UNIVERSITÄT ZU LÜBECK

Graph Grammar Slicing

An Approach to Reducing the Combinatorial Complexity of Tracking Observables in Complex Reaction Systems

Albert Krewinkel

Master's Thesis supervised by

Prof. Dr. Till Tantau
Institute of Theoretical Computer Science

Prof. Dr. Amir Madany Mamlouk
Institute of Neuro- and Bioinformatics

March 2011

Abstract

Rule-based models are widely adopted among system biologists since such models enable terse descriptions of molecular pathways. The combinatorial complexity inherent to molecular systems, stemming from the numerous interactions and formation of molecule complexes, is handeled using general rules to describe all possible reactions. This increases the expressive power available to model builders, yet leaves the total complexity of a model unchanged: The application of the rules on a specific set of start molecules currently requires full calculation of all distinct tranformations and graphs reachable in this system, regardless of what part of the system is of interest.

In this thesis, a slicing technique for graph transformation systems is developed and application in biological context is discussed. Graph-based models of chemical and biochemical reaction networks are given. It is demonstrated that a slice of a chemical graph grammar reproduces the behavior of the original grammar with regard to a subgraph. Biological applications and possible extensions of this approach are discussed.

Declaration

I hereby declare that this thesis submitted for the degree *Master of Science: Computational Life Science* at the University of Lübeck is my own original work and has not previously been submitted to any other institution of higher education. I further declare that all sources cited or quoted are indicated and acknowledged by means of a comprehensive list of reverences.

Albert Krewinkel

Acknowledgements

I wish to express my gratitude to Aneil Mallavarapu, who initiated this research and gave me the opportunity to work on the project. Working with him was a most enjoyable and inspiring experience which I am truely grateful for.

I thank my supervisors Till Tantau and Amir Madany Mamlouk for giving me the freedom to pursue a research topic of my own choice while offering support and constructive criticism. Their excellent input and their far reaching help was beyond what I had hoped for and is highly appreciated.

The people at SRI International provided a pleasant, stimulating and creative atmosphere during my stay there. I especially thank Carolyn Talcott for including me into her group at the computer science laboratory and Mark-Oliver Stehr for his introduction to formal methods and category theory.

Likewise, I am grateful to Lis Pohl for the support provided with categorical problems. The fruitful discussions greatly advanced my understanding of the essential prerequisites for this research and furthered completion of this thesis in its later phase.

Thanks are also due to Ronny Bergmann for discussing my theoretical concepts. Furthermore, his typographic advise resulted in considerable improvements of the layout.

Finally, I would like to express special thanks to S.D. for proofreading preliminary drafts of this thesis and for her support and encouragement.

Contents

1 Introduction

The knowledge and understanding of biochemical pathways and the interactions within is a cruicial prerequisite for further advancement of the life sciences. The qualitative knowledge of biochemical pathways has been increasing rapidly within the last decades. However, this is in crass contrast to the still restricted use of mathematical and computational tools to understand living systems.

In this thesis a method is developed to modularize graph-based models of chemical reaction and protein–protein interaction networks. The graph grammar slicing approach used for this allows to reduce the size of a reaction network generated from a model, with consequences for model building and numerical simulations in systems biology and artificial chemistry.

1.1 Background

During the recent years, we have witnessed a vast growth of biological information publicly available in databases and currated collections. This gain of information is not limited to genomic sequences and protein structures, but also leads to increasingly high detail databases on metabolic pathways [13, 41, 43]. However, the raw data in such databases must be interpreted and put into context to gain new insights into the inner workings of biological systems like cells, organs or whole organisms.

The research field which aims at a holistic understanding of these entities is generally referred to as systems biology [62]. This includes the development of mathematical methods to analyse complex systems and computational representations and algorithms to process the available data necessary for simulations and valueable predictions.

In this section, a brief overview is given describing the components which make up the systems under consideration in this thesis.

1.1.1 Chemical and Biochemical Reactions

A chemical reaction in general is the process in which the electron configurations of one or more molecules are changed to yield new molecules. Chemical reactions may be coupled for the synthesis of newly designed substances bearing desired properties.

Chemical reactions in biological systems rarely involve only small molecules but depend on enzymes or other large proteins. These biochemical reactions are thereby highly specific: The tertiary structure of proteins restrict the number of potential reaction partners, especially in enzymes.

Biochemical reactions are highly interconnected, forming a network within the cell. The network can be thought of as a graph consisting of molecular species and reactions transfering between species. Enzyme kinetics, regulating the rate at which species react, are often given using Michaelis-Menten kinetics, but can also be specified using basic kinetics of elementary chemical reactions, that is mass-action kinetics.

1.1.2 Cellular Signal Transduction Pathways

Information from the outside of the cells is processed in signal transduction pathways: A signal from outside the cell is picked up on the membrane by a receptor and is passed on to inner parts of the cell by small molecules and interacting proteins. The main processing of the signal is performed by protein–protein interaction networks. These intracellular networks result in the modification and activation of proteins and the formation of protein complexes. The end-points of the cascade are transcription factors, affecting the rate of sequence transcription and therefore the kind and concentration of proteins in the cell.

The activity and interaction capability of a protein is determined by its inner state, which is set by small modifications to key-parts of the protein. The most common modifications are phosphorylation of a serine, threonine or tyrosine residues altering the proteins tertiary structure and therby increasing its enzymatic activity. The MAP kinase pathway is a well known example of cellular signal transduction pathways, featuring many examples for protein activation by means of amino acid phosphorylation.

1.1.3 Combinatorial Explosions

The number of protein complexes occurring in a cell is tremendous and even within a fairly restricted signal transduction pathway, it may be unmanageable for computational simulations [17, 22]. There is a multitude of protein types involved in the pathways and the internal states of a protein as well as the multitude of possible interactions increase the number of differentiable complexes. Consequently, even moderately sized models of such pathways quickly result in simulations requiring considerable computational power to execute.

For example, linear formation of protein complexes results in quadratic number of possible complexes, and the number of protein configurations grows exponentially with the number of modification sites.

However, it often is possible to take a macroscopic view of a system, such that on a molecular level distinguishable complexes are identified on the macroscopic level. In this thesis, a new such method is described for graph-based models of proteins.

1.2 Aims and Contributions

The aim of this thesis is to further the understanding of modularity and model reduction in graph-based models in the special case of graph grammars.

Formal and graph-based models are given for low-molecular chemical components as well as for proteins. The chemical model includes information on the level of atoms and electrons, while the model for protein complexes is based on protein domains and their interactions. Both models are developed and described in Chapter 2. While such models have been designed before, this thesis emphasises the similarity of the models on a mathematical level, thereby demonstrating that methods developed for artificial chemistry may also be applied in systems biology and *vice versa*.

Graph grammar slicing is introduced as an approach to generate rewrite systems which contain the same as the original system on a more abstract level while ignoring unnecessary detail. The relationship of the constructs introduced here to similar constructs described by other authors is discussed.

1.3 Thesis Outline

We start in Chapter 2 with a brief overview of the required basic mathematical tools and continue with the description of graph-based models for small molecules and their reactions. This is followed by a model for protein–protein interaction networks. Mathematical tools used in the models will be described as we go along.

In Chapter 3, graph grammar slicing is developed as mathematically sound approach to modify a rewrite system such that its behavior is unchanged with respect to specific parts of its components.

The thesis ends with conclusions and an outlook to further work.

2

Graph-Based Models of Reaction Systems

Graphs are not only an expressive and flexible way of describing real world entities, but also mathematical objects. Models using graphs are found throughout most scientific fields, and the number of tools and methods available to build and analyze these models is vast. These favorable properties stay valid in recent application areas of graph theory, such as biochemistry, metabolomics and artificial chemistry. Here we will focus on simple graph-based models in these areas. Graph rewriting is introduced as a formal yet intuitive way to model dynamical changes.

First we will discuss how graphs are used in artificial chemistry, and how molecules and reactions can be modeled by graphs and graph rewrite rules, respectively. General advantages and problems of graph rewriting will be discussed here.

A shift in the abstraction level leads from simple organic molecules to more complex structures: Graph-based models of proteins and protein complexes enable rule-based descriptions of biochemical processes. We focus on graph rewriting in the context of protein–protein interactions and discuss how theoretical constructs such as graph-typed graphs can help to validate the results of graph rewriting steps.

In the last section we discuss other common methods used to model biochemical processes, most notably Petri nets and process calculi.

2.1 Graphs and Graph Morphisms

A *graph* is a tuple $G = (V_G, E_G, s_G, t_G)$, where V_G is a set of *vertices*, E_G is a set of *edges* and $s_G, t_G \colon E \to V$ are the *source* and *target* functions.

2.1.1 Graph Morphisms

A graph morphism is a mapping between two multigraphs, respecting the structure of the graphs. Graph morphisms preserve connectivity in that two vertices, which are connected in the domain of the morphism, must also be connected in the the same way in the morphism's image.

Given multigraphs G and H, a *graph morphism* $f \colon G \to H$ is a pair of functions $f = (f_V \colon V_G \to V_H, f_E \colon E_G \to E_H)$ such that sources and targets are preserved, that is, $f_V \circ s_G = s_H \circ f_E$ and $f_V \circ t_G = t_H \circ f_E$.

In this work, the main focus will be on injective and bijective mapping functions, *graph monomorphisms* and *graph isomorphisms*, respectively. These types of morphisms conveniently capture the idea of "partially similar" and "equivalent" graphs.

We say a graph G is a *subgraph isomorphic* to H, and write $G \preceq H$, if there is a monomorphism $f \colon G \to H$ from G to H. The graphs G and H are called *isomorphic*, written $G \equiv H$, if the monomorphism f can be inverted such that $f^{-1} \circ f = id_G$.

2.1.2 Typed Graphs

Typed graphs are an extension to the usual labeled graphs. Vertices and edges of a labeled graphs are colored with elements of a color set, while in typed graphs they are mapped to a *graph of types*. Graph morphisms are easily extended to typed graphs.

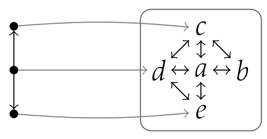
A graph T consisting of vertex types V_T and edge types E_T is called a *graph of types*. A *typed graph* is a tuple $G = (\tilde{G}, \tau_{\tilde{G}})$, where $\tilde{G}$ is a graph, T is a graph of types and $\tau_{\tilde{G}} \colon \tilde{G} \to T$ is a graph morphism.

Let $G = (\tilde{G}, \tau_{\tilde{G}})$ and $H = (\tilde{H}, \tau_{\tilde{H}})$ be typed graphs. A T-typed graph morphisms $f \colon G \to H$ is given by a graph morphism $\tilde{f} \colon \tilde{G} \to \tilde{H}$ consistent with typing, that is, such that $\tau_{\tilde{G}} = \tau_{\tilde{H}} \circ \tilde{f}$.

Example 2.1 The graph

$$T = \begin{tikzcd}[row sep=tiny, column sep=tiny] & c \arrow[d, leftrightarrow] \arrow[dr, leftrightarrow] \arrow[dl, leftrightarrow] & \\ d \arrow[r, leftrightarrow] \arrow[dr, leftrightarrow] & a \arrow[r, leftrightarrow] \arrow[d, leftrightarrow] & b \\ & e & \end{tikzcd}$$

is a type graph having just one edge type for each pair of vertices. In this thesis, graphs typed over T, for example



will not be drawn in full detail. Instead, nodes are replaced by their respective vertex type, such that the above typed graph will be depicted

$$\begin{tikzcd}[row sep=tiny] c \\ d \arrow[u] \arrow[d] \\ e \end{tikzcd}$$

◆

The models in this chapter are based on typed graphs, so we will refer to graphs as explicitly as *untyped graphs* and use the term *typed graph* synonymiously to graph if no confusion about the meaning of the term is possible.

2.2 Graph-Based Artificial Chemistry

Artificial chemistry is generally understood as a theoretical or computational system that is similar to a real chemical system. A common formal definition of artificial chemistry is based on a set of molecules S, a set of molecular interaction rules R and an algorithm A specifying the conditions of how and when the interactions occur [28]. A multitude of artificial chemistry systems has been developed, offering different representations of molecules and reactions. In the context of biochemical reactions involving protein–protein interactions, graph-based representation of molecules are of particular interest since they are expressive, straightforward to develop and easy to understand for mathematicians and biologists alike [21, 48].

2.2.1 Modeling Molecules as Graphs

Structural formulæ are the most common method of describing molecules, especially in organic chemistry. Labeled graphs are closely related to structural formulæ and therefore are a natural choice for the representation of molecules.

The following formalization is based on a Toy Model of chemistry, introduced by Benkö et al. [5] in 2003 as a framework to explore properties of large chemical networks. This approach features the potential to perform energetic calculations as well as an artificial chemistry model based on graph rewrite rules. The latter aspect is elaborated on in the next section.

Formally, a graph-based model of molecules uses vertices to represent atoms, while covalent bonds become edges connecting two vertices. All vertices and edges in the graph are typed over a type graph specifying the types of atom orbitals and bonds, respectively.

It should be noted there is no restriction on the number of edges between two vertices, allowing two or more parallel edges. Hence, we need additional restrictions to ensure that each graphs corresponds to a structure potentially produced by a chemical system. For example, each vertex that is typed as a sp^3 hybridized carbon atom should have exactly four bonds connecting it to its neighbors.

The exact conditions that mark a graph as a representation of a molecule molecular graph will depend on the details of the model. The following definition for molecule representing graphs is intentionally left unspecific for that reason.

Definition 2.1 (Molecular Graph) Given a set of properties, let T be a type graph of atoms and orbitals and $\mathbf{G_T}$ be a class of graphs typed over T. The set of graphs $\mathcal{M}$ in $\mathbf{G_T}$ for that all properties are true is the set of *molecular graphs*.

The three-dimensional configuration and similar information is not included in this graph-based model. Instead, the molecular structure is reduced to atoms and connectivity. However, this loss of information often does not seriously influence the predictional reliability of qualitative models involving small molecules [4].

2.2.2 Modeling Reaction Rules as Graph Productions

The focus of models for artificial chemistry is the simulation and analysis of chemical reaction systems. In the graph-based model, chemical reaction rules are modeled as graph productions acting on molecular graphs [4, 5].

Chemical reactions given by structural formulæ frequently specify only the relevant parts of the reacting molecules. This is reflected in the graph-based abstraction in that rules may contain partial molecular graphs. Partial molecular graphs do not directly correspond to molecules found in the reaction system, but represent a *substructure* of molecules.

