiction

(b) Chemical example

Figure 13. General rule composition where D is a common subgraph of both R_1 and L_2. The commutative diagram is shown in Fig. 9. a Abstract depiction, with the common subgraph D shown as the intersection of R_1 and L_2. b Chemical example; the rule for the Diels-Alder reaction is composed with itself. The context graphs and D are omitted for simplicity, though the embedding of the common graph D in R_1 and L_2 is indicated by the red dashed lines.

Several further subclasses of composition can be defined, for example requiring D to be maximal with respect to inclusion in R_1 and L_2, but we have not encountered a natural use of further classes in the context of chemistry.

3.2 Binding, Unbinding, and Identification of Graphs

In the beginning of this section we argued that given a graph and a rule we can define how to bind the graph onto the rule and obtain a new rule modelling the partial application. To formalise graph binding we now consider a rule $p_1 = (\emptyset \leftarrow \emptyset \rightarrow G)$ to be equivalent to the graph G, since it models the unconditional creation of G. The creation of H in the derivation $G \overset{p_2,d_1}{\Longrightarrow} H$ is then equivalent to the full composition $p_1 \bullet_\supseteq p_2 = (\emptyset \leftarrow \emptyset \rightarrow G) \bullet_\supseteq p_2 = (\emptyset \leftarrow \emptyset \rightarrow H)$ (see Fig. 11b). Consider now a division of L_2 into the disjoint subgraphs L_2' and L_2'', and assume we want to model the binding of G to L_2'. We can then see this as the partial composition $p = (\emptyset \leftarrow \emptyset \rightarrow G) \bullet_{L_2'} p_2$, illustrated in Fig. 14.

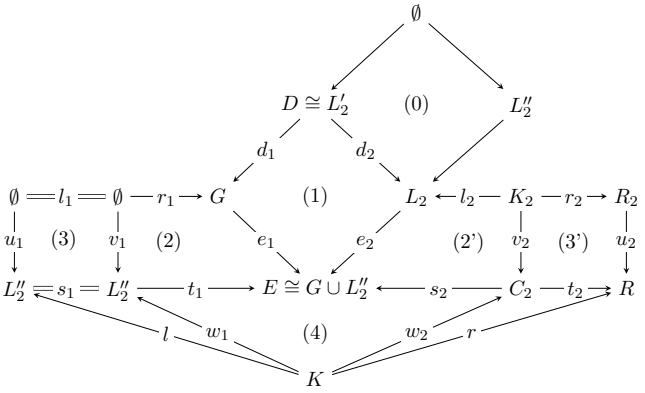


Figure 14. Commutative diagram for binding a graph G to a rule p_2 by reduction to a partial composition $(\emptyset \leftarrow \emptyset \rightarrow G) \bullet^{c}_{\supseteq} p_2$. Double-lines represent isomorphisms. Pushout (0) makes L_2 the disjoint union of L'_2 and L''_2, while the morphism d_1 embeds L'_2 into G. For (1) to be a pushout we thus have E being the disjoint union of G and L''_2. Pushouts (2) and (3) then propagates L''_2, the unbound part of L_2, to be the left-side graph of the resulting rule.

From the division of L_2 into disjoint graphs and the definition of pushouts we have that the resulting composed rule has L''_2 as its left-side graph, which exactly was the part of L_2 we did not want to bind onto.

Many reactions in biochemistry have compounds, such as water, as educts or products that are present in a high quantity such that we can argue they are part of the environment and hence are always present. Reaction patterns, in form of DPO rules, can then be simplified by binding the abundant compounds to obtain the rules modelling only the "interesting" part of the reaction pattern. Graph binding can only simplify the educt side of the reaction pattern, while for the product side we need to use the symmetric composition $p_1 \bullet^{c}_{\supseteq} (H \leftarrow \emptyset \rightarrow \emptyset)$, which we then refer to as *graph unbinding*.

The third trivial way of creating a rule from a graph G is to create the identity rule $(G \leftarrow G \rightarrow G)$. We use this construction, together with full composition, to calculate atom traces. The property we use is that the result of the composition $(G \leftarrow G \rightarrow G) \bullet_{\supseteq} p_2$ is a rule $(G \leftarrow D \rightarrow H)$ corresponding the lower half of the usual DPO diagram for the direct derivation $G \overset{p_2}{\Rightarrow} H$.

Based on their use we refer to the rules $(\emptyset \leftarrow \emptyset \rightarrow G)$, $(G \leftarrow \emptyset \rightarrow \emptyset)$, and $(G \leftarrow G \rightarrow G)$ respectively as the *bind rule*, *unbind rule*, and the *identity rule* for the graph G.

4 Chemical Graphs and Chemical Rules

So far our presentation has been motivated by chemical modelling, only with restrictions on the morphism class to monomorphisms. The methods are thus generic and in principle applicable in other areas.

However, the graph model of chemistry has additional intricacies that must be handled, or can be exploited. For the sake of a streamlined presentation we focus here on organic chemistry. In particular, we ignore here organometallic complexes, compounds with multi-centre bonds, as well as ionic crystal lattices, etc. That is, systems for which representations in the form of (small) graphs are at least not obvious.

4.1 The Category of Simple Graphs

In our model of chemistry the connectivity of molecules is modelled by undirected graphs. While a double bond is realised by twice as many electrons as a single bond, its behaviour is not accurately reflected by encoding it as two parallel single bonds. We thus encode bond orders in labels on edges, and require a molecular graph to have no loops or parallel edges. This restriction forces us to consider multiple distinct constructions of pushout, pullback, and pushout complement. To see this, consider the span in Fig. 15a.

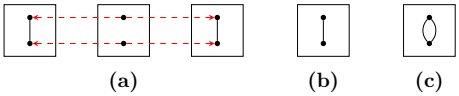


Figure 15. The pushout object of a, for multigraphs is the graph depicted in c. When restricted to simple graphs the result could be the graph depicted in b, where the two edges are merged. However, for the modelling of chemistry we define that the span can not form a pushout.

For graphs where parallel edges are allowed we can create the pushout object depicted in Fig. 15c, which conforms to the idea of a disjoint union of the non-common vertices and edges. However, for simple graphs this is not possible since parallel edges are not allowed. Hence we either have to insist that no pushout exists for this span, or that non-common edges can be merged in the pushout object and thus accept the graph in Fig. 15b as the pushout object. In [28], the authors chose the latter option, which necessitates the definition of *minimal* pushout complements.

For the modelling of chemistry it is more natural to take the first option. Hence we posit that no pushout exists in this case. The span in Fig. 15a can arise in the

context of transformation of molecules if a rule asks for a bond to be created between two atoms that are already connected by a chemical bond. In chemical terms, such a reaction cannot take place since the existing bond cannot miraculously vanish and chemistry does not provide a natural notion of "merging" chemical bonds between the same two atoms. In both cases the transformation would have a side-effect not explicitly specified by the rule. In chemistry all reactions are in principle invertible, but side-effects would hinder a straightforward incorporation of this property in a graph transformation model.

4.2 Labelled Graph Transformation

Chemical graphs have labelled vertices and edges for encoding bond orders, atom types, charges, etc. For a category of labelled graphs, let Ω be a set of labels used on vertices and edges. We write $\ell_G(x)$ to denote the label attached to a vertex or edge x of a graph G. A labelled graph morphism $m\colon G \to H$ must then fulfil that $\ell_G(x) = \ell_H(m(x))$ for each vertex and edge in G, in addition to the usual graph morphism requirements. However, now consider a rule $(L \xleftarrow{l} K \xrightarrow{r} R)$, and some vertex or edge x of K. By the morphism definition x must have the same label in L and R. Using labelled graphs it is thus not possible to encode change of labels, which is needed for example for modelling charge changes on atoms. For edges, we could perform a deletion and recreation with a different label, but this is not possible for vertices in unknown context, as it would violate the dangling condition.

The limitations of labelled graph transformation are well-known in the literature [15, 29], but can easily be circumvented, e.g., by using the more expressive framework of typed attributed graphs [15], or simply allowing the graph K of a rule, and the graph of a derivation D, to be partially labelled [29]. Following the latter approach, an unlabelled vertex or edge x of a graph G is denoted $\ell_G(x) = \bot$. For a morphism $m\colon G \to H$ and a vertex or edge x in G we then allow $\ell_G(x) = \bot \neq \ell_H(m(x))$. That is, an unlabelled vertex (resp. edge) may be mapped to any vertex (resp. edge) irrespective of label. It is clear that this is again a category since it contains the identity morphism and the composition of morphisms is again a morphism on partially labelled graphs by virtue of the transitivity of label matches.