Substructures correspond to subgraphs in graph-based models. They are used as means to define premises and changes of a reaction. Current approaches use the double pushout approach to graph rewriting, see Corradini et al. [19] for details on the basic concepts.

A *production* is a pair $(p\colon L \xleftarrow{l} I \xrightarrow{r} R)$, where p is the productions *name* and the span $L \xleftarrow{l} I \xrightarrow{r} R$ consists of a pair of injective graph morphisms $l\colon I \to L$ and $r\colon I \to R$. The graphs L, I and R are called the *left hand side*, *right hand side* and *interface* of the production, respectively. If both morphisms l and r are monic, then p is *linear*. Often we will reference to a production $(p\colon L \xleftarrow{l} I \xrightarrow{r} R)$ simply by its name p if there is no possible confusion of its meaning.

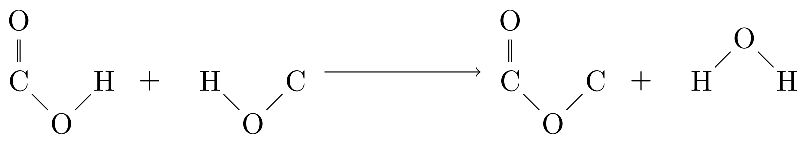
In chemical terms, the left hand side of a production corresponds to those parts of a molecule that are necessarily present in an educt for the reaction to take place. The name of a production will typically be linked to a reaction rate, but it may also carry additional information.

Distinct components of a molecule stay distinct throughout a reaction. This is ensured by restricting the class of morphisms in a production to monomorphisms, so productions modeling chemical reaction rules must be linear. Furthermore, chemical reactions are reversible:

A production $\pi = (p\colon L \xleftarrow{l} I \xrightarrow{r} R)$ is *reversible* if there exists an inverse name p^{-1}. The production $(p\colon L \xleftarrow{l} I \xrightarrow{r} R)^{-1} = (p^{-1}\colon r \xleftarrow{l} I \xrightarrow{r} l)$ in which the left- and right morphisms are exchanged is the *inverse of* π and we say that π *is invertible*.

Definition 2.2 (Chemical production) A production is *chemical* if its linear and reversible.

Example 2.2 Below is a rewrite rule specifying an esterification reaction.



◆

2.2.3 Modeling Chemical Reaction Systems as Graph Grammars

Graph productions by themselves are insufficient to describe a dynamical system like reaction networks in a cell or a chemical reactor. A system model is typically comprised of a start configuration and the productions acting on the graph. This determines the potential behavior of a graph transformation system and leads to graph grammars as a mathematical model. Graph grammars are the prevalent construct used in graph transformation research and are also suitable for chemical system models.

Definition 2.3 (Graph grammar) A *typed graph grammar* is a tuple $\mathcal{G} = (T, G_0, P, \Pi)$ consisting of a type graph T, a *start graph* G_0, an index set of production names P and a set of productions $\Pi = (p : L_p \leftarrow I_p \rightarrow R_p)_{p \in P}$ indexed over P.

Application of a production on a graph is handeled using direct derivations, formulated here in a category theoretical way. A production may only be applied on a graph when the result is uniquely determined. For more details on the construction of pushouts and pushout complements in the category of graphs, see for example Corradini et al. [19].

Given a production $\pi = (p\colon L \xleftarrow{l} I \xrightarrow{r} R)$, the *dangling condition* is satisfied for a match $m\colon L \to G$ if the pushout complement of the composable arrows $I \xrightarrow{L} B \xrightarrow{m} G$ exists.

Let G be a graph, $\pi = (p\colon L \xleftarrow{l} I \xrightarrow{r} R)$ a production and $m\colon L \to G$ a morphisms (also called a *match*). A *direct derivation from G to H using p based on m*, written $G \overset{p,m_L}{\Longrightarrow} H$, is the given by a diagram

$$\begin{tikzcd}
L \arrow[d, "m_L"'] & I \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "m_I"] & R \arrow[d, "m_R"] \\
G & C \arrow[l, "g"] \arrow[r, "h"'] & H
\end{tikzcd} \tag{2.1}$$

such that all squares are pushouts.

A chemical reaction is modeled by the application span of a direct derivation using a chemical production. The graphs G, H and I in the application span of diagram (2.1) correspond to the educts, products and the unchanged context of a chemical reaction, respepectively.

Coupled reactions and pathways are given by sequential derivations. The products of one reaction are regarded as the educts of the next reaction, linking arbitrary numbers of reactions as long as they can be applied in sequence.

A *sequential derivations* (over $\mathcal{G}$) is either a sequence of direct derivations $\rho = \{G_{i-1} \overset{\pi_i,m_i}{\Longrightarrow} G_i\}_{i\in\{1,\ldots,n\}}$ with $\pi_i \in \Pi$ or the identity derivation given by a graph G. In cases where we are not interested in the intermediate steps of the sequential derivation, we will write $\rho\colon G_i \Rightarrow^*_{\mathcal{G}} G_n$. The *language* of a graph grammar $\mathcal{G}$, denoted $\mathcal{L}(\mathcal{G})$, is the set of possible ending graphs of a sequential derivations from the start graph,

$$\mathcal{L}(\mathcal{G}) = \{G_n : G_0 \Rightarrow^*_{\mathcal{G}} G_n\}$$

Often, the details of a derivation are of no concern to us. We then ignore the production span and its matches, focusing on the resulting span of the derivation which contains the pushout objects.

If ρ is a sequential derivation as above, the sequence $\mathrm{app}(\rho) = G_i \xleftarrow{g_{i+1}} C_{i+1} \xrightarrow{h_{i+1}}$

$G_{i+1} \leftarrow \ldots \xleftarrow{g_n} C_n \xrightarrow{h_n} G_n$ is called the *application sequence* of ρ (or *application span* if the sequence is of length 1).

Graph grammars allow us to describe a dynamical system and give us a model for the reactions which are possible within the given setup. We use this scheme as the base for our discussions in the remainder of this thesis. For variations and alternative formal descriptions of graph-based models of artificial chemistry see for instance Rosselló and Valiente [57] and Kerber et al. [42].

2.3 Graph-Based Abstractions for Protein–Protein Interactions

Proteins are large and complex molecules, consisting of peptide chains with up to several hundreds or thousands of amino acids monomers in length. This complexity is furthered by additional secondary and tertiary structure, which is essential to the proteins function within a cell. This renders understanding and modeling proteins difficult.

Molecular graphs are best suited for the analysis of small proteins and their respective interactions. Their level of detail on the atomic level renders molecular graphs close to useless in the analysis of cellular signaling networks. Different, suitable reductions must be used in the modeling process in order to enable analyses on higher levels.

The base for protein-level models is based on the observation that unlike general reactions, protein–protein interactions depend only on few and very specific parts of the protein. The process of a protein domain being bound or the occurrence of a phosphorylation- or glycosylation event is sufficient to state conclusions about the biochemical properties of the protein.

Like for molecular graphs defined in the previous section, our model for proteins is based on typed graphs. We define the type graphs suitable for protein modeling and then explicitly specify what constraints must be satisfied in such model.

2.3.1 Type Graphs for Protein Modeling: Domain Interaction Maps

Most interactions between proteins are very specific: Not only the types of proteins capable of interactions is limited, but interactions are also associated with certain domains withing the proteins. These interaction properties are captured by a domain interaction map. (Some authors use the term *contact map* instead, but we avoid this term due to its bioinformatical meaning which is unrelated to the construct we are using here.)

All node types and possible interactions occur exactly once in a domain interaction map. This property is exploited by our model for protein complexes, in which protein complexes are modeled as graphs typed over a domain interaction map. Additionally to proteins, sites and their interactions, we include internal states of sites to allow for easy modeling of configuration changes and amino acid phosphorylation.

Definition 2.4 (Domain interaction map) Let P, S and A be disjoint sets whose elements are called *protein types*, *site types* and *internal states*, respectively. A graph $D = (V_D, E_D, s_D, t_D)$ is called a *domain interaction map* if its vertices are the union of P, S and A.

Example 2.3 The epidermal growth factor (EGF) receptor (EGFR) signaling is involved in the regulation of numerous cellular processes and plays a crucial role in the development of many cancer types [51]. It is among the best-investigated signaling systems. A model of the early events of EGFR signaling [7] has been established as a semi-conventional example for rule-based models of protein–protein interactions.

EGFR is a trans-membrane receptor which, upon binding of its ligand EGF, dimerises and is consequently autophosphylated. The phosphorylated EGFR tyrosin residues interact with other intracellular proteins and thereby signal the EGF binding event to other parts of the cascade. This motivates to use the following interaction sites to describe EGFR molecules: A ligand-binding site l, a receptor-dimerisation site d and two tyrosine sites named after the respective amino acid positions, $Y48$ and $Y68$. Interaction sites of the other proteins are represented using similar considerations.

See Figure 2.1 for a domain interaction map of the early events in the EGFR signaling model by Blinov et al. [7]. ◆

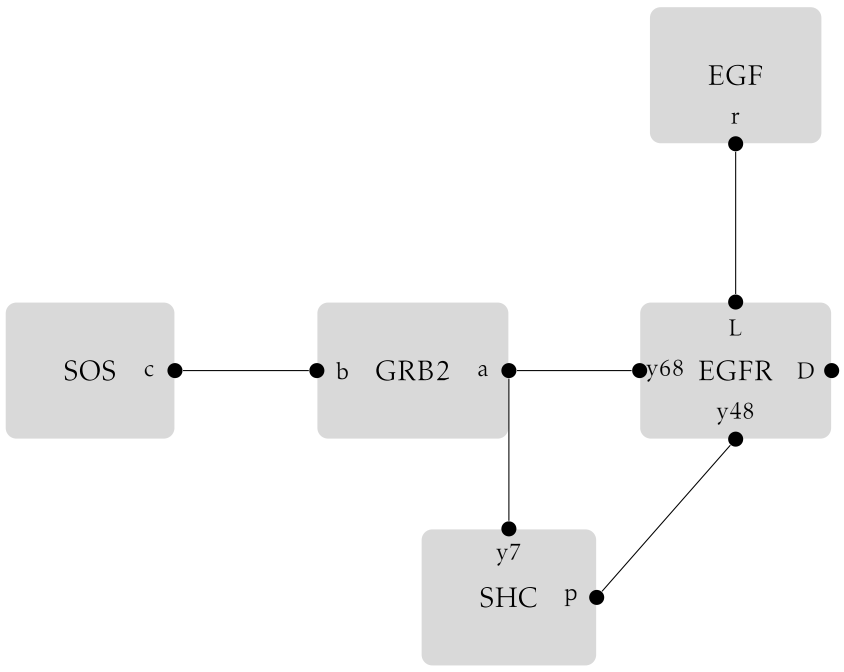


Figure 2.1: A domain interaction map for the early EGFR signaling model [7].

2.3.2 Modeling Protein Complexes as Graphs

The focus on modification sites, domains, and internal states of proteins greatly reduces the size of graphs used in the models compared to molecular graphs used in artificial chemistry. The model presented here is in concert with most of the approaches used in the literature (see for instance [8, 21, 36]). Included is information on protein–protein binding sites and on amino acids which are subject to modifications like phosphorylation or glycosylation. However, the details of the graph-based protein models vary; to the best of the authors knowledge, the use of constrained, typed graphs to model protein complexes has not been described yet.

We now define protein complexes as interacting protein graphs. Simple graph typing over a domain interaction map is very permissive, motivating additional constraints on protein-complex graphs.

Definition 2.5 (Protein complex graph) Given a domain interaction map D, a *protein complex graph* P is a D-typed graph such that the following conditions are met:

- PC1 Protein interaction is limited to sites: The neighborhoods of two protein nodes p_1, p_2 in P are disjoint, i.e. $N(p_1) \cap N(p_2) = \emptyset$
- PC2 Binary site interactions: Each site node in P is connected to at most one further site node.
- PC3 Connectedness: P is connected.

The definition ensures that protein-complex graphs are build in a sound way using protein types, site types and internal states. There are no edges connecting two protein nodes, protein interaction is only possible through interaction sites.

2.3.3 Modeling Protein–Protein Interactions as Graph Rewrite Rules

Graph-based modeling of protein–protein interactions is highly related to artificial chemistry and shares the idea of using graph rewriting to represent (bio)chemical reactions: Protein complexes are represented by protein-complex graphs, and modifications to these graphs are specified by graph

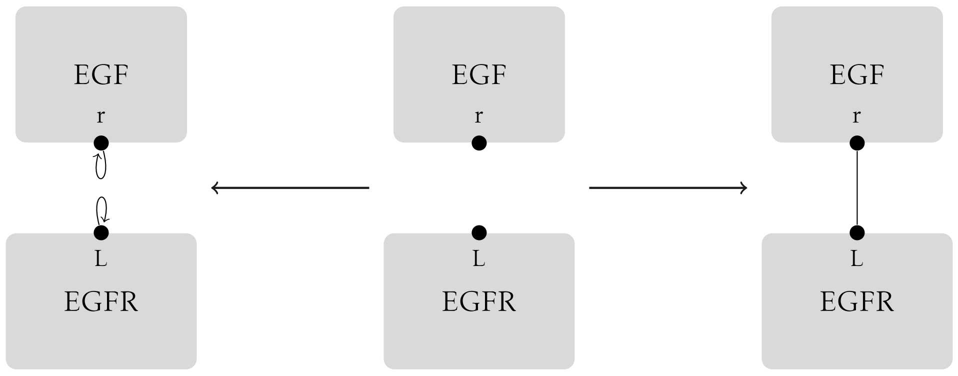


Figure 2.2: Example of a protein-interaction rule: EGF receptor binding.

transformation rules. While most authors in the field use single pushout rewriting [8, 21, 33], we content ourselves with the slightly more restrictive case of double pushout rewriting [19]: We find the additional generality of the single pushout approach unnecessary, and the double-pushout approach both sufficient in power and ease of understanding. Additionally, single pushout rewriting allows deletion processes that are unnatural in the context of molecular biology.

Definition 2.6 (Protein interaction rule) A *protein interaction rule* is a graph rewrite rule $L \xleftarrow{l} I \xrightarrow{r} R$ such that L and R are protein complex graphs.

Example 2.4 Figure 2.2 depicts a rule that expresses the binding of an EGF ligand to the extracellular part of the receptor. The binding process is assumed to be independent of the binding states of the intracellular sites. Therefore, the site nodes of the internal domain can be omitted in the rewrite rule description of the binding process. ◆

2.4 Integration of Quantitative Parameters

A main goal of biological systems modeling is the gain of new insights on the internal dynamics of the system. The dynamics are determined by concentrations, reaction rates and other numerical parameters which have not been considered yet. In order to enable quantitative analysis and simulation of the systems, these values must be an integral part of the models.