In Fig. 16 a direct derivation with change of labels is depicted.

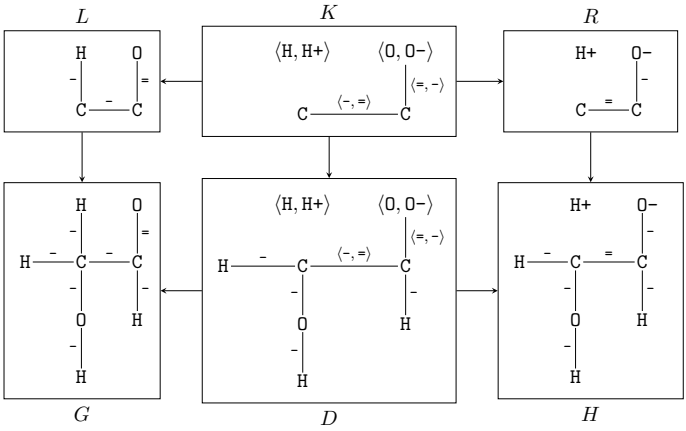


Figure 16. Example of a derivation using labelled graphs, but allowing partial labelling of the context graph K and intermediary graph D, to facilitate changing of labels. Each vertex and edge of these graphs without labels are visualised with a pair of labels from L and R, respectively G and H. To reduce clutter in illustrations we usually completely omit the depiction of edges in K and D that change label.

Instead of using the no-label symbol $\perp$ in K (resp. D) we instead depict the pair of labels from the L and R (resp. G and H). To further reduce clutter in illustrations we usually completely omit edges in K and D when they change labels.

4.3 Chemically Valid Rules

The general labelled graph transformation formalism allows arbitrary label changes, and even insertion and deletion of vertices, in rules. A chemically valid rule in addition must preserve the chemical elements, and only change charges. Bond orders may be changed as well, but the overall change of the number of electrons must be balanced. At present our implementation leaves it to the user to ensure that reaction rules are chemically meaningful. We note that checking conservation constraints become easy when a representation of molecules is chosen that explicitly describes the distribution of valence electrons into bonding electron pairs and lone pairs. Lewis diagrams [30] may serve as an example.

Note that if a rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ conserves the atoms, then the restrictions $l_{|V}$ and $r_{|V}$ of the morphisms l and r to the vertices (atoms) necessarily are bijections. Incidentally this means the dangling condition is trivially satisfied for all matches of such

a rule. Since the composition of bijections is again a bijection, conservation property of chemical rules is preserved under rule composition. Similarly, conservation of chemical elements and electrons is transitive under rule composition. However, we have found it useful to work with rules that do not fulfil the conservation property, both in low-level algorithms, and for certain types of modelling. In the following sections we therefore do not assume the property to hold in general.

The basic graph-based model of chemistry does not include stereochemical information. Based on permutation groups it is however possible to extend the model to cover such local geometric information [31]. One can then define the additional constraints that a stereochemically valid rule must satisfy as further discussed in [31].

5 Implementation Considerations

Transformation rules can in practice be represented in different ways, which in turn influences how best to implement an algorithm for rule composition. In this section we describe details of our implementation in the software package MØD [32, 33].

5.1 Representation of Transformation Rules

A transformation rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ in the category of simple graphs could naturally be represented directly by three graphs and two vertex maps. However, this is arguably a rather verbose representation when restricted to monomorphisms, where K is a subgraph of both L and R and thus stored three times. To obtain a more compact representation that also allows for simpler algorithm implementations we disallow certain rules that have alternative equivalent encodings. Consider the rule depicted in Fig. 17a, which models first the removal and then the addition of an edge, but with a different label.

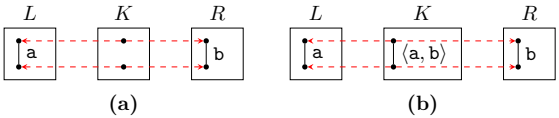


Figure 17. The rule in a models the removal of an edge with label `a` and the addition of the same edge but with label `b`. We do not represent this kind of transformation, but instead allow rules to change labels as in the rule in b.

Fig. 17b shows a functionally equivalent rule, which models the label change directly, i.e., in the underlying graph the edge is invariant. Only if we were to attach auxiliary

data to edges then the application of these two rules would have observable differences.

To simplify our representation of a rule we opt not to allow rules such as in Fig. 17a. That is, for all rules $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ we require that for pairs of vertices $u, v \in K$, if $(u, v) \notin E_K$ then $(l(u), l(v)) \notin E_L \vee (r(u), r(v)) \notin E_R$. Now let $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ be a labelled transformation rule as described in the previous section, where Ω is the label set for L and R. We can then create a new undirected, labelled graph $\mathcal{C}_p = (V_p, E_p, m_p)$ being the pushout object of $L \xleftarrow{l} K \xrightarrow{r} R$. The vertices and edges are labelled with the function $m_p \colon V_p \cup E_p \to \Omega' \times \Omega'$, where $\Omega' = \Omega \cup \{\perp\}$ is the original label set augmented with a distinct new label $\perp$ that indicates the absence of a label, as previously described. Each vertex in V_p and each edge in E_p was created because it was in one of $L \backslash l(K)$, $R \backslash r(K)$, or K. For vertices/edges created from $L \backslash l(K)$ with label α in L we attach the new label $\langle \alpha, \perp \rangle$. Similarly, a vertex/edge in $R \backslash r(K)$ has a label of the form $\langle \perp, \beta \rangle$, and a vertex/edge in K on the form $\langle \alpha, \beta \rangle$. Clearly the original rule can be recovered from the placement of the $\perp$ labels. In the following section we thus use L, K, and R as shorthands for subgraphs of $\mathcal{C}_p$ with appropriate label projections.

Fig. 18 shows how the rule $p = (L \leftarrow K \rightarrow R)$ from Fig. 16 is represented by the pushout object $\mathcal{C}_p$.

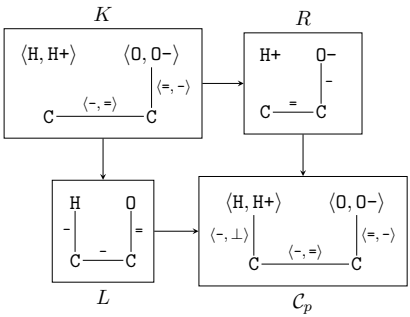


Figure 18. Pushout diagram for the construction of the graph $\mathcal{C}_p$ representing the rule $p = (L \leftarrow K \rightarrow R)$ from Fig. 16.

Note that the rules that are not representable in this manner are exactly those for which we have previously opted not to define a pushout in the category of simple graphs.

5.2 Enumeration of Compositions

The construction of pushouts and pushout complements are unique up to isomorphism, and enumeration of compositions thus reduces to enumeration of spans $R_1 \leftarrow D \rightarrow L_2$ for the given rules. In practice it is desirable to minimise the algorithmic overhead, and we thus assume that the span is represented by a partial vertex map $\psi\colon R_1 \rightarrowtail L_2$, which implicitly encodes D. General compositions $p_1 \bullet_{\cap} p_2$ can thus be enumerated by finding common subgraphs between R_1 and L_2. Full compositions $p_1 \bullet_{\supseteq} p_2$ (and the symmetric case: $p_1 \bullet_{\subseteq} p_2$) can be enumerated by finding all monomorphisms $L_2 \rightarrow R_1$ (symmetrically: $R_1 \rightarrow L_2$), e.g., using the VF2 algorithm [34, 35].

In [23] we described how the span enumeration for partial compositions $p_1 \bullet^c_{\supseteq} p_2$ can be done by first considering all partial assignments of connected components of L_2 to connected components of R_1. For each such assignment, the componentwise monomorphisms can then be enumerated and merged.

5.3 Algorithm for Composition

Given two rules $p_i = (L_i \overset{l_i}{\leftarrow} K_i \overset{r_i}{\rightarrow} R_i,\ i = 1, 2$ and a span $R_1 \leftarrow D \rightarrow L_2$ relating p_1 and p_2, we wish to either construct the composed rule $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R) = p_1 \bullet_D p_2$, or determine that the composition is not defined. This can be done with the procedure described earlier, but in practice it is desirable to avoid constructing intermediary graphs, and instead create the resulting rule p directly. In [23] a brief sketch of such an algorithm for this purpose was given. In the remainder of this section we provide the full details with illustrations.