Augmenting rule-based models with numerical parameters is straightforward, and was already mentioned in the definition of productions for chemical systems (see Section 2.2.2 on page 9). Each rewrite rule in the model, or production in our graph grammar terminology, has a parameter assigned determining how often or how fast the rule is executed within the system. The type of the parameter, reaction constant, rate or probability, may differ depending on the intended type of simulation.

Simulation of systems becomes increasingly important the more complex the examined system is; simulations are often possible even when the analytical methods fail. We describe two common techniques of reactive systems simulation.

2.4.1 Simulations Using Ordinary Differential Equations

A common and simple simulation technique is based on mass action kinetics and elementary reactions. An elemental reaction occurs in a single step and without intermediate products [58]. The rate of an elementary reaction is determined by mass action kinetics, that is the rate is the product of educt concentrations and a single reaction constant. For a unimolecular reaction $A \to$ products, the rate is given by

$$\frac{d[A]}{dt} = -k[A]$$

and for a bimolecular reaction $A + B \to$ products, the rate is

$$\frac{d[A]}{dt} = \frac{d[B]}{dt} = -k[A][B]$$

where k is the reaction constant $[A]$ and $[B]$ are the concentrations of molecules A and B, respectively (for details, see for example [49]).

Assuming all rewrite rules in a model represent elemental reactions and have reaction constants assigned, it is possible to generate a system of ordenary differential equations for numerical simulations. The algorithms for this was first described by Blinov et al. [8] in 2006. Given a set of molecules which make up the start solution, all possible direct derivations are determined and the corresponding differential equation is generated for each direct derivation using mass action kinetics. The resulting sytems can then be analysed computationaly using standard tools of numerical integration.

This method is used by many rule-based biochemical modeling tools, including `BioNetGen` [6], `little b` [47] and `Bio-PEPA` [14]. The algorithm may be computationally expensive due to the need for repeated solution of the subgraph isomorphism problem [8] and the possibly large number of reactions [22] (see also Section 2.6). However, complexity can be tamed using reduction techniques as described in Section 2.6.2. Once a reasonably compact ODE systems has been generated, it allows rapid exploration of systems dynamics.

2.4.2 Simulations Using Stochastic Approaches

Stochastic simulations are an alternative to the deterministic simulations lined out in the previous section. The Gillespie algorithm [39] is a classic stochastic method to generate trajectories of molecule concentrations within a reaction system. Like for ODE-based simulations described above, reactions must be elementary in order for the original algorithm to produce correct results. However, extension of the algorithm have been developed such that the restriction to mass action kinetics is lifted and more complex kinetics, like the biochemical important Michael-Menten kinetics, may be used [11, 54].

Another interesting approach are continuous time Markov chains, modeling concentrations as discrete quantities. This allows for probabilistic model checking and numerical simulations which converge to the ODE case for increasing numbers of states [10].

For both simulation types there exist modeling tools with support for the respective method, for example `BioNetGen` [6] for the Gillespie algorithm or `Bio-PEPA`, which supports all types of simulations mentioned above.

2.5 Related Modeling Approaches

Biological systems are very complex and difficult to analyze. There is no single modeling framework suitable for each one of the vast numbers of current research questions. It is therefore not surprising that a multitude of models have been developed to address biological problems through model building. Appart from graph-based models, other approaches to biological system modeling include Petri nets, process calculi and multiset rewriting.

Petri nets and multiset rewriting are strict subsets of graph transformation systems.

2.5.1 Petri Nets

Petri nets are a well understood and widely used modeling tools for distributed systems. There exist a broad range of programs and utilities to build and analyze Petri Nets. Consequently, Petri nets are a natural choice for the description of biological systems [40, 55].

Some of the restrictions of plain Petri nets are lifted by various extensions which add additional properties to the models. Examples include continuous, colored and hybrid Petri nets [27]. Such generalized Petri nets are frequently found to be useful for application in biology as well. For example, continuous time Petri nets [26] can be used as a means to add reaction rates to transitions [38].

It is worth noting that Petri nets may be encoded using typed graph grammars [18].

2.5.2 Multiset Rewriting

The concept of multiset rewriting is similar to graph rewriting, but objects in a multiset are not structured by edges. Like Petri nets, which are a special case of multiset rewriting, the method is highly used in computer sciences, especially for analysis of security protocols. Nonetheless, multiset rewriting has also become an established method to investigate properties of cellular systems.

Multiset rewriting enables formal analyses of reaction systems, including reachability analysis and model checking [35]. A software solution for cellular systems modeling based on multiset rewriting is, for example, `Pathway Logic` [35, 59].

2.5.3 Process Calculi

Another approach to modeling transfered from adapted from computer science into biology is given by process calculi which were designed as formal models for concurrent systems.

A multitude of process calculi have been used or newly defined for biological systems, including the π calculus [56], the κ calculus [21] the ρ_{pg} calculus [2, 3], brane calculi [12] and Bio-PEPA [14].

2.6 The Challenge of Combinatorial Explosion in Reaction Systems

The term of combinatorial explosion in reaction systems referes to the rapid growth of reactions and molecular species even if the number of reactions or start conditions are increased only slightly. Due to the nature of reactions and protein–protein interactions, combinatorial explosion is a feature observed so frequently in reactive systems that it may be regarded as an inherent property of such.

2.6.1 Combinatorial Complexity in Protein–Protein Interaction Networks

For protein–protein interactions, consider the number protein complexes which may be formed from a set of proteins. In the simple case of independent reactions and where the protein complexes take the form of a chain, the number of possible complexes is given by the number of substrings of the longest chain. If there are n proteins interaction to form a chain in which each protein occures exactly once, there exist $\frac{n^2+n}{2}$ different protein complexes.

An even more drastic increase of possible species occures in proteins with multiple interaction sites. Given, for example, a protein having m phosphorylation sites where each site may be phosphorylated or dephosphorylated independently of the other sites, then there are 2^m possible configurations for this protein.

Even moderatly sized models of biochemical systems with a few dozend rules easily result in billions and more of possible complexes [22]. Such huge number of species renders simulations of the system difficult if not practicaly impossible. Therefore, the systems must be simplified, approximated or otherwise reduced to limit the number of species.

2.6.2 Methods for Model Reduction

Numerous model reduction methods for biochemical systems have been published recently which all share the same basic idea: A decrease in the level of detail, in order to yield a macroscopic view of observable properties, reduces the size of a model without changing its properties with respect to these observables. This is a procedure called *exact model reduction* or *coarse graining*.

A reduction method for systems of ordinary differential equations (ODEs) was developed by Conzelmann et al. [16] in 2006 (see also [15, 17]). Considerations from control theory and dependencies between protein domains are used to reduce the number of differential equations in hierarchical models while preserving the macroscopic properties of the system.

The domain-oriented reduction described by Borisov et al. [9] is a similar to above method and also based on dependencies between protein domains. It allows for automatic reduction of models given in the graph-like `BioNetGen` language, yet produces incorrect results for indirect dependencies.

An approach to model reduction which works also for non-hierarchical systems and indirect dependencies has been developed by Feret et al. [37] in 2009. Models given in the κ calculus [21] are reduced automatically while the ODE sematics of the system are preserved [23, 25, 37].

3 Graph Grammar Slicing

I introduce graph grammar slicing as an approach to limiting a graph grammar to only as little information as necessary to follow the evolution of a graph pattern during derivations. The principles of graph grammar slicing are inspired by those of *program slicing* [61], a technique developed to debug and analyze computer programs: Program slicing reduces a program to only those instructions that affect a single variable or a set thereof. Various variants of program slicing exist, including dynamic and static slicing [60]. While the former is applied on source code without further information on the input, dynamic slicing also takes the input into account. Both methods have different strengths and weaknesses, depending on the application.

I transfer the concept of slicing from programs to graph grammars. Significant differences include how a slice is defined, what properties are preserved within the slice and what it means for an element to depend on another.

In the first section, mathematical tools required for the remainder of this chapter are introduced. This includes classic as well as more recent definitions and results of category theory.

The semantics of changes in graph grammars are examined in Section 3.2; pattern change identifiers are introduced as categorical means to formalize the effects of rewriting steps on patterns. Additionally, chemical graph grammars are introduced as well-behaved systems which are suitable as models of chemical processes. This is followed by the introduction of graph grammar equivalence, a term defined to capture the idea of systems behaving identically with regard to a pattern.

One of the main results of this thesis, formal definitions for graph grammar slicing as well as basic properties of such, are given in Section 3.3. We first introduce specialized productions as a central concept of graph grammar slicing; specialized productions reduce the number of graphs which a production can be applied on.

Finally, Section 3.4 gives a brief overview over application possibilities of graph grammar slicing in systems biology. These claims are supported by showing that graph grammar slicing can be automated, giving straightforward algorithms for the computation of slices.

3.1 Category Theoretical Tools for Grammar Slicing

The algebraic approach to graph rewriting, especially in the double pushout approach [19], relies heavily on constructs and results from category theory. Since we follow the algorithmic approach here, graph grammar slicing is also going to be developed in categorical terms. In this section, we review recent category theoretical results developed as generalizations and extensions of double-pushout rewriting. For a detailed yet general introduction to category theory, see for example Mac Lane [46].

3.1.1 Adhesive Categories

Adhesive categories, introduced in 2004 by Lack and Sobociński [44], include many categories arising in graph theory that are used in computer science and biological modeling. They provide a generalization of the double-pushout approach to graph rewriting. We will use some of their properties, as the category of typed graphs is an adhesive category.

We first define the notion of a *van Kampen square* which is a pushout for which a cube, having the pushout square as its bottom, enjoys additional properties.

Definition 3.1 (Van Kampen square) A *van Kampen square* is a pushout

$$\begin{tikzcd}
C \arrow[r, "f"] \arrow[d, "m"'] & B \arrow[d, "n"] \\
A \arrow[r, "g"'] & D
\end{tikzcd} \tag{3.1}$$

such that for a commutative cube that has diagram (3.1) as its bottom face and pullbacks as back faces,

$$\begin{tikzcd}
& & C' \arrow[dll, "m'"'] \arrow[dr, "f'"] \arrow[ddd, "c"] & \\
A' \arrow[dr, "g'"] \arrow[ddd, "a"'] & & & B' \arrow[dll, "n'"] \arrow[ddd, "b"] \\
& D' \arrow[ddd, "d"] & & \\
& & C \arrow[dll, "m"'] \arrow[dr, "f"] & \\
A \arrow[dr, "g"'] & & & B \arrow[dll, "n"] \\
& D & &
\end{tikzcd} \tag{3.2}$$

the front faces are pullbacks if and only if the top face is a pushout.

Adhesive categories are well behaved, in that the categorical limits most interesting in rewriting are guaranteed to exist and are well-behaved.

Definition 3.2 (Adhesive Category) A category **C** is *adhesive* if

1. **C** has pushouts along monomorphisms;
2. **C** has pullbacks;
3. pushouts along monomorphisms are van Kampen squares.

The value of adhesive categories for rewrite systems is given by the fact that properties desirable for rewriting in the double pushout approach hold in all adhesive categories.

Fact 3.3. *The following statements hold in adhesive categories:*

1. *Monomorphisms are stable under pushout.*

2. *Pushouts along monomorphisms are also pullbacks.*
3. *If the pushout complement of monomorphisms exists, then it is unique up to isomorphism.*

Many basic categories are adhesive, including the category of graphs and graph morphisms as well as its comma categories. This allows us to use the abstract framework of adhesive categories for our description of graph grammar slicing.

Fact 3.4. *The following categories are adhesive categories:*

- *The category* **Set** *of sets and functions;*
- *the category* **Graph** *of untyped graphs and graph morphisms;*
- *the category* $\mathbf{Graph_T}$ *of graphs typed over a graph T and typed graph morphisms.*

See the paper of Lack and Sobociński [44] for further discussions and for proofs of above facts.

3.1.2 Subobjects

The concept of objects being a part of a bigger object, like for example subsets or subgraphs, is transferable into a categorical context. The categorical definition of subobjects is based on isomorphic monomorphisms.

Given an object C of a category $\mathbf{C}$, a *subobject* of an object C in $\mathbf{C}$ is an isomorphism class of monomorphisms $a\colon A \to C$ into C, where two morphisms $a\colon A \to C$ and $b\colon B \to C$ are *isomorphic* if there exists an isomorphism $i\colon A \to B$ such that $a = b \circ i$.

A preorder is imposed on subobjects by a relation

$$(a\colon A \to C) \leq (b\colon B \to C)$$

defining the subobject a to be less than b precisely if there exists a morphism $f\colon A \to B$ such that $a = b \circ f$.

Subobjects are morphisms, but if there is no confusion, it is often convenient to identify a subobject $a\colon A \to C$ with the domain A of the morphism.

We write $A \preceq C$ to denote that there exists a monomorphism from A to C. Whenever the specific monomorphism is of interest it will be refered to or defined explicitly.

The notion of patterns used in this thesis is based on subobjects. Subobjects of objects in adhesive categories behave nicely in that their coproduct, the categorical equivalent to a binary union, always exists.

Fact 3.5. *Given an object C of an adhesive category $\mathbf{C}$, the category $\mathrm{sub}(C)$ of subobjects of C has binary coproducts; for two subobjects $a, b \in \mathrm{sub}(C)$ of C, the coproduct is the pushout of their intersection.*

Patterns and subobjects in the adhesive category of graphs and graph morphisms are discussed in Section 3.2.1.

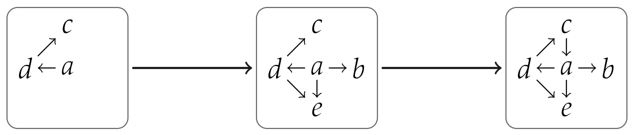
3.1.3 Final Pullback Complement

Like the well-known pushout complement, a pullback complement is a construct used in rewriting to complete a pair of composable arrows into a square [29, 34]. The advantage of a pullback complement over the more traditional pushout complement is that often the former exists even when the latter does not [20]. Different types of pullback complements exist, characterized by their relation to other pullback complements. While most authors often use the term pullback complement to mean the *maximal* or *final* pullback complement, we also make use of the *minimal* pullback complement which matches all other possible complements.