Using the rule representation described in Section 5.1, assume the input rules p_1, p_2 are given in the form of the two pushout objects $\mathcal{C}_{p_1}$ and $\mathcal{C}_{p_2}$. With this representation we further assume that the relation span $R_1 \leftarrow D \rightarrow L_2$ is given in the form of a partial vertex map $\psi\colon R_1 \rightarrowtail L_2$. For simplicity of the description we extend ψ to the edge set by induction, i.e., if $e = (u, v) \in E(R_1)$, both $\psi(u)$ and $\psi(v)$ are defined, and $(\psi(u), \psi(v)) \in E(L_2)$, then $\psi(e) = (\psi(u), \psi(v))$. We say that a vertex/edge of R_1 (resp. L_2) is *matched* if it is in the pre-image (resp. image) of ψ, otherwise it is *unmatched*.

We have assumed that all morphisms are injective, and given a morphism m and a vertex/edge x in its domain we will therefore simply refer to x and $m(x)$ as the same vertex/edge being present in multiple graphs.

The algorithm has the following overall structure where it directly constructs first the

vertices and then the edges of the composed rule, represented by the graph $\mathcal{C}_p$:

1. Handle each vertex $v \in V(\mathcal{C}_{p_1})$.
2. Handle each vertex $v \in V(\mathcal{C}_{p_2})$.
3. Handle each edge $e \in E(\mathcal{C}_{p_1})$.
4. Handle each edge $e \in E(\mathcal{C}_{p_2})$.

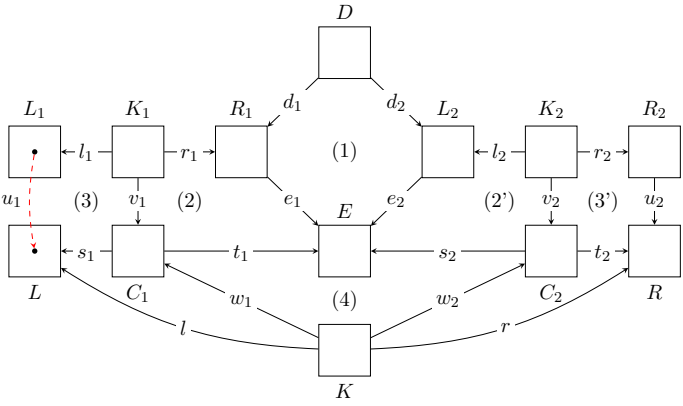
The following subsections provide a case-by-case description of how to handle each vertex/edge. Due to the symmetry of the problem we describe the cases with p_1 as starting point, and only note the symmetric case. The handling of edges is similar to that of vertices, except that additional dangling conditions must be checked. If we work in the category of simple graphs, we must also check parallelism conditions.

In each case we use specialisations of the general commutative diagram for rule composition, shown in Figure 9.

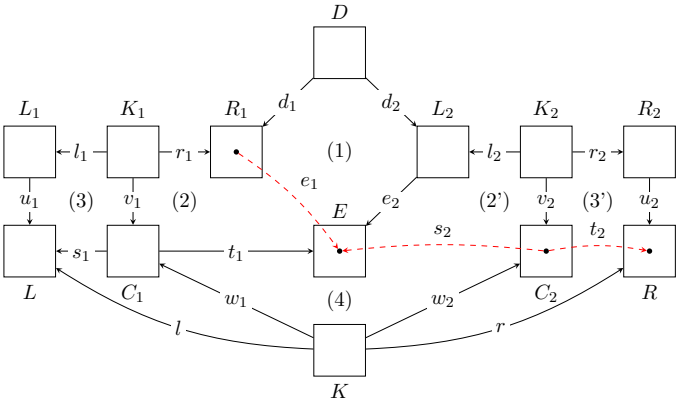
5.3.1 Unmatched Vertices

Let v be a unmatched vertex of $\mathcal{C}_{p_1}$ (symmetrically $\mathcal{C}_{p_2}$), the vertex is simply copied to $\mathcal{C}_p$:

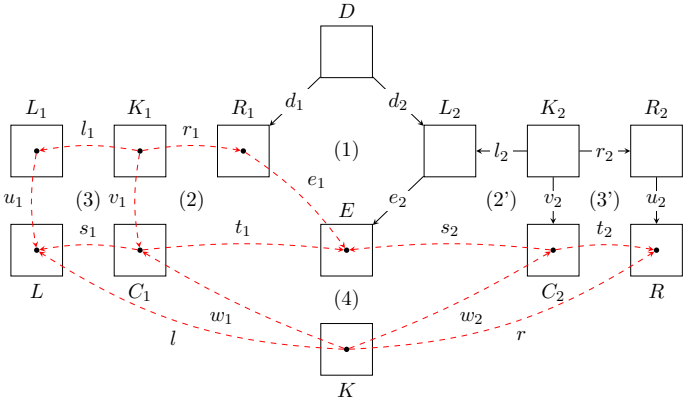
Case $v \in L_1 \backslash K_1$ (symmetrically $v \in R_2 \backslash K_2$): The vertex is in L due to the morphism u_1. The vertex is not in K, as that implies it is both in C_1 and C_2 due to (4) being a pullback. If it were in C_1 it would be in K_1 due to (3) being a pushout, thus contradicting the assumption of the case.



Case $v \in R_1 \backslash K_1$ (symmetrically $v \in L_2 \backslash K_2$): The common graph E is a pushout object of (1), and the vertex is thus in E. The vertex is similarly in C_2 and R due to pushouts (2') and (3'). The vertex is not in K as that would imply that is is also in C_1. If it were in C_1, then it would be in K_1 due to (2) being a pushout, thus a contradiction of the assumption of the case.



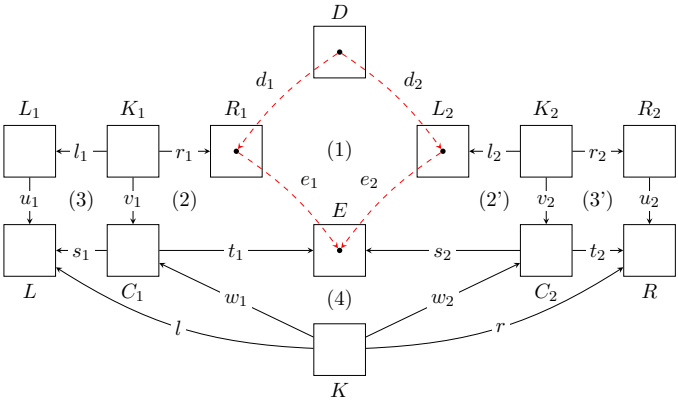
Case $v \in K_1$ (symmetrically $v \in K_2$): The vertex is by definition also in L_1 and R_1, and following the previous case it is thus in R. From the morphisms u_1 and v_1 we get the vertex into L and C_1. The square (4) is a pullback, and as the vertex is in both C_1 and C_2 it is then also in K.



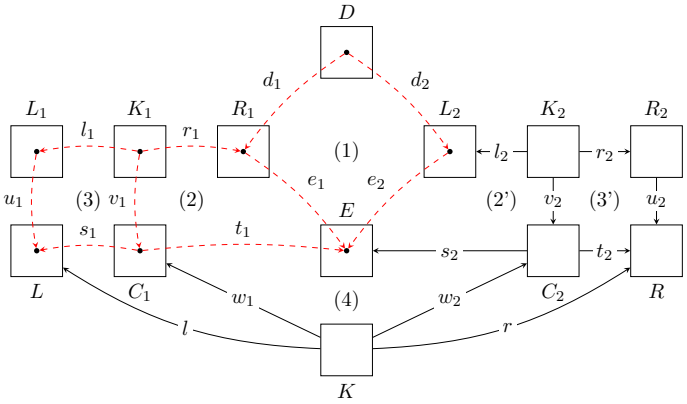
5.3.2 Matched Vertices and Edges

The handling of matched edges is the same as for matched vertices. Let v be a matched vertex. This means it is in D, R_1, and L_2.

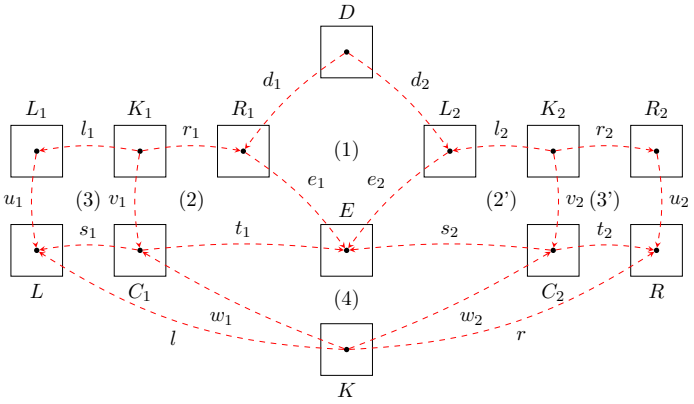
Case v is not in K_1 and not in K_2: The squares (2) and (2′) are pushouts, and by the case assumption the vertex is thus in neither of C_1 and C_2. The vertex is therefore not represented in the resulting rule, and we say that it is *deleted*.