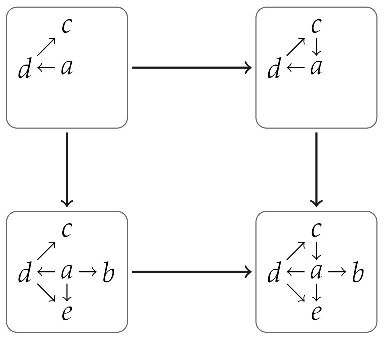
Definition 3.6 (Maximal pullback complement) The *maximal pullback complement* of composable arrows $A \xrightarrow{a} B \xrightarrow{b} C$ is a pair of arrows $A \xrightarrow{i} D \xrightarrow{d} C$ such that the square $A \xrightarrow{a} B \xrightarrow{b} C \xleftarrow{d} D \xleftarrow{i} A$ is a pullback and for each pullback $A' \xrightarrow{a'} B \xrightarrow{b'} C \xleftarrow{d'} D' \xleftarrow{i'} A'$ and for each $f\colon A' \to A$ such that $a' = a \circ f$ there exists a unique $g\colon D' \to D$ such that $d' = d \circ g$ and $i \circ f = i' \circ g$. Within diagrams, maximal pullback complements are marked by the symbol $\lrcorner$.

$$\begin{tikzcd}
A' \arrow[rrr, "i'"] \arrow[dr, bend left, "f"] \arrow[ddr, bend right, "a'"'] \arrow[dr, phantom, "\lrcorner", very near start] & & & D' \arrow[dl, bend right, "g"'] \arrow[ddl, bend left, "d'"] \\
& A \arrow[r, "i"] \arrow[d, "a"] \arrow[dr, phantom, "\lrcorner", very near start] & D \arrow[d, "d"] & \\
& B \arrow[r, "b"'] & C &
\end{tikzcd} \tag{3.3}$$

Example 3.1 (Maximal pullback complement) Given a pair of composable arrows like



a pullback complement is given by



The additional vertices in the rightmost graph are always added to the new graph, while edges are added only if source and target vertex of the edge are within the new graph. This demonstrates a central property of the maximal pullback complement of graphs, namely that the maximal pullback complement is a restriction of the second morphism on the first. ◆

Maximal pullback complements always exist in the category of graphs and graph morphisms; furthermore, if two composable arrows are monic, then so are the arrows of their maximal pullback complement [20].

We omit the definition for minimal pullback complements, since it is similar to the maximal pullback complement but with arrows f, g, a' and d' reversed.

Note that the minimal pullback complement of two monic morphisms $A \xrightarrow{a} B \xrightarrow{b} C$ is trivially given by $A \xrightarrow{id} A \xrightarrow{b \circ a} C$.

Lemma 3.7. *Let $A \xrightarrow{f_1} C \xrightarrow{g_1} X \xleftarrow{g_2} B \xleftarrow{f_2} A$ be a pullback and let $Y \xrightarrow{h_1} A \xrightarrow{f_2} B \xleftarrow{f_3} D \xleftarrow{h_2} Y$ be a square such that the diagram below commutes.*

$$\begin{tikzcd}
Y \arrow[r, "h_1"'] \arrow[d, "h_2"] & A \arrow[r, "f_1"'] \arrow[d, "f_2"] & C \arrow[d, "g_1"] \\
D \arrow[r, "f_3"'] & B \arrow[r, "g_2"'] & X
\end{tikzcd} \tag{3.4}$$

If all morphisms above are monic then the following statements hold:

1. *The left square is a pullback if and only if the outer square is a pullback.*
2. *If B is a maximal pullback complement in the right square, then D is a maximal pullback complement in the left square if, and only if, D is also a maximal pullback complement in the outer square.*
3. *If B is a minimal pullback complement in the right square, then D is a minimal pullback complement in the left square if, and only if, D is also a minimal pullback complement in the outer square. Under these conditions $B = A$ and $D = Y$.*

Proof. We prove only the forward implications, as the proof of the reverse is similar for each.

1. Let Y' be the pullback of the span $C \leftarrow X \rightarrow D$. Due to the universal property of A as a pullback object, there exists a unique morphism $u\colon Y' \rightarrow A$ such that all triangles commute. Likewise, Y is a pullback object, so the existence of commuting morphisms u and h_2' implies that there exists a unique morphisms $u'\colon Y \rightarrow Y'$ which makes dia-

gram (3.5) commute.

$$\begin{tikzcd}
Y' \arrow[drrr, bend left, "h'_1"] \arrow[drr, dashed, "u"] \arrow[ddr, bend right, "h'_2"'] &&& \\
& Y \arrow[ul, dashed, "u'"] \arrow[r, "h_1"'] \arrow[d, "h_2"] & A \arrow[r, "f_1"'] \arrow[d, "f_2"] & C \arrow[d, "g_1"] \\
& D \arrow[r, "f_3"'] & B \arrow[r, "g_2"'] & X
\end{tikzcd} \tag{3.5}$$

On the other hand, there must be a commuting morphism $u''\colon Y' \to Y$ since Y is a pullback. Hence, Y' is also a pullback of (B, f_2, f_3) and $Y = Y'$ due to the uniqueness of pullbacks up to isomorphism.

2. Let $Y \xrightarrow{y} Z \xrightarrow{z} X$ be the maximal pullback complement of the outer square. Since B is a maximal pullback complement, there must be an unique morphism $v\colon Z \to B$ such that $z = g_2 \circ v$. From the first part of the lemma it follows that all squares are pullbacks, hence there is also a unique commuting morphism $u'\colon D \to Z$. It follows that $Z = D$.

3. This trivially follows from the definition of minimal pullback complements.

□

3.2 Pattern Changes in Graph Grammars

The idea of graph grammar slicing was inspired by a problem frequently encountered in molecular biology: How does the occurrence of certain structures within a system change over time? Translated to the graph-based model, the equivalent question is how does the occurrence of a graph change in the process of graph transformations.

We first introduce the notion of patterns, which is based on subobjects as defined in Section 3.1.2; this is followed by a formal description of changes on patterns and chemical grammars, which provide a suitable basis for grammar slicing. The section is concluded with an analysis of grammar equivalence with respect to pattern changes.

3.2.1 Patterns and Pattern Changes

Formally, a *graph pattern* or just *pattern* is a graph. A pattern F *occurs* in a graph G if there exists a monic graph morphism $f : F \to G$ from the pattern F into the graph G. The morphism f is a subobject of G and is referred to as a *match of* F *in* G.

Example 3.2 (Graph pattern) A simple example of a graph pattern is given by

$$F_1 = \begin{tikzcd} & c \arrow[d] \\ d \arrow[ur] & a \arrow[l] \end{tikzcd}.$$

Occurrence of a graph pattern, here in the example of the above pattern F_1, G_0, is marked visually:

$$G_0 = \begin{tikzcd} & c & \\ d \arrow[ur] \arrow[dr] & a \arrow[u] \arrow[l] \arrow[r] \arrow[d] & b \\ & e & \end{tikzcd}$$

◆

Note that the frequently encountered graph theoretical notion of induced subgraphs is distinct from the notion of matched subgraphs as defined by the existence of monomorphisms. While the former requires that all edges between vertices in the image of a morphism are included in the induced subgraph, there is no such implicit additon of edges in the latter case: Only edges which are the image of an edge in the source graph are considered a part of the matched subgraph.

In graph grammar slicing, graph patterns take the role that variables play in program slicing: While a sliced program is indifferentiable from the original program with regard to changes of selected variables, graph grammar slicing intends to preserve the possible changes to a graph pattern between original and refined graph grammar. Therefore, we need a method to formalize the concept of pattern changes.

In the following, a method for the description of changes to patterns is developed. This framework is then used to give a formal meaning to the conceptual idea of graph grammars which are indifferentiable with regard to pattern changes.

The framework of double pushout rewriting is based on spans, so it is natural to express the changes to a pattern using a span. While pushouts are of fundamental importance for this approach, they are not a suitable basis for the definition of such spans, which is why we resort to pullbacks and pullback complements.

Definition 3.8 (Pattern change span) Let $s = B \xleftarrow{b} A \xrightarrow{c} C$ be a span, F a pattern and $f\colon F \to B$ an occurrence of F in B. The *maximal (minimal) pattern change span* of s with respect to f, written $s/_{+}f$ $(s/_{-}f)$, is a span $s_f = F \xleftarrow{p} J \xrightarrow{k} K$ along with a span morphism $(f\colon F \to B, \phi_J\colon J \to A, \phi_K\colon K \to C)$ such that $F \xleftarrow{p} J \xrightarrow{\phi_J} A$ is the pullback of $F \xrightarrow{f} B \xleftarrow{b} A$ and $J \xrightarrow{k} K \xrightarrow{\phi_K} C$ is the maximal (minimal) pullback complement of $J \xrightarrow{\phi_J} A \xrightarrow{c} C$:

$$\begin{tikzcd}
F \arrow[d, "f"'] & J \arrow[l, "p"'] \arrow[r, "k"] \arrow[d, "\phi_J"] \arrow[dl, phantom, "\llcorner", very near start] \arrow[dr, phantom, "\lrcorner", very near start] & K \arrow[d, "\phi_K"] \\
B & A \arrow[l, "b"] \arrow[r, "c"'] & C
\end{tikzcd} \tag{3.6}$$

The term *pattern change span* is used whenever either the maximal or minimal pattern change span is meant and the difference between the two is irrelevant. As usual in category theoretical approaches, we regard spans as unique only up to isomorphism. Two spans $s = B \leftarrow A \rightarrow C$ and $s' = B' \leftarrow A' \rightarrow C'$ are isomorphic, denoted $s \cong s'$, if there exists an invertible span morphism

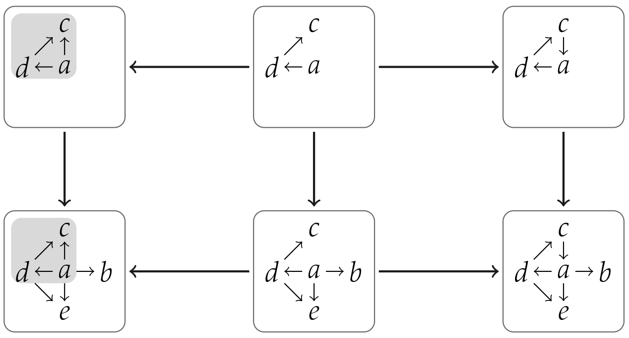
$$(\phi\colon s \to s') = (\phi_B\colon B \to B', \phi_A\colon A \to A', \phi_C\colon C \to C')$$

such that all squares commute.

Theorem 3.9 (Existence and uniqueness of pattern change spans). *Let all morphisms of a span s and an occurrence f be monomorphisms in the category* $\mathbf{Graph_T}$ *of typed graphs and typed graph morphisms. Then the pattern change span of s with regard to f exists and is unique up to isomorphism.*

Proof. The category $\mathbf{Graph_T}$ is an adhesive category [44], hence it has all pullbacks and the pullback in the left square in diagram (3.6) exists and is unique up to isomorphism. Likewise, the minimal as well as the maximal pullback complement of two composable morphisms exist in the category of graphs and both are unique up to isomorphism (see Corradini et al. [20]). □

Example 3.3 An example of a maximal pattern change span, here for a match of the pattern F_1, is given below.



The lower span is the application span of some derivation on the graph G_0 defined above, while the upper span is the corresponding pattern change graph of F_1. Note how the occurrence of F_1 is destroyed in both spans. ◆

In biological contexts, not only the changes to a pattern are of interest, but also what production triggered this change: For example, when a protein is dephosphorylated, it may make a difference if the reaction was catalyzed by a slow, unspecific phosphatase or by a fast, highly specific phosophatase. Given a phosphorylated protein as a pattern, these reactions would result in the same protein change spans within our graph based model. We make allowance for differentiation of these processes within our framework by inclusion of the production's name into the pattern change identifier.

Definition 3.10 (Pattern change identifier) The tuple (p, s') is a *maximal (minimal) pattern change indicator* of a production $(p\colon L \xleftarrow{l} I \xrightarrow{r} R)$ if s' is a maximal (minimal) pattern change span of the rule span $L \leftarrow I \rightarrow R$. Likewise, (p, s') is a pattern change indicator of a derivation $\rho = (G \overset{p,m_L}{\Longrightarrow} H)$, if s' is a pattern change span of the derivation's application span $\mathrm{app}(\rho)$.

A pattern change indicator is called *trivial* if its pattern change span is an identity span, that is if both morphisms are identity morphisms.

As for pattern change spans, the classifiers maximal and minimal are dropped in the term pattern change indicator whenever this detail can be omitted.

Two pattern change identifiers (p,s) and (p',s') are isomorphic if $p = p'$ and $s \cong s'$.

If there exists a pattern change identifier for a production, then this identifier is carried over to the derivations over the production. This means that all direct derivations using a production with an occurrence of a graph pattern in the rule span will have the same pattern change indicator as the production.

Lemma 3.11 (Transfer of a production's pattern change identifiers). *Let $(p\colon L \xleftarrow{l} I \xrightarrow{r} R)$ be a production, $f\colon F \to L$ a match and $p/f = F \xleftarrow{j_F} J \xrightarrow{j_K} K$ a corresponding pattern change span. For all p-derivations $\rho : G \overset{p,m_L}{\Longrightarrow} H$ there exists a match $f'\colon F \to G$, such that $((p,J), F \xleftarrow{j_F} J \xrightarrow{j_K} K)$ is the corresponding pattern change indicator.*

Proof. Let $j\colon J \to I$ and $f^*\colon K \to R$ be functions such that the square $J \xrightarrow{j_F} F \xrightarrow{f} L \xleftarrow{l} I \xleftarrow{j} J$ is a pullback and $J \xrightarrow{j_K} K \xrightarrow{f^*} R \xleftarrow{r} I \xleftarrow{j} J$ is a pushout as depicted in diagram (3.7).

$$\begin{tikzcd}
F \arrow[d, "f"'] & J \arrow[l, "j_F"'] \arrow[r, "j_K"] \arrow[d, "j"'] & K \arrow[d, "f^*"'] \\
L \arrow[d, "m_L"'] & I \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "m_I"'] & R \arrow[d, "m_R"'] \\
G & C \arrow[l, "g"'] \arrow[r, "h"] & H
\end{tikzcd} \tag{3.7}$$

The required occurrence of F in G is given by the composition $f' = m_L \circ f$ of the occurrence f of F in L and the match m_L of L in G. Since the lower left square $I \xrightarrow{l} L \xrightarrow{m_L} G \xleftarrow{g} C \xleftarrow{m_I} I$ in diagram (3.7) is a pushout, it follows by Lemma

3.7 that the left outer square is a pullback and the right outer square is a pullback with K being a maximal pullback complement. Hence, $F \xleftarrow{f} J \xrightarrow{k} K$ is the derivation's pattern change span. □

3.2.2 Chemical Graph Grammars as Well-Behaved Systems

The definitions introduced in this chapter are valid for all graph grammars. However, the focus of this work is on graph grammars modeling reactive systems, so some of the results presented in the following are valid only for grammars with additional properties.