Case $v \in K_1$ (Symmetrically $v \in K_2$): The vertex is in L and C_1 due to the morphisms u_1 and v_1.



Case v is in both K_1 and K_2: This is the combination of the previous case with its symmetric case. We thus have the vertex in both C_1 and C_2, and to make (4) a pullback the vertex must be in K as well.



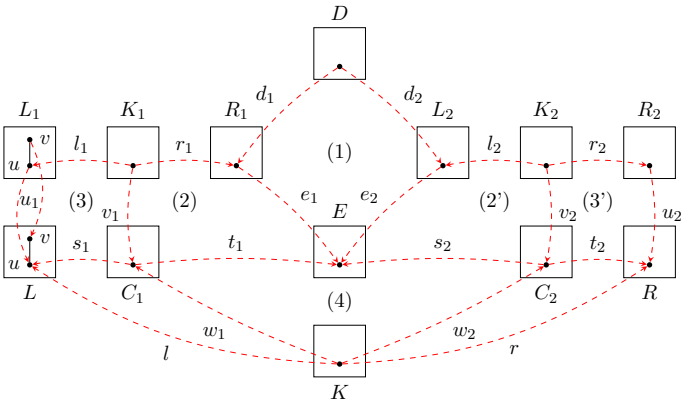
5.3.3 Unmatched Edges

Let $e = (u, v)$ be an edge, handled similar to an unmatched vertex, but with the following extra conditions.

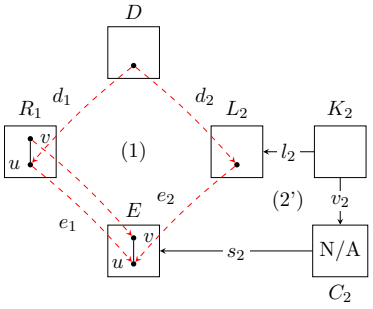
Case No Endpoints Matched: No extra conditions, the edge follows the same logic as unmatched vertices.

Case One Endpoint Matched: Let u be the matched vertex.

Subcase e is in $L_1 \backslash K_1$ (symmetrically e in $R_2 \backslash K_2$): The edge is in L, but not in C_1 and thereby not in K.

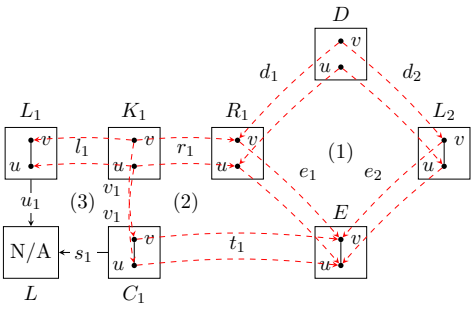


Subcase e is in R_1 and u is in $L_2 \backslash K_2$ (symmetrically e in L_2 and u in $R_1 \backslash K_1$): The composition is undefined; As e is unmatched, it is supposed to be in E, C_2, and R. However, u is matched but deleted by p_2, and the edge will dangle in C_2. This is equivalent to the dangling condition for a direct derivation from E with the rule p_2.

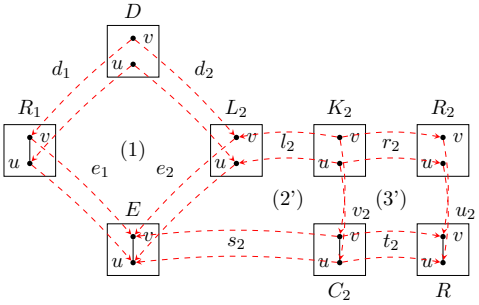


Case Both Endpoints Matched: Both u and v are matched, but e is not. In addition to the dangling condition described above, if the category is restricted to simple graphs then certain parallel edge conditions must be checked.

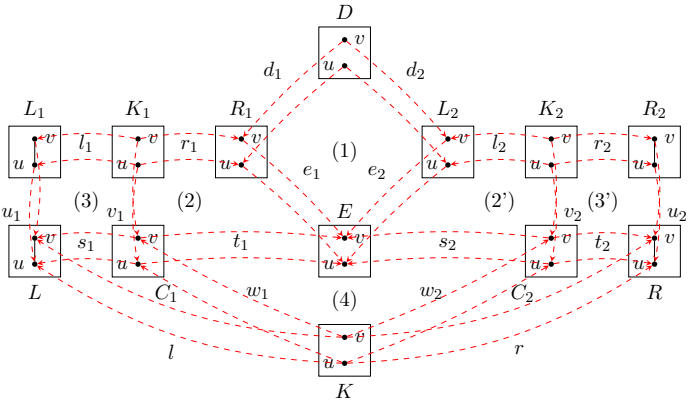
Subcase e in $L_1 \backslash K_1$ (symmetrically e in $R_2 \backslash K_2$): The edge must be in L but not in C_1. However, if another edge (u, v) is in L_2, the composition is not defined. Such an edge would be in both E and C_1, leaving the pushout object L of $L_1 \leftarrow K_2 \rightarrow C_1$ undefined when restricted to simple graphs (i.e., the pushout problem described in Section 4.1).



Case e is in R_1 (symmetrically e in L_2): As the edge is unmatched is will be in E, C_2, and R. If the edge dangles, the composition is undefined. Otherwise, we have the endpoints u and v present in K_2, L_2, C_2, and R. If there is an equivalent edge in K_2 it would then also be in L_2 and thereby be matched, contradicting the case assumptions. However, if there is an equivalent edge in $R_2 \backslash K_2$, the square (3') then has the pushout problem described in Section 4.1. That is, the edge is not in K_2, but in both C_2 and R_2. In this case, when restricted to simple graphs, the pushout object R therefore does not exist and the composition is undefined.



Case e is in both $L_1 \backslash K_2$ and $R_2 \backslash K_2$: This is a special case, as the edge will be in both L and R, but in neither of C_1 and C_2, and thereby not in K. Due to our choice of representation of DPO rules we can not represent this result. Instead of leaving the result undefined we let the result be the equivalent rule where the edge is added to K as well.



6 Application to Computation of Atom Traces

Recall that reactions modelled by direct derivations $G \overset{p}{\Rightarrow} H$ can be represented by rules $(G \overset{g}{\leftarrow} D \overset{h}{\rightarrow} H)$ obtained through the enumeration of full compositions $(G \leftarrow G \rightarrow G) \bullet_{\supseteq} p$. The atom map for each such reaction is a bijection between the vertices of G and H, commuting with the DPO diagram for the direct derivation. That is, the atom map $\alpha \colon V(G) \leftrightarrow V(H)$ is obtained as the vertex map composition $\alpha = h_{|V} \circ g_{|V}^{-1}$.

For a more concise notation we simply use $\bullet$ to mean $\bullet_{\supseteq}$, and $\imath_G$ as a shorthand for the identity rule $(G \leftarrow G \rightarrow G)$. Additionally we assume that the rule composition operator is left-associative, i.e., $a \bullet b \bullet c$ means $(a \bullet b) \bullet c$.

Given a multiset of educt graphs G and a sequence of transformation rules $p_1, p_2, \ldots, p_k$, possibly modelling complete chemical reactions, we can compute all k-step reactions specified by the rules as $\imath_G \bullet p_1 \bullet p_2 \bullet \ldots \bullet p_k$. However, each step in the composition enumeration may lead to multiple different rules. The final right-hand sides may thus be many different combinations of molecules. For tracing atoms in a specific pathway we are only interested in the composed rules that have the output of the pathway in their right-hand sides. Denoting this target multiset of molecules by H, we can then extend the composition expression to $\imath_G \bullet p_1 \bullet p_2 \bullet \ldots \bullet p_k \bullet_H \imath_H$ where the last composition implements an isomorphism check of the right-hand side by composing with $\imath_H$ using H as the common subgraph. If we assume that H consists only of complete molecules, and that no complete molecule is a proper subgraph of another complete molecule, we can simply use full composition for this last step as well. Using full composition also enables the constraint that H only

specifies a sub-multiset of the produced molecules. Such a "check-point constraint" can also be inserted in the middle of a composition sequence if there is a specific requirement for an intermediary state of the system.