We impose simple yet important limitations on the graph grammars under consideration, which is justified by the fact that these limitations are common features of reactive systems. In the following, we assume that the applicability of the rules in the graph grammars and the result of the application are purely match determined, meaning that every existing match of the left hand side of a rule will always satisfy the *dangling* and *identification* conditions.

The *identification condition* requires that two objects of which one is preserved and the other is deleted, must not be mapped to the same image. This can happen only for non-injective morphisms; however, injective matches are the intuitive and traditional way in which chemical reaction rules are interpreted, ensuring such basic properties as mass preservation within a reaction. Restricting to the category theoretical generalization of injective morphisms, namely monomorphisms, is therefore a reasonable limitation in the context of reaction networks. Consequently, the identification condition will always be met.

The *dangling condition* requires that there may not be any dangling edges after the rewrite step. An edge is *dangling* if it has either a source vertex or a target vertex, but not both. Such edges are produced when the source or target vertex of an edge is deleted in a production while the other vertex is preserved. In any typical chemical reaction system, elements may be deleted if they interact with their environment in a specific, unambiguous way. These interactions are reflected in deletion process modeling productions that require enough context to ensure that the surrounding context of all deleted elements is completely specified. This renders the occurrence of dangling edges as a result of vertex deletions impossible. The dangling condition is therefore satisfied in rewrite steps regardless of the applied production and match.

Restrictions on the grammars under consideration are captured in the following definition. Recall that an object A for which there exists a monomorphism into another object C, the object A is called a subobject of C (see Section 3.1.2)

Definition 3.12 (Chemical grammar) Let $\mathbf{Graph_T}$ be the category of graphs typed over T, let $\mathcal{M}$ be a class of objects in $\mathbf{Graph_T}$ called *molecules* and let $\mathrm{sub}(\mathcal{M})$ be the class of subobjects of the elements of $\mathcal{M}$. Furthermore, let $\Pi = \{p\colon L_p \xleftarrow{l_p} I_p \xrightarrow{r_p} R_p\}_{p \in P}$ be a set of linear productions such that the components $L_p, I_p, R_p \in \mathrm{sub}(\mathcal{M})$ are subobjects of $\mathcal{M}$ for all $p \in P$. A grammar $\mathcal{G} = (T, G_0, P, \Pi)$ is *chemical* if the following conditions hold:

1. $G_0 \in \mathcal{M}$,
2. all productions are completely application safe and
3. for all derivations $G \xRightarrow{n_p, m_L} H$ with $p \in P$, if G is a molecule, then so is H.

Here a production $\pi = (p\colon L \xleftarrow{l} I \xrightarrow{r} R)$ is called *application safe* if for all matches $m\colon L \to G$ into some subobject in $\mathrm{sub}(\mathcal{L}(\mathcal{G}))$, the dangling condition is satisfied. It is called *completely application safe* in $\mathcal{G}$ if it is invertible and both π and π^{-1} are application safe.

Note that, while for each derivation in a chemical grammar there must exist an inverse derivation, the inverse production is not necessarily within the set of productions of the grammar and hence the inverse derivation may very well not exist in the set of the grammar's possible derivations.

Example 3.4 (Chemical graph grammar) As a running example, in this chapter we will use graphs typed over the graph

$$T = \begin{tikzcd} & c \arrow[dl, leftrightarrow] \arrow[d, leftrightarrow] \arrow[dr, leftrightarrow] & \\ d \arrow[r, leftrightarrow] \arrow[dr, leftrightarrow] & a \arrow[r, leftrightarrow] \arrow[d, leftrightarrow] & b \\ & e & \end{tikzcd}$$

which was presented first in Example 2.1. The class $\mathcal{M}$ of objects is the set of graphs satisfying all of the following conditions:

- Graphs must be typed over T.
- The graph must not contain any parallel or anti-parallel edges.
- The sum of the in-degree and the out-degree must be at most 4 for vertices of type a and at most 3 for types b, c, d and e.

The productions in this example only change the directions of edges. This is similar to the edge relabeling approach for chemical reactions by Rosselló and Valiente [57]. The graph morphisms in the productions below are not given explicitly but are implied by the vertex labels.

Production p_1 inverts an edge $a \to c$ if there is also an edge from a to e:

$$\begin{tikzcd}
p_1 : & \boxed{\begin{matrix} c \\ \uparrow \\ a \\ \downarrow \\ e \end{matrix}} & \boxed{\begin{matrix} c \\ \\ a \\ \downarrow \\ e \end{matrix}} \arrow[l, "l_1"'] \arrow[r, "r_1"] & \boxed{\begin{matrix} c \\ \downarrow \\ a \\ \downarrow \\ e \end{matrix}}
\end{tikzcd}$$

The second production p_2 checks for the existence of two edges into e from both a and d and for an edges from a to d and then inverts all edges adjacent to e.

$$\begin{tikzcd}
p_2 : & \boxed{\begin{matrix} d & \leftarrow & a \\ & \searrow & \downarrow \\ & & e \end{matrix}} & \boxed{\begin{matrix} d & \leftarrow & a \\ & & \\ & & e \end{matrix}} \arrow[l, "l_2"'] \arrow[r, "r_2"] & \boxed{\begin{matrix} d & \leftarrow & a \\ & \nwarrow & \uparrow \\ & & e \end{matrix}}
\end{tikzcd}$$

If there is a feed forward loop from e to d going through a, then application of production p_3 converts the loop into a clockwise cycle.

$$\begin{tikzcd}
p_3 : & \boxed{\begin{matrix} d & \leftarrow & a \\ & \nwarrow & \uparrow \\ & & e \end{matrix}} & \boxed{\begin{matrix} d & & a \\ & \nwarrow & \\ & & e \end{matrix}} \arrow[l, "l_3"'] \arrow[r, "r_3"] & \boxed{\begin{matrix} d & \rightarrow & a \\ & \nwarrow & \downarrow \\ & & e \end{matrix}}
\end{tikzcd}$$

Finally, production p_4 inverts an edge $a \to b$ if it exists.

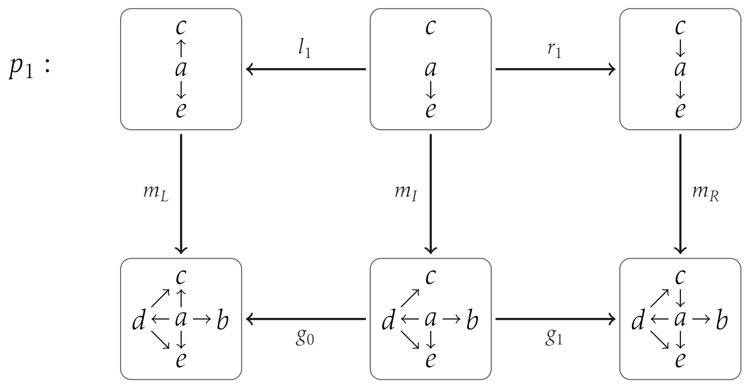
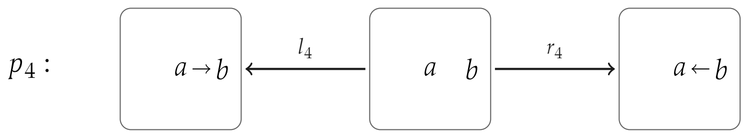
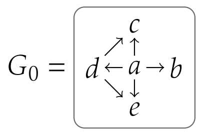


Figure 3.1: An example of a direct derivation based on Example 3.4.



Taking the production names $P = \{p_i\}_{i\in\{1,2,3,4\}}$ and $\Pi = \{p\colon Lp \xleftarrow{lp} Ip \xrightarrow{rp} Rp\}_{p\in P}$, the grammar $\mathcal{G}_E = (T,G_0,P,\Pi)$ is a chemical grammar for every T-typed start graph G_0. Here we set



An explicit example of a direct derivation of on this example grammar is given in Figure 3.1. A depiction of all derivations which are possible in $\mathcal{G}_E$ is given in Figure 3.2. It is easily verified from the latter figure that the grammar is indeed similar to a chemical reaction system in that no vertices are deleted and the number of edges is kept constant. ◆

3.2.3 Global Pattern Changes in Chemical Graph Grammars

Pattern change spans as defined in Definition 3.8 contain information about changes to a pattern on the local level of rules and direct derivations. The

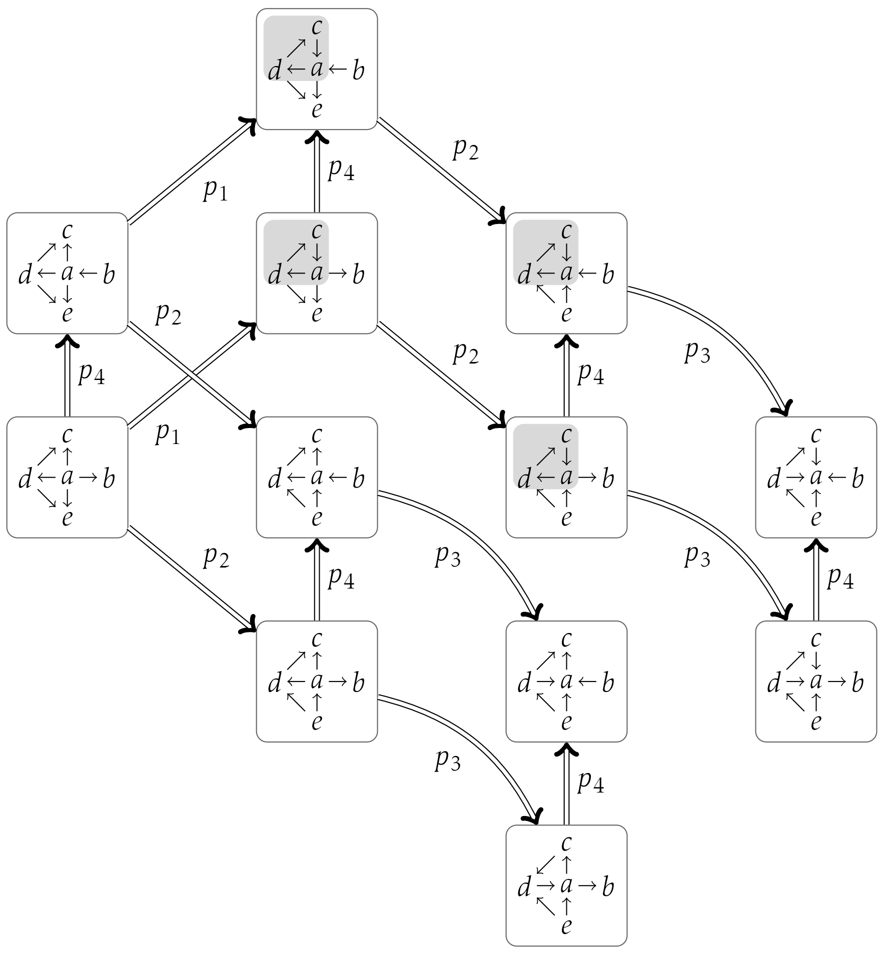


Figure 3.2: All possible derivations of the example graph grammar presented in Example 3.4. The graph pattern F_1 is highlighted when present in a graph. Parallel derivation edges are generated by the same production.

global view of possible changes in a grammar is captured by characteristic sets of pattern changes.

Definition 3.13 (Characteristic set of pattern changes) Given a pattern F, a production $(p\colon L \xleftarrow{l} I \xrightarrow{r} R)$ and a grammar $\mathcal{G} = (T, G_0, P, \Pi)$, the *characteristic set of p-mediated F-changes in $\mathcal{G}$*, denoted $\chi^F_{\mathcal{G}}(p)$, is the union of all non-trivial pattern change indicators of F-occurrences in p-derivations that are possible in $\mathcal{G}$:

$$\chi^F_{\mathcal{G}}(p) = \{\mathrm{app}(\varrho)/f \quad | \quad \varrho \in R_p, \quad f\colon F \to G, \quad \mathrm{app}(\varrho)/_{-}f \text{ is non-trivial}\}$$

where

$$R_p = \{G \overset{p,m_L}{\Longrightarrow}_{\mathcal{G}} H \quad | \quad G_0 \Rightarrow^*_{\mathcal{G}} G\}$$

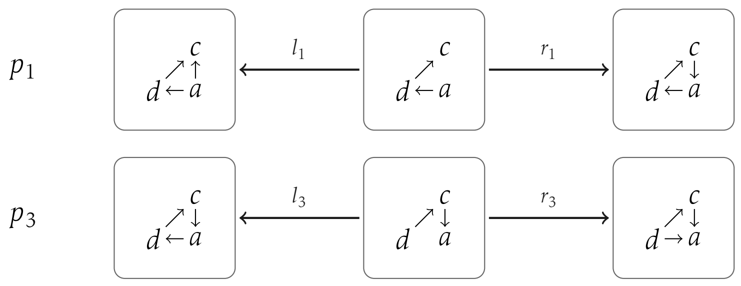
is the set of possible p-derivations.

We write

$$\chi^F_{\mathcal{G}}(\Pi) = \bigcup_{p \in \Pi} \chi^F_{\mathcal{G}}(p)$$

to denote the set of all non-trivial pattern change indicators of a pattern F in $\mathcal{G}$.

Example 3.5 (Characteristic pattern changes) The set of characteristic pattern changes of pattern F_1 for the chemical graph grammar specified in Example 3.4 is given by the following set of pattern change identifiers:



There also exist pattern change identifiers generated from derivations using productions p_2 and p_4, but those pattern change identifiers are trivial and therefore not in the characteristic pattern change set. Hence, the productions p_1 and p_3 are the only productions in the example grammar that directly affect the pattern F_1. ◆

We are now able to define equivalence of grammars with regard to a pattern. Basically, we say a grammar $\mathcal{G}$ is equivalent to another grammar $\mathcal{G}'$ with respect to a graph pattern if they can affect the pattern in the same way.

Two characteristic sets of F-changes are isomorphic, written $\chi^F_{\mathcal{G}}(\Pi) \cong \chi^F_{\mathcal{G}'}(\Pi')$, if there exists a bijection $f : \chi^F_{\mathcal{G}}(\Pi) \to \chi^F_{\mathcal{G}'}(\Pi')$ between the sets such that $c \cong f(c)$ for all $c \in \chi^F_{\mathcal{G}}(\Pi)$.