6.1 The β-lactamase Mechanism

β-lacatamases (MACiE entry 0002, EC number 3.5.2.6) are bacterial enzymes that convey resistance against β-lactame antibiotics such as penicillins by catalysing the overall reaction

$$\underset{\text{(CHEBI:35627)}}{\beta\text{-Lactam}} + \underset{\text{(CHEBI:15377)}}{\text{water}} \rightarrow \underset{\text{(CHEBI:33705)}}{\text{substituted } \beta\text{-amino acid}}$$

by means of a 5-step mechanism. Using rule composition of those steps we will compute overall rules for the complete enzyme mechanism.

The individual steps in the enzyme mechanism are detailed in the MACiE database [36] as follows (see database entry for full details):

1. Lys73 deprotonates Ser70 thereby initiating a nucleophilic addition onto the carbonyl carbon of the β-lactam.

2. The resulting intermediate collapses, cleaving the C-N bond of the β-lactam and the nitrogen deprotonates Ser130.

3. Ser130 deprotonates Lys73.

4. Glu166 deprotonates water, which initiates a nucleophilic addition at the carbonyl carbon.

5. Collapse of this intermediate leads to cleavage of the acyl-enzyme bond and liberates Ser70, which in turn deprotonates the Glu166.

The 5 individual steps were modelled as transformation rules $p_1, \ldots, p_5$ depicted in Fig. 19.

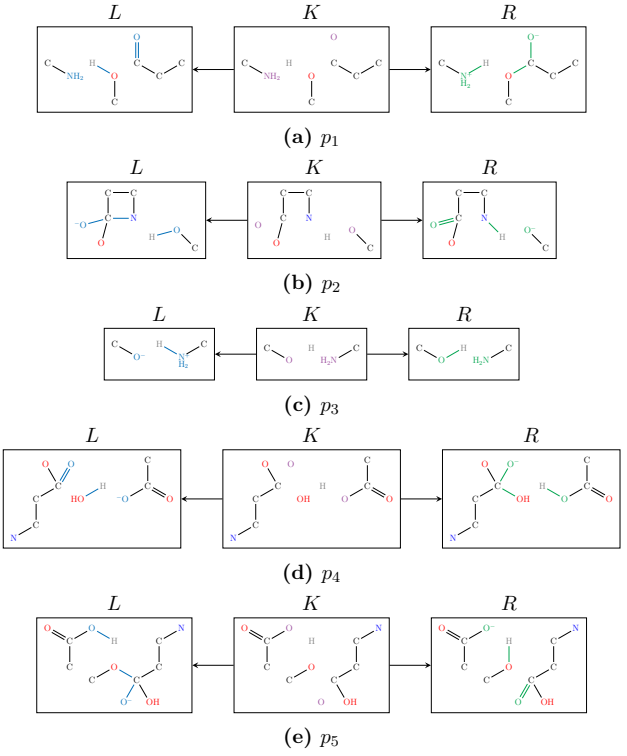


Figure 19. Transformation rules for the 5-step enzyme β-lacatamase mechanism (MACIE entry 0002, EC number 3.5.2.6).

For step (2) an alternative mechanism has been suggested [37]: protonation of the β-lactam nitrogen occurs as the first step in the reaction as an initiation step and not as a consequence of the C-N bond cleavage. We modelled this alternative as a replacement of rule p_2 by two transformation rules p_{1b} and p_{2b}, depicted in Fig. 20.

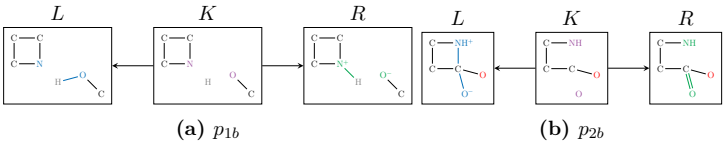


Figure 20. Transformation rules to replace step p_2 from Fig. 19, based on the mechanism as suggested in [37].

The atom traces for the overall reaction are computed by a composition of the rules

$p_1, \ldots, p_5$ with the identity rule for the input compounds, i.e., the β-lactam, water, and the amino acid catalysts (Glu, Lys, and two Ser). Let G and H be the graph representation of the input and output compounds, respectively. The overall composition enumeration

$$\imath_G \bullet p_1 \bullet p_2 \bullet p_3 \bullet p_4 \bullet p_5 \bullet \imath_H \tag{1}$$

results in the two overall rules depicted in Fig. 21.

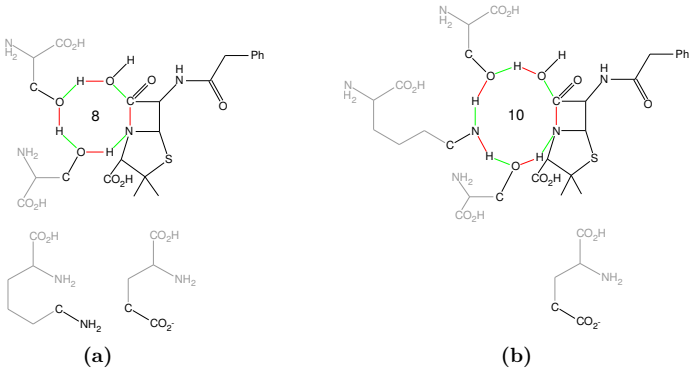


Figure 21. The two overall reactions resulting from either composition Eq. (1) and (2), using the elementary steps of the β-lactamase (`MACiE` entry 0002, EC number 3.5.2.6). Red bonds are broken and green bonds are formed during the transformation. While the overall reactions (as typically found in metabolic databases such as KEGG or MetaCyc) are identical, they differ in their hydrogen trace and the size (8 or 10) of the cyclic virtual transition state. Note that the acid/basic catalysts (the two amino acids lysine and glutamic acid) needed for the reaction to work still show up as precondition in the overall rules. Using partial composition results in two more generic overall reactions. These two rules are depicted as the strict subgraphs resulting from removing the grey parts from the catalysts.

Both are in agreement with the overall mechanism given in `MACiE` and differ only in their hydrogen traces. The overall cyclic virtual transition states are an 8 cycle and a 10 cycle, which only differ by the exchange of a hydrogen in the amino group of Lys. The alternative model for step 2, which corresponds to

$$\imath_G \bullet p_1 \bullet p_{1b} \bullet p_{2b} \bullet p_3 \bullet p_4 \bullet p_5 \bullet \imath_H \tag{2}$$

results in the same two overall rules.

In order to check the flexibility of the reaction with respect to the order of the individual steps of the enzyme mechanism, we investigated all permutations of the rules

for the composition order and verified whether the resulting overall rule produces the substituted β-amino acid as final product. Formally, we compute

$$\imath_G \bullet p_{\sigma(1)} \bullet \ldots p_{\sigma(5)} \bullet \imath_H$$

for all 120 permutations σ. Only the following three compositions are well-defined and result in the expected overall rules: $(p_1, p_2, p_3, p_4, p_5)$, $(p_1, p_2, p_4, p_3, p_5)$, and $(p_1, p_2, p_4, p_5, p_3)$. A detailed inspection shows that step p_3 is the recycling step of the mechanism, which can be applied concurrently to steps p_4 and p_5.

The same experiment based on the rule set $\{p_1, p_{1b}, p_{2b}, p_3, p_4, p_5\}$ shows that eight compositions are possible, all resulting in the same atom traces as given above. The first two steps need to be p_1 and p_{1b}, their relative order however is arbitrary. The subsequent rules p_{2b}, p_4, and p_5 must be in this order. The recycling step p_3 requires the rules p_1 and p_{1b} as prerequisite, but can be performed concurrently to the remaining steps, i.e., it may appear in position 3, 4, 5, or 6, thus accounting for the 8 feasible permutations.

This method allows for an automated analysis of the flexibility of the ordering of individual steps. Usually, only a relatively small number of all possible permutations has to be computed, as most often already the composition of a prefix of an arbitrarily chosen permutation is not possible. In the previous example, for instance, only two of the 30 possible initial two steps are feasible, which prunes most compositions early. The DPO framework provides an inroad to reduce the computational efforts even further. Since each rule is reversible, feasibility can be tested by exploring the space of overall rules from both ends and checking for overlaps at intermediate steps rather then expanding the possible pathways from one end only.

When using full composition we must specify all educt molecules in the initial graph, but we can instead use partial composition to automatically detect the required functionality of the catalysts and the additional compounds (in this case a water molecule). Let G' be the graph representation of β-lactam which is the core compound of the reaction, and H' the corresponding core product molecule. The partial composition of the rules

$$\imath_{G'} \bullet^c_\supseteq p_1 \bullet^c_\supseteq p_2 \bullet^c_\supseteq p_3 \bullet^c_\supseteq p_4 \bullet^c_\supseteq p_5 \bullet^c_\supseteq \imath_{H'}$$

result in the overall rule as depicted highlighted in Fig. 21, i.e., any grey molecule or edge disappears. The overall rules show the automatic inference of the necessity of the four functional units of the catalysts and the necessity of the water molecule, as they are subsequently added to the left side of the overall rule during the partial rule composition.

When defining transformation rules the difficulty often lies in the question of defining the size of the context around a reaction centre: a large context leads to a very specific rule, while a too small context might lead to chemically invalid reactions. Comparing full and partial compositions can be employed as a method to detect the functional units of the catalysts.

The atom mapping of the full composition result shows that in the composed rule with the 8-cycle the acid-base catalysts lysine and glutamic acid are unmodified during the overall process although they are necessary for the mechanism. In the composed rule with the 10-cycle only the acid-base catalyst glutamic acid is unmodified. The other catalysts and the water molecule are modified, however only based on the fact that the hydrogen atom for proton donation is different from the accepting hydrogen.

6.2 The Glycolysis Pathway

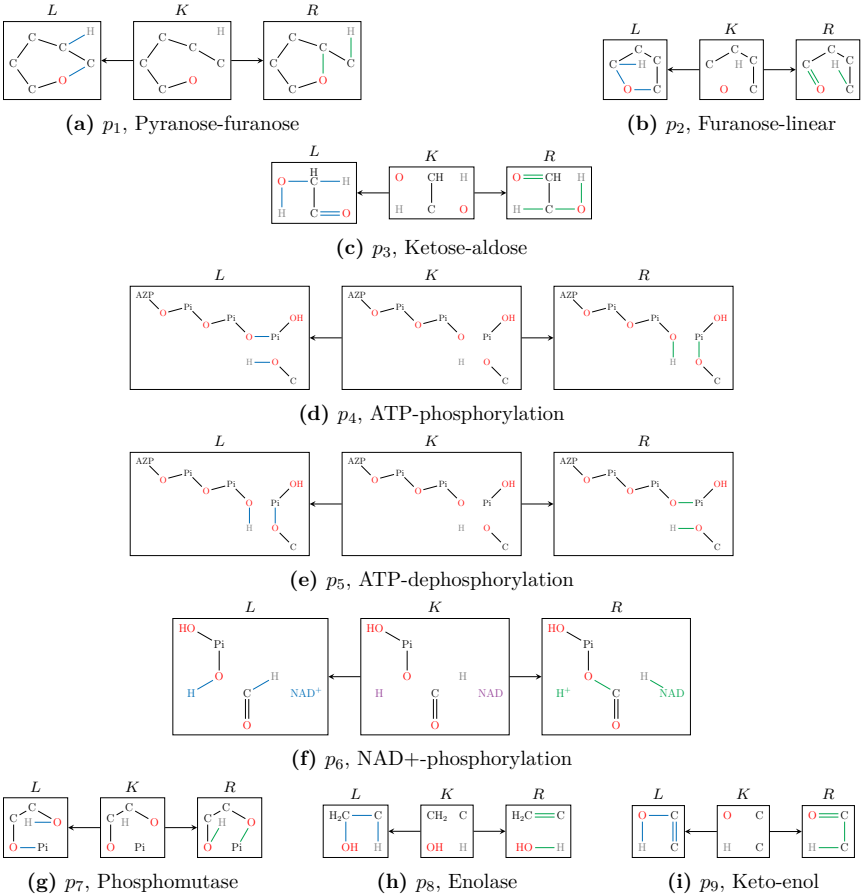
Glycolysis is one of the central pathways in carbon metabolism, which converts 1 glucose molecule (a C6 sugar) into 2 pyruvate molecules (a C3 acid) while creating energy-rich ATP molecules. There are multiple variations of the glycolysis process (see [38] for a review), with the most common being the Embden-Meyerhof-Parnas (EMP) pathway. An alternate pathway is the Entner-Doudoroff (ED) pathway [39], which only creates 1 ATP molecule as opposed to the EMP which creates 2 ATP molecules for each glucose. In this section we illustrate how rule composition expressions can be used to automatically create accurate atom maps for the overall EMP and ED pathways.

Isotope labelling experiments in glycolysis are commonly used to analyse the activity of the different pathway variations (e.g., see [40]). It is known that the EMP and ED pathways lead to different carbon traces, but since atom maps are usually not available in databases it can be tedious and error prone to analyse trace data manually. Here we demonstrate that a chemistry model based on DPO transformation rules enables the automatic inference of atom traces for complete pathways, illustrated with the EMP and ED pathways. The two pathways have been modelled using the following transformation rules:

p_1 Pyranose-furanose	p_5 ATP-dephosphorylation	p_9 Keto-enol
p_2 Furanose-linear	p_6 NAD+-phosphorylation	p_{10} NAD+-oxoreductase
p_3 Ketose-aldose	p_7 Phosphomutase	p_{11} Lactonohydrolase
p_4 ATP-phosphorylation	p_8 Enolase	p_{12} Hydrolyase

p_{13} Reverse aldolase

The rules were manually created using information from `MACiE` database (described in Sec. 6.1) to ensure accurate atom maps for the individual reaction patterns. They are visualised in Fig. 22 and Fig. 23.



(a) p_1, Pyranose-furanose **(b)** p_2, Furanose-linear

(c) p_3, Ketose-aldose

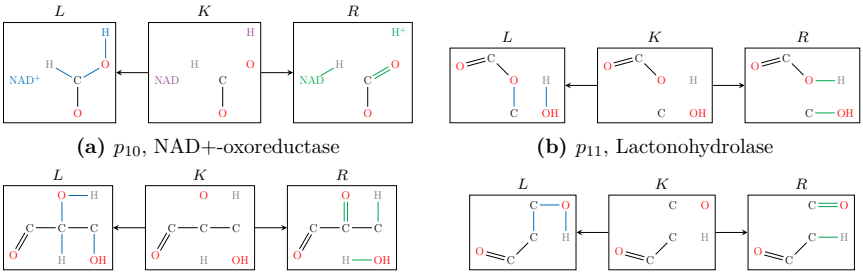
(d) p_4, ATP-phosphorylation

(e) p_5, ATP-dephosphorylation

(f) p_6, NAD+-phosphorylation

(g) p_7, Phosphomutase **(h)** p_8, Enolase **(i)** p_9, Keto-enol

Figure 22. Transformation rules for modelling the reactions in the EMP and ED pathways for glycolysis. In the modelling we have abbreviated certain groups into graphs with non-chemical labels. For example, part of phosphate groups are represented by vertices with the label `Pi`, and adenosine is represented by the label `AZP`. See also Fig. 23.



(a) p_{10}, NAD+-oxoreductase **(b)** p_{11}, Lactonohydrolase

(c) p_{12}, Hydrolyase **(d)** p_{13}, Reverse aldolase

Figure 23. Further transformation rules for modelling the reactions in the EMP and ED pathways for glycolysis. See also Fig. 22.

We use $G(EMP)$ to denote the graph of educts to the EMP pathway, consisting of 1 glucose, 2 ATP, 2 ADP, 2 phosphates, and 2 NAD^+. For the ED pathway we likewise have $G(ED)$, consisting of 1 glucose, 1 ATP, 1 ADP, 1 phosphate, and 2 NAD^+. Correspondingly we have $H(EMP)$ as the output graph of the EMP pathway, with 2 pyruvates, 4 ATP, 2 NADH, 2 water, and 2 H^+. In the case of the ED pathway $H(ED)$ models 2 pyruvates, 2 ATP, 2 NADH, 2 water, and 2 H^+. Note that the same approach as presented in the β-lactamase example (Sec. 6.1) for automatically inferring the necessary functional groups could be applied, and an explicit definition of the catalysts in $G(\cdot)$ and $H(\cdot)$ would not be necessary.