Definition 3.14 (Equivalence of Graph Grammars) Given a graph pattern F and graph grammars $\mathcal{G} = (T, G_0, P, \Pi)$ and $\mathcal{G}'_E = (T', G'_0, P', \Pi')$, the graph grammars are *F-change equivalent*, denoted $\mathcal{G} \equiv_F \mathcal{G}'$, if the characteristic sets of F-changes are isomorphic:

$$\chi^F_{\mathcal{G}}(\Pi) \cong \chi^F_{\mathcal{G}'}(\Pi').$$

Example 3.6 (Equivalent graph grammars) Given the pattern

$$F_1 = \begin{tikzcd} & c \arrow[d] \\ d \arrow[ur] & a \arrow[l] \end{tikzcd}$$

from Example 3.2, we examine the characteristic change set of the grammar $\mathcal{G}'_E = (T, G_0, P', \Pi')$, where the set Π' is given by the productions

$$\begin{tikzcd}
 & & c & & & c & & & c \arrow[d] \\
p_1' : & d \arrow[ur] & a \arrow[l] \arrow[u] \arrow[d] & & d \arrow[ur] \arrow[ll, "l_1'"'] & a \arrow[l] \arrow[d] \arrow[rr, "r_1'"] & & d \arrow[ur] & a \arrow[l] \arrow[d] \\
 & & e & & & e & & & e
\end{tikzcd}$$

$$\begin{tikzcd}
 & & c & & & c & & & c \\
p_2' : & d \arrow[dr] & a \arrow[l] \arrow[u] \arrow[d] & & d \arrow[ll, "l_2'"'] & a \arrow[l] \arrow[u] \arrow[rr, "r_2'"] & & d & a \arrow[l] \arrow[u] \\
 & & e & & & e & & & e \arrow[u] \arrow[ul]
\end{tikzcd}$$

$$\begin{tikzcd}
 & & c & & & c & & & c \\
p_3' : & d & a \arrow[l] \arrow[u] & & d \arrow[ll, "l_3'"'] & a \arrow[u] \arrow[rr, "r_3'"] & & d \arrow[r] & a \arrow[u] \arrow[d] \\
 & & e \arrow[u] \arrow[ul] & & & e \arrow[ul] & & & e \arrow[ul]
\end{tikzcd}$$

The derivations possible with $\mathcal{G}'_E$ are depicted in Figure 3.3. It is easily verified that the characteristic F_1-pattern changes $\chi^F_{\mathcal{G}_E}(\Pi')$ are given by

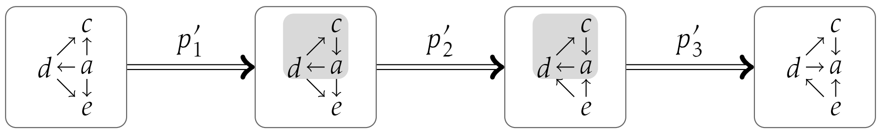
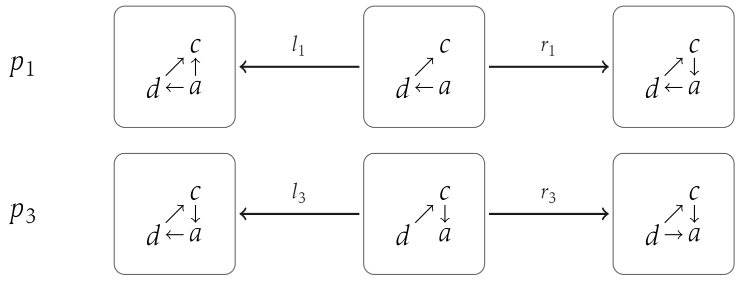


Figure 3.3: All possible derivations of the graph grammar $\mathcal{G}'_E$ defined in Example 3.6 .



and therefore equal to the pattern change set $\chi^F_{\mathcal{G}_E}(\Pi)$ as is shown in Example 3.5. It follows that the grammars $\mathcal{G}'_E$ as defined above and $\mathcal{G}_E$, defined in Example 3.4, are F_1-change equivalent. ◆

3.3 Slicing of Graph Grammars with Respect to Patterns

A model of computation based on graph grammars is inherently nondeterministic, and the data present at any time directly influences which computations are possible. The aim of this section is to define slices of graph grammars as refined grammars preserving all computations relevant with regard to a pattern. We start by introducing specialized productions, which restrict the applicability of productions without affecting the graph transformation capabilities with regard to a pattern. This leads to the discussion of pattern-affecting graphs, the graph grammar equivalent to dependencies in program slicing. It is then continued with the introduction of graph grammar slicing, an approach to soundly restrict complex grammars to the components with an effect on a graph pattern.

3.3.1 Specializations on Graph Patterns

The ways in which a pattern may be changed within derivations depends on the occurrences of the pattern within the graphs of the grammar's language and on the productions that can be applied on these graphs. However, a pattern may be changed in a derivation step using a certain production while another derivation using the same production may leave the pattern unchanged. This depends on the subgraph on which the rule is applied: The subgraph that is changed by the application of the production must obviously overlap with the occurrence of a pattern in the processed graph. This holds for creation of a pattern, where the overlap must be in the result graph, as well as for destruction, where the overlap must be in the source graph.

We formalize the possible effects of a production on a pattern using the concept of specialized productions. These productions, which are more specialized in the sense that they require a bigger subgraph to be matched, are used to define dependencies of patterns on graphs. We start by defining what it means for a pattern to overlap with a graph.

A *partial match* of F in X is a span of monomorphisms $d = F \overset{d_F}{\leftarrow} D \overset{d_X}{\rightarrow} X$. The match is non-trivial if D is not the empty graph.

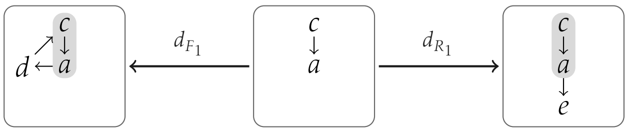
Example 3.7 (Partial match) Consider the pattern

$$F_1 = \boxed{\begin{tikzcd} & c \arrow[d] \\ d \arrow[ur] & a \arrow[l] \end{tikzcd}}$$

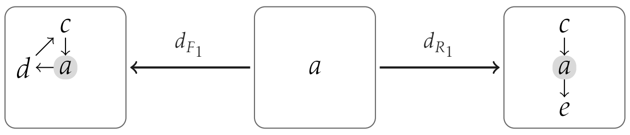
and the right hand side of the production p_1 from Example 3.4,

$$R_1 = \boxed{\begin{tikzcd} c \arrow[d] \\ a \arrow[d] \\ e \end{tikzcd}}$$

Then the spans



and



are both partial matches of F_1 in R_1. ◆

Partial matches mark subgraphs that are similar in two graphs. Furthermore, the matched regions may be merged such that the two graphs are fused to create a new graph. For patterns and productions, this is used here to create new, specialized productions, such that the partial match is extended to become a complete match.

Roughly speaking, a specialized rule is identical to the original rule with regard to the part of the graph that is being rewritten, but requires additional context. The bigger context graph effectively reduces the set of graphs the rule can be applied on.

Definition 3.15 (Specialized production) Let F be a pattern, let $(p\colon L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R)$ be a production, let $F \overset{d_F}{\leftarrow} D \overset{d_L}{\rightarrow} L$ be a partial match of F in L and let (L', m_F, m_L) be its pushout. A *left specialized production* over (d_F, d_L) is a production $((p, I')\colon L' \overset{l'}{\leftarrow} I' \overset{r'}{\rightarrow} R')$ such that there exists a span morphism

$$\phi = (\phi_L\colon L \to L', \phi_I\colon I \to I', \phi_R\colon R \to R')$$

for which all squares in diagram (3.8) are pushouts.

$$\begin{tikzcd}
 & D \arrow[dl, "d_F"'] \arrow[dr, "d_L"] & & & \\
F \arrow[dr, "\phi_F"'] & & L \arrow[dl, "\phi_L"] & I \arrow[l, "l"'] \arrow[r, "r"] \arrow[dl, "\phi_I"] & R \arrow[dl, "\phi_R"] \\
 & L' & I' \arrow[l, "l'"] \arrow[r, "r'"'] & R' &
\end{tikzcd} \tag{3.8}$$

Analogously, given a partial match $F \overset{d_F}{\leftarrow} D \overset{d_R}{\rightarrow} L$ and its pushout (R', m_F, m_R), a *right specialized production* over (d_F, d_R) is a production $((p, I')\colon L' \overset{l'}{\leftarrow} I' \overset{r'}{\rightarrow} R'$

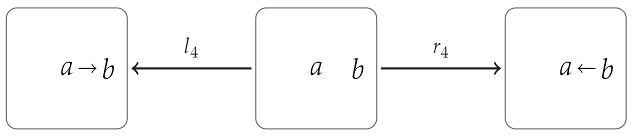
such that there exists a span morphism ($\phi_L \colon L \to L', \phi_I \colon I \to I', \phi_R \colon R \to R'$) for which all squares in diagram (3.9) are pushouts.

$$\begin{tikzcd}
 & & & & & D \arrow[dl, "d_R"'] \arrow[dr, "d_F"] & \\
L \arrow[dr, "\phi_L"'] & & I \arrow[ll, "l"'] \arrow[rr, "r"] \arrow[dr, "\phi_I"'] & & R \arrow[dr, "\phi_R"'] & & F \arrow[dl, "m_F"] \\
 & L' & & I' \arrow[ll, "l'"] \arrow[rr, "r'"'] & & R' &
\end{tikzcd} \tag{3.9}$$

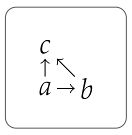
A productions is a *specialized production* if it is a left or right specialized production. The above span morphisms are also called the *left* and *right specialization morphism*, respectively.

The original production name is extended by the specialized context, so that specialized productions still may be indexed over their production names. It is worth stressing that specialized productions are closely related to the notion of rewriting with borrowed context as defined by Ehrig and König [32].

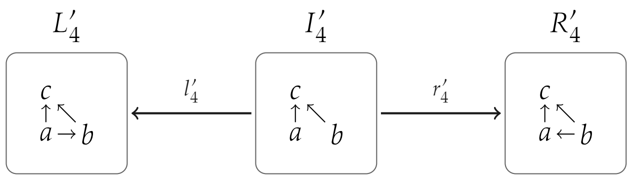
Example 3.8 We generate a specialized production of production p_4 which was defined in Example 3.4:

p_4 :


A specialization of the production p_4 on the pattern

$F_2 =$


is the production (p_4', I_4' as specified below.

(p_4', I_4') :


The effect of specialization becomes clear when given a graph

$$G = \begin{tikzcd} & c & \\ b & a \arrow[l] \arrow[r] \arrow[u] & b \arrow[ul] \end{tikzcd}$$

for which there are two possible derivations using production p_4. One derivation uses the match in the left of the graph, the second one the match on the right. However, the left hand side of specialized rule p_4' has only one match in above graph; it is specialized on the destruction of the pattern F_2.
◆

A production $(p\colon L \xleftarrow{l} I \xrightarrow{r} R)$ is *applicable* in a graph grammar $\mathcal{G}$ if there exists a graph $G \in \mathcal{L}(\mathcal{G})$ and a match $m_L\colon L \to G$ such that both the dangling and identification condition are satisfied by the match. This is equivalent to stating that there is at least one derivation using p that is possible in $\mathcal{G}$. If a production is not applicable in a grammar, then the grammar will produce the same language of reachable graphs and the same set of possible derivations regardless of whether the production is within the grammar's production set or not. This can be regarded as a measure for soundness and relevance of a production.

Theorem 3.16 (Existence and uniqueness of specialized rules). *Let F and D be patterns, let $\pi = (p\colon L \xleftarrow{l} I \xrightarrow{r} R)$ be an application safe production that is applicable in a chemical grammar $\mathcal{G}$ and let $F \xleftarrow{d_P} D \xrightarrow{d_L} L$ be a partial match. If there exists a graph $G \in \mathcal{L}(\mathcal{G})$ in the grammar's language for which the pushout object L' of partial match D is a subobject, then the left-specialized production π exists and is unique up to isomorphism.*

The equivalent statement for a span $F \xleftarrow{d_P'} D' \xrightarrow{d_R'} R$ and the right-specialized production of π holds as well.

Proof. Suppose $G \in \mathcal{G}$ is a graph such that $L' \leq G$. The definition of chemical grammars implies that L' is a subobject $L' \in \text{sub}(\mathcal{L}(\mathcal{G}))$. Since π is an application safe production by definition, satisfaction of the dangling condition is guaranteed for any match $s\colon L \to L'$. □

Application safety is a central feature of chemical graph grammars. It is desireable that this property is preserved in a refinement of such grammar. This is the case if a new grammar is constructed using specialized productions, as is shown by the following lemma:

Lemma 3.17 (Application safety of specialized productions). *A specialized production of an application safe production is itself application safe.*

Proof. We assume $(p'\colon L' \xleftarrow{l'} I' \xrightarrow{r'} R')$ to be a specialized rule of $(p\colon L \xleftarrow{l} I \xrightarrow{r} R)$ for which $\phi = (\phi_L\colon L \to L', \phi_I\colon I \to I', \phi_R\colon R \to R')$ is the specialization morphism. There is nothing left to show if there is no graph which is matched by the left hand side L'. Therefore, let $m'_L\colon L' \to G$ be a match of the left hand side into any graph $G \in \mathcal{L}(\mathcal{G})$. We show that the pushout complement of $I' \xrightarrow{l'} L' \xrightarrow{m'_L} G$ exists:

The production $(p\colon L \xleftarrow{l} I \xrightarrow{r} R)$ is application safe and there is a morphism $m_L = m'_L \circ \phi_L$ from its left hand side into the graph G, hence the pushout complement $I \xrightarrow{i} C \xrightarrow{g} G$ exists. Pushouts along monomorphisms exist, so there is also an object C' and morphisms $I' \xrightarrow{i'} C' \xleftarrow{c'} C$ such that the cube in diagram (3.10) commutes.

$$\begin{tikzcd}
& & I \arrow[dll, "\phi_I"'] \arrow[dr, "l"] \arrow[ddd, "i"] & \\
I' \arrow[dr, "l'"] \arrow[ddd, "i'"'] & & & L \arrow[dll, "\phi_L"] \arrow[ddd, "m_L"] \\
& L' \arrow[ddd, "m'_L"] & & \\
& & C \arrow[dll, "c'"'] \arrow[dr, "g"] & \\
C' \arrow[dr, "g'"'] & & & G \arrow[dll, "id"] \\
& G & &
\end{tikzcd} \tag{3.10}$$

The back and top faces are pushouts by construction. From that, it is easily verified that all faces of above cube are pushouts. Therefore it follows that $C' \cong C$ and we get (i', g') to be the required pushout complement. □

Looking at the proof from a different perspective, we get the following proposition:

Proposition 3.18 (Isomorphic application spans of refined productions). *Given a production* $(p\colon L \xleftarrow{l} I \xrightarrow{r} R)$*, a specialized production* $(p'\colon L' \xleftarrow{l'} I' \xrightarrow{r'} R')$ *with specialization morphism* $(\phi_L\colon L \to L', \phi_I\colon I \to I', \phi_R\colon R \to R')$ *and a match* $m'\colon L' \to G$*, it holds that*

$$\mathrm{app}(G \xRightarrow{p',m'} H) \cong \mathrm{app}(G \xRightarrow{p,(m' \circ s)} H)$$

Proof. This readily follows from above proof, where it is shown that the derivations context graphs $C \cong C'$ are isomorphic, implying isomorphism of the complete application span due to the composability of pushouts. □

Therefore, specialized productions also have the nice property to produce the same graph transformation as the original rule whenever both are applicable on the same part of a graph.