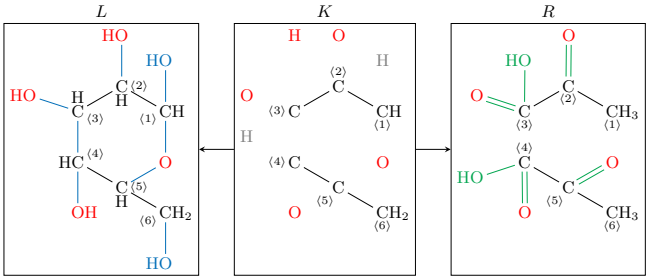
For EMP we compute the composition enumeration

$$\imath_{G(EMP)} \bullet \overbrace{p_4 \bullet p_1 \bullet p_4 \bullet p_2 \bullet p_{13} \bullet p_3}^{\text{Glucose} \to \text{2 G3P}} \bullet \underbrace{(p_6 \bullet_\emptyset p_6) \bullet (p_5 \bullet_\emptyset p_5) \bullet (p_7 \bullet_\emptyset p_7) \bullet (p_8 \bullet_\emptyset p_8) \bullet (p_5 \bullet_\emptyset p_5) \bullet (p_9 \bullet_\emptyset p_9)}_{\text{2 G3P} \to \text{2 Pyruvate}} \bullet \imath_{H(EMP)}$$

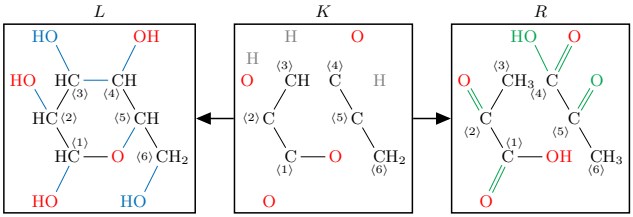
and for ED we compute

$$\imath_{G(ED)} \bullet \underbrace{p_4 \bullet p_{10} \bullet p_{11} \bullet p_{12} \bullet p_{13}}_{\text{Glucose} \to \text{G3P + Pyruvate}} \bullet \underbrace{p_6 \bullet p_5 \bullet p_7 \bullet p_8 \bullet p_5 \bullet p_9}_{\text{G3P + Pyruvate} \to \text{2 Pyruvate}} \bullet \imath_{H(ED)}$$

The resulting rules are depicted in Fig. 24.



(a) EMP



(b) ED

Figure 24. Simplified transformation rule for the overall a EMP pathway and b ED pathway, with each carbon atom labelled. The green and blue hydroxyl groups are those that are respectively removed and introduced through molecules other than glucose and pyruvate.

To reduce clutter we only draw the glucose and pyruvate components. Formally this can be achieved by composing with rules on the left and right that bind and unbind the unwanted components. Clearly, the carbon traces of the two rules differ. Such an approach can be used for an automated design of labelling experiments to detect the activity of pathway alternatives.

The prefixes of the rule composition expressions allow the inference of all the intermediate compounds and their corresponding atom traces relative to the input compounds. The summary of this analysis is depicted in Fig. 25 for both pathways, though only with the traces for carbon atoms shown.

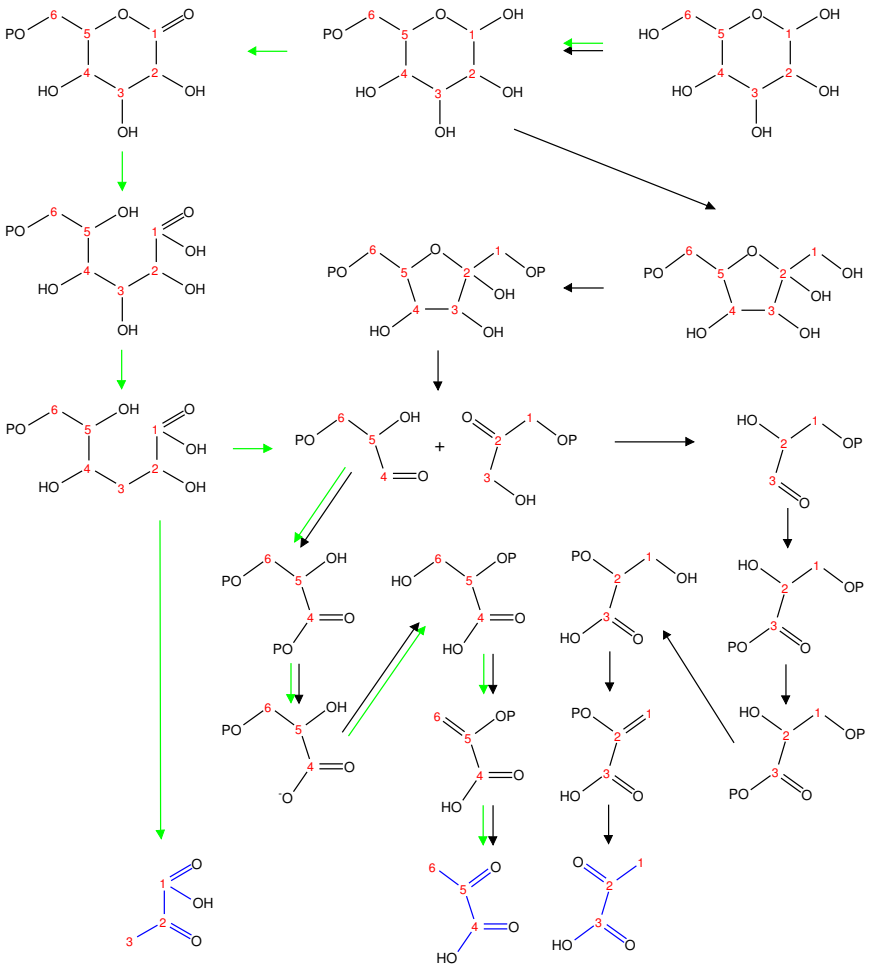


Figure 25. Carbon trace of two glycolysis pathways. The Embden-Meyerhof-Parnas pathway (EMP) is depicted with black reaction arrows, and the Entner-Doudoroff pathway (ED) is depicted with green reaction arrows. The six carbon atoms from glucose are converted into two pyruvate molecules, highlighted in blue, in two different ways depending on whether EMP or ED was used to catabolise glucose. The trace for one pyruvate overlaps in the pathways, while the sequence of carbons is inverted in the other pyruvate.

The black reaction arrows show the EMP pathway, while the green arrows show the ED pathway. The six carbon atoms from glucose are converted into two pyruvate molecules in two different ways depending on whether EMP or ED was used to catabolise glucose.

While the EMP pathway has a fructose 1,6-bisphosphate as an intermediate, in which a pentose ring is cleaved, in the ED pathway the hexose ring of the glucose 6-phosphate is cleaved. The carbon trace of one of the two pyruvates is identical, while it is inverted in the other pyruvate.

7 Concluding Remarks

The main purpose of this contribution was to provide a concise and complete description of rule composition in graph transformation systems. In particular, we have shown that rule composition as implemented in MØD is firmly grounded in category theory, and several special cases of composition have interesting semantics in the context of chemistry. This sets the stage for the analysis of complex overall reactions, since these have consistent representations as compositions of elementary reactions. The implementation in MØD is indeed capable of explicitly computing step-wise reaction mechanisms given a set of rules for plausible elementary steps. An important property of chemical rules is the explicit representation of atom maps, which derives from the restriction of morphisms to the vertex sets, in direct derivations. An immediate benefit is that composite reactions are also endowed with valid atom maps. As an application we have shown that this feature makes is possible to compute isotope traces explicitly. DPO graph transformation thus provide a framework in which otherwise difficult questions in computational chemistry can be dealt with easily and efficiently.

Still, many open questions remain for future developments. Although the DPO framework has been chosen to be particularly accommodating to chemical applications, at present the adherence to the semantics of chemistry is entirely left to the reader. While it does not seem desirable to specialise neither the mathematical specification nor the implementation of MØD further towards chemistry as a specific application, it would be extremely useful to develop a chemistry-specific layer to facilitate the construction of chemical rules by enforcing and/or checking conservation of mass, atom type, and charges that are required for chemical reactions. This could be achieved with the help of refined graph models that explicitly represent valence and bond electrons and their changes in a reaction so that chemical conservation laws correspond to graph invariants. We will pursue this issue in forthcoming work.

The mathematical structure of the transformation system also deserves further analysis. The composition of rules is not unique in general because different common sub-

graphs may exist. This begs the question whether rule composition is associative at the set level. An affirmative answer, which we conjecture, ensures for instance that efficient strategies for decomposing composite reactions into their elementary constituents are exhaustive. We will address this technical point in a forthcoming manuscript.

Finally, it is a continuing enterprise to develop a foundational computational framework for graph-based modelling of chemical systems. This for example includes the incorporation of stereochemical information, which we have only briefly alluded to [31]. It involves an extension of the vertex and edge labels with permutations groups for encoding local geometry, and an introduction of non-trivial stereo-morphisms taking these labels into account. In future work we will explore the details of this stereochemical extension in the context of rule composition.

Acknowledgements: This work was supported in part by the Volkswagen Stiftung proj. no. I/82719, the COST-Action CM1304 "Systems Chemistry", by the Danish Council for Independent Research, Natural Sciences, grants DFF-1323-00247 and DFF-7014-00041, and by the National Science Foundation (INSPIRE 1648973). It is also supported by the ELSI Origins Network (EON), which is supported by a grant from the John Templeton Foundation. The opinions expressed in this publication are those of the authors and do not necessarily reflect the views of the John Templeton Foundation.