3.3.2 Pattern-affecting Graphs

In the same manner in which a variable assignment depends on preceding commands, occurrence of graph patterns is influenced by the occurrence of other graphs. We define pattern-affecting graphs to account for the fact that the creation or destruction of a pattern is subject to the existence of specific graphs in a graph grammar.

A pattern is *changed* within a derivation if the minimal pattern change span is not the identity span of the pattern. We focus on graphs for which matching another graph ensures the possiblity of pattern modification. For a match to be changed, its not sufficient that it overlaps with the left or right hand side of a production, but furthermore there may not be a corresponding partial match in the interface graph.

Theorem 3.19 (Pattern changing specializations). *Let* $F \xleftarrow{d_F} D \xrightarrow{d_L} L$ *be a partial match of a pattern* F *into the left hand side of a production* $(p\colon L \xleftarrow{l} I \xrightarrow{r} R)$*. The minimal pattern change span of the corresponding specialized production* $(p'\colon L' \xleftarrow{l'} I' \xrightarrow{r'} R')$ *is the identity span of* F *if and only if there exists a morphism* $d_I\colon D \to I$ *such that* $d_L = l \circ d_I$*.*

Proof. Let $F \xleftarrow{j_F} J \xrightarrow{j_K} K$ be the minimal pattern change span of the specialized production. The graph J is a pullback so there is a unique commuting arrow $d_J \colon D \to J$. The bottom face of the cube in diagram (3.11) is a van Kampen square and the side faces are pullbacks, so the top face is a pushout.

$$\begin{tikzcd}
 & & D \arrow[dll, "d_J"'] \arrow[dr, "d_I"] \arrow[ddd, "id"] & \\
J \arrow[dr, "f_J"] \arrow[ddd, "j_F"'] & & & I \arrow[dll, "\phi_I"] \arrow[ddd, "l"] \\
 & I' \arrow[ddd, "l'"] & & \\
 & & D \arrow[dll, "d_F"'] \arrow[dr, "d_L"] & \\
F \arrow[dr, "f"'] & & & L \arrow[dll, "\phi_L"] \\
 & L' & &
\end{tikzcd} \qquad (3.11)$$

Pushouts are composable and pullback complements are unique up to isomorphism, so it follows $J \cong F$.

The forward implication follows from I being the pullback object of $I' \xrightarrow{l'} L' \xleftarrow{\phi_L} L$ and $I' \xleftarrow{f_J \circ d_J} D \xrightarrow{d_L} L$ producing a commuting square with aforementioned span. Hence there is a unique morphism from D to I. □

Since we are interested in specialized productions that are guaranteed to change the occurrence of a pattern, we can restrict the set of specialized rules to those that are constructed from partial matches with no commuting morphism to the interface graph.

Given a monomorphism $x \colon I \to X$, the span $(D, d_F \colon D \to F, d_X \colon D \to X)$ is a *pattern changing partial match* of a pattern F in X if there is no lifting of d to I making the triangle commute.

$$\begin{tikzcd}
 & D \arrow[dl, "d_F"'] \arrow[d, "d_X"'] \arrow[dr, dashed, "/" marking] & \\
F & X & I \arrow[l, "x"]
\end{tikzcd} \qquad (3.12)$$

For convenience, we define a function that gives all pattern-changing specializations of a production.

Definition 3.20 (Pattern-changing specializations) Given a pattern F, a graph grammar $\mathcal{G} = (T, G_0, P, \Pi)$ and a production $\pi = (p\colon L \xleftarrow{l} I \xrightarrow{r} R)$ with $p \in P, \pi \in \Pi$, the *F-changing specializations of* π, denoted $\zeta_{\mathcal{G}}^{F}(\pi)$, is the set of all specializations of π that are applicable in $\mathcal{G}$ and F-changing.

We write

$$\zeta_{\mathcal{G}}^{F}(\Pi) := \bigcup_{\pi \in \Pi} \zeta_{\mathcal{G}}^{F}(\pi)$$

to refer to the F-changing specializations of all productions of the grammar $\mathcal{G}$.

A derivation using a specialized rule $\pi' \in \zeta_{\mathcal{G}}^{F}(\Pi)$ will certainly affect the occurrence of the pattern F. We only consider chemical graph grammars here, so the specialized rules in $\zeta_{\mathcal{G}}^{F}(\Pi)$ are application safe (see Lemma 3.17). Hence, if the left hand side of a specialized rule in $\zeta_{\mathcal{G}}^{F}(\Pi)$ matches a graph, then there exists a derivation affecting the pattern. This motivates the following definition:

Definition 3.21 (Pattern-affecting patterns) Given a pattern F and a graph grammar $\mathcal{G} = (T, G_0, P, \Pi)$, the set of *$F$-affecting pattern* is the set of left hand sides of F-changing specializations, that is

$$\mathcal{A}_{\mathcal{G}}^{F} = \{L' : (p'\colon L' \xleftarrow{l'} I' \xrightarrow{r'} R') \in \zeta_{\mathcal{G}}^{F}(\Pi)\}.$$

Example 3.9 (Pattern-affecting graphs) Considering the pattern F_1

$$F_1 = \boxed{\begin{tikzcd} & c \arrow[d] \\ d \arrow[ur] & a \arrow[l] \end{tikzcd}}$$

and the production

$$p_1 : \quad \begin{tikzcd} c & & c & & c \arrow[d] \\ a \arrow[u] \arrow[d] & & a \arrow[d] \arrow[ll, "l_1"'] \arrow[rr, "r_1"] & & a \arrow[d] \\ e & & e & & e \end{tikzcd}$$

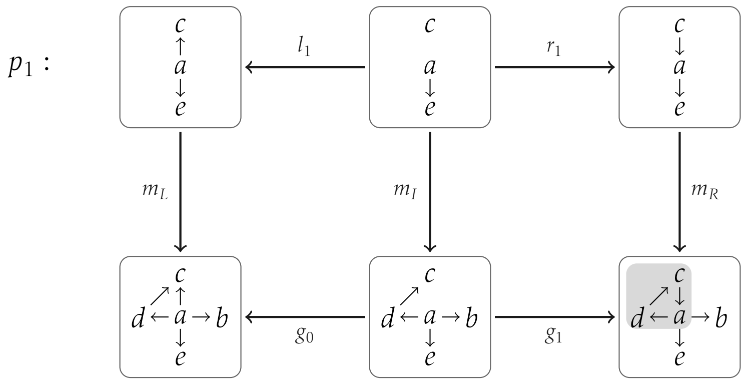
The pattern

$$\begin{tikzcd}
 & c \\
d \arrow[ur] & a \arrow[u] \arrow[l] \arrow[d] \\
 & e
\end{tikzcd}$$

is the F_1-affecting pattern for the production p_1: For every graph matched by this pattern there exists a derivation such that F_1 is changed. For example, given the matched graph

$$\begin{tikzcd}
 & c & \\
d \arrow[ur] & a \arrow[u] \arrow[l] \arrow[r] \arrow[d] & b \\
 & e &
\end{tikzcd}$$

there is a derivation



with a non-trivial F_1 change span (matching of F_1 is highlighted in the derivation).

◆

3.3.3 Definition and Properties of Graph Grammar Slices

Using the results from the previous sections of this chapter, we are finally able to define and characterize graph grammar slicing. The aim of graph grammar slicing is not to reduce the number of productions but to reduce the number of possible derivations. However, the sliced version preserves all changes with regard to a pattern while still resulting in the same or smaller set of possible derivations as the original grammar. This is archived using beforementioned specialized productions which produce similar graph transformations whenever they are applicable. However, the context graphs of specialized productions is equal or greater to the original productions. As a result, specialized productions can be applied on less graphs than their unrefined counterparts.

The definition of graph grammar slicing below is followed by some of the basic yet important properties of grammar slices. Roughly speaking, the slices are refinements where all graph grammar details which do not affect the occurrence of a pattern are omitted, while no new rewrite possibilities are introduced. We give a recursive definition of a graph slice.

Definition 3.22 (Slice of a graph grammar) Let F be a pattern and $\mathcal{G} = (T, G_0, P, \Pi)$ a grammar. A grammar $\mathcal{G}' = (T', G_0', P', \Pi')$ is a *slice* of $\mathcal{G}$ over F if the following conditions hold:

- $G_0' \preceq G_0$,
- $F \preceq G_0 \iff F \preceq G_0'$,
- foreach production $\pi' \in \Pi'$ there exists a production $\pi \in \Pi$ such that π' is a specialization of π and
- $\mathcal{G}'$ is also a slice for all elements in the set $\mathcal{A}_{\mathcal{G}}^F$ of F-affecting patterns.

As for the basic properties, we first show that the possible derivations of the slice are smaller in the sense that they can be *embedded* into the possible derivations of the original grammar.

The application sequence of a derivation ρ' is *embeddable* into a derivation sequence ρ if there exist commuting morphisms between corresponding graphs of the two derivations such that the sequential order is preserved.

Definition 3.23 (Embedding of span sequences) Let

$$s = G_0 \overset{g_1}{\leftarrow} C_1 \overset{h_1}{\rightarrow} G_1 \leftarrow \cdots \rightarrow G_{n-1} \overset{g_n}{\leftarrow} C_n \overset{h_n}{\rightarrow} G_n$$

and

$$s' = G_0' \overset{g_1'}{\leftarrow} C_1' \overset{h_1'}{\rightarrow} G_1' \leftarrow \cdots \rightarrow G_{m-1}' \overset{g_m'}{\leftarrow} C_m' \overset{h_m'}{\rightarrow} G_m'$$

be two span sequences with $m \leq n$. The span sequence s' is *embeddable* into s if there exists a surjective order preserving function

$$\alpha\colon \{0,\dots,n\} \to \{0,\dots,m\} \text{ with } i \leq i' \implies \alpha(i) \leq \alpha(i')$$

and a family of monomorphisms

$$\psi = \{\gamma_0\colon G'_0 \to G_0, \gamma_i\colon G_{\alpha(i)} \to G_i, \chi_i\colon X_i \to C_i\}_{i\in\{1,\dots n\}}$$

where

$$X_i = \begin{cases} G_{\alpha(i)} & \text{if } \alpha(i-1) = \alpha(i) \\ C_{\alpha(i)} & \text{otherwise} \end{cases}$$

such that all triangles and squares commute. The family of monomorphisms ψ is called the *embedding morphism*.

Basically, each span of the embeddable sequence matches into a span of the target sequence, while on the other hand a span in the target sequence is matched by either a span or a single graph of the embeddable sequence, as is depicted below

$$\begin{tikzcd}
 & G'_{j-1} \arrow[dl, "\gamma_{i-2}"'] & C'_j \arrow[l, "g'_j"'] \arrow[r, "h'_j"] \arrow[dl, "\chi_{i-1}"'] & G'_j \arrow[dl, "\gamma_{i-1}"'] \arrow[d, "\chi_i"'] \arrow[dr, "\gamma_i"] & C'_{j+1} \arrow[l, "g'_{j+1}"'] \arrow[r, "h'_{j+1}"] \arrow[dr, "\chi_{i+1}"] & G'_{j+1} \arrow[dr, "\gamma_{i+1}"] & \\
G_{i-2} & C_{i-1} \arrow[l, "g'_{i-1}"] \arrow[r, "h'_{i-1}"'] & G_{i-1} & C_i \arrow[l, "g'_i"] \arrow[r, "h'_i"'] & G_i & C_{i+1} \arrow[l, "g'_{i+1}"] \arrow[r, "h'_{i+1}"'] & G_{i+1}
\end{tikzcd}$$

Part of the usefulness of grammar refinement is given by the fact that the set of possible derivations of original and refined grammar is similar.

Definition 3.24 (Application embeddable grammar) A grammar $\mathcal{G}'$ is *application embeddable* into $\mathcal{G}$ if for each sequential derivation ρ' possible in $\mathcal{G}'$, there exists a derivation ρ that is possible in $\mathcal{G}$ such that the application sequence of ρ' is embeddable into the application sequence of ρ.

Theorem 3.25 (Application embeddability of sliced grammars). *Sliced grammars of chemical graph grammars are embeddable into the original grammar.*

Proof. We inductively prove that the application sequence of each sequential derivation in $\mathcal{G}'$ is embeddable into an application sequence in $\mathcal{G}$. The base case for the trivial derivation of the start graph $G'_0 \Rightarrow^*_{\mathcal{G}'} G'_0$ is obviously true by definition of sliced graph grammars (see Definition 3.22).

Now let $\rho' = G'_0 \Rightarrow^*_{\mathcal{G}'} G'_j$ and $\rho = G_0 \Rightarrow^*_{\mathcal{G}} G_i$ be sequential derivations such that the application sequences $\operatorname{app}(\rho')$ is embeddable into $\operatorname{app}(\rho)$. For the induction step, it is sufficient to prove that for each direct derivation $\rho' = G'_j \xRightarrow{p', m^j_{L'}} G'_{j+1}$ with $(p' \colon L' \xleftarrow{l'} I' \xrightarrow{r'} R') \in \Pi'$ there exists a production $(p \colon L \xleftarrow{l} I \xrightarrow{r} R) \in \Pi$ and a direct derivation $\rho = G_i \xRightarrow{p, m_L} G_{i+1}$ such that the former production is a specialization of the latter. The application span $\operatorname{app}(\rho') = G'_j \leftarrow C_{j+1} \rightarrow G'_{j+1}$ together with a suitable production name is a specialization of p. This production is application safe (see Lemma 3.17) and hence applicable on G_i since $G'_j \leq G_i$. The rest follows from Proposition 3.18. □

This is an intuitive result. A refined grammar has a smaller or equal subgraph as the original grammar and a specialized production yields the same rewrite step on a given graph whenever both can be applied. Since the specialized productions are application safe, none previously inapplicable production becomes applicable in the slice due to previously unsatisfied dangling conditions. Hence, there is no way to produce derivations in the slice that do not match the original grammar.