Bibliography

[1] J. B. Hendrickson, Comprehensive system for classification and nomenclature of organic reactions, *J. Chem. Inf. Comp. Sci.* **37** (1997) 852–860.

[2] G. Benkö, C. Flamm, P. F. Stadler, A graph-based toy model of chemistry, *J. Chem. Inf. Comp. Sci.* **43** (2003) 1085–1093.

[3] G. Berry, G. Boudol, The chemical abstract machine, in: *POPL '90 – Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on principles of programming languages*, Assoc. Computing Machinery, New York, 1990, pp. 81–94.

[4] P. Dittrich, J. Ziegler, W. Banzhaf, Artificial chemistries — a review, *Artif. Life* **7** (2001) 225–275.

[5] W. Fontana, L. W. Buss, What would be conserved if "the tape were played twice"? *Proc. Natl. Acad. Sci. USA* **91** (1994) 757–761.

[6] A. Regev, E. Shapiro, Cells as computation, *Nature* **419** (2002) 343–343.

[7] V. Danos, Formal molecular biology, *Theor. Comp. Sci.* **325** (2004) 69–110.

[8] M. L. Blinov, J. Yang, J. R. Faeder, W. S. Hlavacek, Graph theory for rule-based modeling of biochemical networks, in: C. Priami, A. A. Ingólfsdóttir, B. Mishra, H. R. Nielson (Eds.), *Transactions on Computational Systems Biology VII*, Springer, 2006, pp. 89–106.

[9] G. Păun, Computing with membranes, *J. Comp. Syst. Sci.* **61** (2000) 108–143.

[10] L. Cardelli, Brane calculi, in: V. Danos, V. Schachter (Eds.), *Computational Methods in Systems Biology, CMSB'04*, Springer, 2005, pp. 257–278.

[11] C. Arnold, P. F. Stadler, S. J. Prohaska, Chromatin computation: Epigenetic inheritance as a pattern reconstruction problem, *J. Theor. Biol.* **336** (2013) 61–74.

[12] W. Hlavacek, J. R. Faeder, M. L. Blinov, R. G. Posner, M. Hucka, W. Fontana, Rules for modeling signal-transduction systems, *Science's STKE* **2006** (2006) 334–re6.

[13] J. A. Sekar, J. R. Faeder, Rule-based modeling of signal transduction: a primer, *Methods Mol. Biol.* **880** (2012) 139–218.

[14] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, M. Löwe, Algebraic approaches to graph transformation – Part I: Basic concepts and double pushout approach, in: G. Rozenberg (Ed.), *Handbook of Graph Grammars and Computing by Graph Transformation*, World Scientific, 1997, pp. 163–245.

[15] H. Ehrig, K. Ehrig, U. Prange, G. Taenthzer, *Fundamentals of Algebraic Graph Transformation*, Springer, Berlin, 2006.

[16] H. Ehrig, Introduction to the algebraic theory of graph grammars (a survey), in: V. Claus, H. Ehrig, G. Rozenberg (Eds.), *Graph-Grammars and Their Application to Computer Science and Biology*, Springer, Berlin, 1979, pp. 1–69.

[17] U. Sauer, Metabolic networks in motion: ^{13}C-based flux analysis, *Mol. Syst. Biol.* **2** (2006) 62–62.

[18] N. Zamboni, ^{13}C metabolic flux analysis in complex systems, *Curr. Opin. Biotech.* **22** (2011) 103–108.

[19] M. Durot, P.-Y. Bourguignon, V. Schachter, Genome-scale models of bacterial metabolisn: reconstruction and applications, *FEMS Microbiol. Rev.* **33** (2009) 164–190.

[20] A. M. Feist, M. J. Herrgård, I. Thiele, J. L. Reed, B. Ø. Palsson, Reconstruction of biochemical networks in microorganisms, *Nat. Rev. Microbiol.* **7** (2009) 129–143.

[21] V. Danos, J. Feret, W. Fontana, R. Harmer, J. Hayman, J. Krivine, C. Thompson-Walsh, G. Winskel, Graphs, rewriting and pathway reconstruction for rule-based models, in: D. D'Souza, T. Kavitha, J. Radhakrishnan (Eds.), *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012)*, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, 2012, pp. 276–288.

[22] L. Félix, F. Rosselló, G. Valiente, Efficient reconstruction of metabolic pathways by bidirectional chemical search, *B. Math. Biol.* **71** (2009) 750–769.

[23] J. L. Andersen, C. Flamm, D. Merkle, P. F. Stadler, Inferring chemical reaction patterns using rule composition in graph grammars, *J. Syst. Chem.* **4** (2013) 4.

[24] J. L. Andersen, C. Flamm, D. Merkle, P. F. Stadler, 50 Shades of rule composition: From chemical reactions to higher levels of abstraction, in: F. Fages, C. Piazza (Eds.), *Formal Methods in Macro-Biology*, Springer, Berlin, 2014, pp. 117–135.

[25] M. Schönfinkel, Über die Bausteine der mathematischen Logik, *Math. Ann.* **92** (1924) 305–316.

[26] H. Ehrig, A. Habel, H. J. Kreowski, F. Parisi-Presicce, Parallelism and concurrency in high-level replacement systems, *Math. Struct. Comp. Sci.* **1** (1991) 361–404.

[27] U. Golas, *Analysis and Correctness of Algebraic Graph and Model Transformations*, Vieweg+Teubner, Wiesbaden, 2010.

[28] B. Braatz, U. Golas, T. Soboll, How to delete categorically — two pushout complement constructions, *J. Symb. Comput.* **46** (2011) 246–271.

[29] A. Habel, D. Plump, Relabelling in graph transformation, in: A. Corradini, H. Ehrig, H. J. Kreowski, G. Rozenberg (Eds.), *Graph Transformation: First International Conference, ICGT 2002 Barcelona, Spain, October 7–12, 2002 Proceedings*, Springer, Berlin, 2002, pp. 135–147.

[30] G. N. Lewis, The atom and the molecule, *J. Am. Chem. Soc.* **38** (1916) 762–785.

[31] J. L. Andersen, C. Flamm, D. Merkle, P. F. Stadler, Chemical graph transformation with stereo-information, in: *Graph Transformation: 10th International Conference, ICGT 2017*, 2017, accepted.

[32] J. L. Andersen, MedØlDatschgerl (MØD), `http://mod.imada.sdu.dk`, 2016.

[33] J. L. Andersen, C. Flamm, D. Merkle, P. F. Stadler, A software package for chemically inspired graph transformation, in: R. Echahed, M. Minas (Eds.), *Graph Transformation: 9th International Conference, ICGT 2016, in Memory of Hartmut Ehrig, Held as Part of STAF 2016, Vienna, Austria, July 5-6, 2016, Proceedings*, Springer, Cham, 2016, pp. 73–88.

[34] L. P. Cordella, P. Foggia, C. Sansone, M. Vento, An improved algorithm for matching large graphs, in: *Proc. of the 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, 2001, pp. 149–159.

[35] L. Cordella, P. Foggia, C. Sansone, M. Vento, A (sub) graph isomorphism algorithm for matching large graphs, *IEEE T. Pattern Anal.* **26** (2004) 1367–1367.

[36] G. L. Holliday, C. Andreini, J. D. Fischer, S. A. Rahman, D. E. Almonacid, S. T. Williams, W. R. Pearson, MACiE: exploring the diversity of biochemical reactions., *Nucleic Acids Res.* **40** (2012) D783–D789.

[37] B. P. Atanasov, D. Mustafi, M. M. W., Protonation of the beta-lactam nitrogen is the trigger event in the catalytic action of class A beta-lactamases, *Proc. Natl. Acad. Sci. USA* **97** (2000) 3160–3165.

[38] A. Bar-Even, A. Flamholz, E. Noor, R. Milo, Rethinking glycolysis: on the biochemical logic of metabolic pathways, *Nat. Chem. Biol.* **8** (2012) 509–517.

[39] N. Entner, M. Doudoroff, Glucose and gluconic acid oxidation of pseudomonas saccharophila, *J. Biol. Chem.* **196** (1952) 853–862.

[40] I. Borodina, C. Schöller, A. Eliasson, J. Nielsen, Metabolic network analysis of streptomyces tenebrarius, a streptomyces species with an active entner-doudoroff pathway, *Appl. Environ. Microb.* **71** (2005) 2294–2302.