Grammar equivalence with regard to a pattern is a desireable as well as intuitivly expected property of graph grammar slices. This is in fact true due to the use of specialized productions in the definition of graph grammar slicing: all pattern change identifier are captured within a specific specialized rule:

Lemma 3.26 (Pattern change capturing specializations). *Let $f \colon F \to G$ be a match of F in a graph G and let $\rho = G \xRightarrow{p, m_L} H$ be a derivation of a linear production $(p \colon L \xleftarrow{l} I \xrightarrow{r} R)$. Then there exists a specialization $(p' \colon L' \xleftarrow{l'} I' \xrightarrow{r'} R')$ of p on F and matches $m'_L \colon L' \to G$ and $f' \colon F \to L'$ such that $f = m'_L \circ f'$ and*

$$\rho/f = p'/f'.$$

Proof. The pullback $F \xleftarrow{d_F} D \xrightarrow{d_L} L$ of f and m always exists in an adhesive category. The morphisms d_F and d_L are monic [46, V.7], so the pushout $F \xrightarrow{d_L} L' \xrightarrow{\alpha_L} X \xleftarrow{f} C \xleftarrow{d_F} F$ exists as well. The specialization $(p'\colon L' \xleftarrow{l'} I' \xrightarrow{r'} R')$ of p over (d_F, d_L) exists by Theorem 3.16 and is applicable on G due to the universal property of L'. The rest follows from Lemma 3.26. □

Since a sliced grammar contains specializations on all pattern matches, this directly leads to grammar equiavlence.

Theorem 3.27. *A slice of a chemical graph grammar over a pattern F is F-equivalent to the sliced grammar.*

Proof. Assume $\mathcal{G}' = (T', G'_0, P', \Pi')$ is a slice of a chemical graph grammar $\mathcal{G} = (T, G_0, P, \Pi)$. It follows from Theorem 3.21 and from the definition of graph grammar slices (Definition 3.22) that all productions in $\zeta^F_{\mathcal{G}}(\Pi)$ are applicable in $\mathcal{G}'$. Hence, by Lemma 3.11 and Lemma 3.26, it is

$$\chi^F_{\mathcal{G}'}(\zeta^F_{\mathcal{G}}(\Pi)) = \chi^F_{\mathcal{G}}(\Pi).$$

□

3.4 Biochemical Application and Construction of Graph Grammar Slices

The motivation for the development of graph grammar slicing originates from the need to reduce comprehensive models of biochemical systems to allow for quick answers of simple questions. In this section we will discuss possible use-cases of graph grammar slicing. Furthermore, a brief outlook into algorithmical aspects of graph grammar slices is given.

3.4.1 Modularisation and Reduction of Reactive Systems

As discussed in Chapter 2, many models of reactive systems are graph-based or can be expressed using graphs and graph rewriting. Simulations of such systems become increasingly difficult and even pratically impossible with the escalating detail and complexity of models. In such cases, graph grammar slicing may be applied to extract the parts of a model that are relevant for a pattern of interest.

A graph-based model of a chemical or biochemical given by reaction rules can be regarded as a graph grammar, as is described Chapter 2. Graph grammar slicing may be used to remove irrelevant detail from such model and is therefore similar to the model reduction techniques discussed in Section 2.6.2. In fact, the author believes that each of these methods are special cases of graph grammar slicing, although a proof of this statement is yet to be given and may be part of future work.

3.4.2 An Algorithmic Approach to Graph Grammar Slicing

Since construction of specialized productions is based on partial matches, we generate and select those partial matches that have the potential to result in a specialized production with the required properties. We give an algorithm to generate pattern changing productions through the means of sufficiently restrictive partial matches.

Given a pattern F and a production $(p\colon L \xleftarrow{l} I \xrightarrow{r} R)$, a *proper specialization nucleus* is a partial match $d = F \xleftarrow{d_F} D \xrightarrow{d_X} X$, $X \in \{L, R\}$ together with its pushout $q = F \xrightarrow{q_F} Q \xleftarrow{q_X} X$ such that

- d is pattern modifying,
- Q is a subgraph of a molecule and
- the dangling condition is satisfied for $q_X\colon X \to Q$.

For each proper specialization nucleus there exists a F-changing specialized production and *vice versa*. However, not each F-changing production is necessarily applicable in a given grammar.

Note that the pushouts of proper specialization nuclei are conceptually related to the idem pushouts as described by Leifer and Milner [45]. Idem pushouts are the basis for the model reduction technique introduced by Danos et al. [25]. Furthermore, proper specialization nuclei are similar to the fragment construction technique of Feret et al. [37].

As demonstrated by Algorithm 1, the explicit generation of proper specialization nuclei is possible. The function takes a pattern graph and a monic graph morphism and returns all proper specialization matches. Note that the innermost test (if a graph is in the class of molecules) is highly application dependent and may be non trivial. However, for protein graphs and chemical graphs there exist axiomatic definitions which are checked easily.

Algorithm 1 Generate all proper specialization nuclei

```
function PROPERSPECNUCLEI($F$, $x\colon I \to X$)
    $N \leftarrow \emptyset$
    $\Delta \leftarrow$ elements of $(X - x(I))$
    for all $D \subseteq X$ with $D \cap \Delta \neq \emptyset$ do
        for all $d_F \in$ SUBGRAPHISOMORPHISMS($F$,$X[D]$) do
            $(Q, q_F, q_X) \leftarrow$PUSHOUT($F \overset{d_F}{\leftarrow} X[D] \overset{id}{\rightarrow} X$)
            if $Q \in \mathrm{sub}(\mathcal{M})$ then                ▹ Is pushout a molecule pattern?
                if DANGLINGCONTITIONSATISFIED($q_I, x$) then
                    $N \leftarrow N \cup \{(Q, q_F, q_X)\}$
                end if
            end if
        end for
    end for
    return $N$
end function
```

The inner workings of Algorithm 1 are straightforward but leave much potential for optimization for specific applications. For example in the case of protein graphs, a pushout of a partial match does not exist in the category of protein graphs, unless the partial match contains the neighboring protein nodes of each site node. Observations like this might lead to a reduction of the number of generated and discarded pushouts. However, these special cases are disregarded here to ensure generality of the algorithm.

The generation of specialized linear productions closely follows Definition 3.15. Algorithm 2 allows for specializations on the left- or right hand side, respectively. It is sufficient to handle left-specialization of a production, that is specialization on pattern destruction: A specialization on the right hand side may be traced back to the the left hand side case by inverting the production, specializing on the left hand side and then invert the resulting production span.

Algorithm 2 Creating all specializations of a rule.

```
function specProds($F$, $\pi$)
    return leftSpecProds($F$, $\pi$) $\cup$ rightSpecProds($F$, $\pi$)
end function

function leftSpecProds($F$, $\pi$)
    Let $\pi$ be given by $(p : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R)$
    $\mathcal{S} \leftarrow \emptyset$
    $\mathcal{D} \leftarrow$ properSpecNuclei($F, l\colon I \rightarrow L$)
    for all $(Q, q_F, q_p) \in \mathcal{D}$ do
        $s \leftarrow$ applyRule($Q, p, q_L$)          ▷ application span of $Q \overset{p,q_L}{\Longrightarrow} Q'$
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{(p : s)\}$
    end for
    return $\mathcal{S}$                              ▷ Set of specialized rules
end function

function rightSpecProds($F$, $\pi$)
    $\mathcal{S} \leftarrow$leftSpecProds($F$, $\pi^{-1}$)
    for all $\pi \in \mathcal{S}$ do
        $\pi \leftarrow \pi^{-1}$
    end for
    return $\mathcal{S}$
end function
```

The specialization procedure itself takes all proper specialization nuclei, generated by Algorithm 1 described above, and applies the production on each pushout object Q using the match q_L. The application span of this derivation is then used as the production span of a specialized production.

The naïve graph grammar slicing algorithm presented on page 60 is based on Definition 3.22. The productions of the slice are generated by specializing on each pattern for which the new grammar must be a slice. Each pattern on which the specialization procedure has been applied is kept track of. Therefore, no pattern is processed twice. As a result, if the set of molecules

$\mathcal{M}$ is finite, then so is the set of graph patterns and the algorithm will terminate.

The slicing algorithm presented here lacks several important features such as checks for occurrence of patterns in a language and removal of inapplicable specialized productions. This has implications for the average runtime and termination for infinite sets of molecules. However, the purpose of Algorithm 3 is just to demonstrate that computational generation of graph grammar slices is possible, so the above details are disregarded intentionally.

Algorithm 3 Generate a graph grammar slice with regard to a pattern F.

```
function grammarSlice($\mathcal{G}$, $F$)
    Let $\mathcal{G}$ be given by $\mathcal{G} = (T, G_0, P, \Pi)$
    $\mathcal{F} \leftarrow \{F\}$                          ▷ patterns to be tracked
    $\mathcal{Q} \leftarrow \emptyset$                      ▷ tracked (processed) patterns
    $\Pi' \leftarrow \emptyset$                             ▷ specialized productions
    while $\mathcal{F} \neq \emptyset$ do     ▷ get specialized productions for patterns
        choose $F \in \mathcal{F}$ and remove it from the set
        for all $\pi \in \Pi$ do
            for all $\pi' \in$ specProds($F$,$\pi$) do
                $L \leftarrow LHS(\pi')$
                if $L \notin \mathcal{Q}$ then           ▷ was pattern processed already?
                    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{L\}$
                    $\mathcal{F} \leftarrow \mathcal{F}\{L\}$
                    $\Pi' \leftarrow \Pi' \cup \{\pi'\}$
                end if
            end for
        end for
    end while
                                                   ▷ determine new start graph
    $G_0' \leftarrow$ empty graph
    for all $Q \in \mathcal{Q}$ do
        for all matches $m\colon Q \rightarrow G_0$ of $Q$ in $G_0$ do
            $G_0' \leftarrow G_0' \cup G_0[m(Q)]$        ▷ graph union with subgraph of $G_0$
        end for
    end for
    return $(G_0', T, P', \Pi')$
end function
```

4

Conclusions and Outlook

In this chapter an overview of the results of this thesis is given. I furthermore evaluate the results with the respect to the aims formulated in Chapter 1. The potential and limitations of the graph grammar slicing approach presented in Chapter 3 are discussed, including its value for models similar to those presented in Chapter 2. Finally we outline future work on graph grammar slicing and its application in artificial chemistry and systems biology.

4.1 Evaluation

The main aim of this thesis, as formulated in the introduction, was the development of a model reduction technique for graph grammars for biological and chemical applications.

It has been demonstrated that small chemical molecules as well as large protein complexes can be modeled using constrained typed graphs. The interactions and transformations of the species can been modeled as typed graph transformations. This demonstrates how the same formal systems can be applied on similar natural systems.

I have specified a slicing approach for graph grammars modeling chemical or biochemical systems. This technique allows for the generation of graph grammars that result in languages and in derivations which are embeddable into the original grammar. The characteristic behavior of the grammar with regard to a pattern graph is conserved in the slice.

Given a graph grammar model of a chemical system, the slicing approach developed here allows for modularization of the system. This can lead to a significant reduction in the number of graphs and reactions in the model.

A limitation of the slicing approach is its current restriction to graph grammars. Graph grammars are a suitable model only for a relatively small number of molecules. Simulation of larger complex systems requires a transformation of the graph-based model into Petri nets, stochastic systems or networks of differential equations. However, there exist indications that the slicing approach can be transfered to these models, yielding similar embedding properties as graph grammar slices.

4.2 Future Work

The description of graph grammar slicing in this thesis is restricted to chemical graph grammars. This avoids the problem of dangling conditions and applicability of rules by removal of subgraphs and implies existence of constructs necessary for grammar slicing. It might be worth exploring if graph grammar slicing can be generalized to non-chemical graph grammars and adhesive grammars in general.

The formalization of graph grammar slicing is largely based on category theoretical constructs, so the concept of grammar slicing is easily tranfered and extended to other data structures than typed graphs. Furthermore, extensions to slicing with regard to patters are simple to develop and describe. We give a brief overview of possible extensions and improvements of the approach presented here.

Some small molecules, like adenosine triphospate (ATP), are abundant within biochemical systems and participate in numberous reactions. Their regulation *in vivo* ensures a nearly constant concentration within a cell. It is therefore reasonable to assume a fixed value for such species in any simulation, so this must be reflected in the way a slice is created. Instead of including the complete model of cellular ATP metabolism, the slicing algorithms should refrain from slicing the system with regard to ATP. This is easily implemented using additional checks before recursing into slicing with regard to a pattern.

An interesting alternative for typed graphs are graphs labeled with a structured alphabet, which might offer a solution to combinatorial explosions caused by minor model pertubations [24] and reactions of unspecific enzymes. Instead of using a flat set of labels, a structure is imposed on the set of labels, yielding SC-graphs, which were introduced by Parisi-Presicce et al. [53]. This greatly improves expressiveness of productions: For example, in a biochemical context, this would allow to describe unspecific phosphatase reactions. A single label could be defined to match a whole set of phosphorylation sites and the number of productions would be greatly reduced, since one production suffices to describe dephosphorylation of different proteins. Compare this to the typed graph-based model which requires specific productions for each substrate. The method of grammar slicing would remain valid for grammars of SC-graphs.

Another straightforward extension is slicing with regard to pattern change identicators. This would be useful in cases where the concentration of a pattern is irrelevant, but instead the rate of a class of reactions is to be followed. This would require specialization on pattern change indicators and might then be traced back to slicing with regard to patterns. Especially in the case of flux analyses of metabolic pathways [1, 50], this might be of value. However, the models in this thesis are by themselves not sufficient for metabolic analyses, which require models to account for small molecules as well as for large protein complexes. A possible approach might be given by multilevel graph representations and transformation of graph grammars as described by [52] but this may as well prove to be too complex.

Graph patterns are conceptually linked to observables, macroscopically measurable properties of low-level systems. Grammar slicing enables to focus on specific subparts of the model. This is a also desireable feature for simulations generated from rule-based models. Relationship of graph grammar slicing with such simulations is yet to be explored.

Our theory of graph grammar slicing is formulated using category theory. Ehresmann and Vanbremeersch [30, 31] suggested category theory as a foundation to formulate hierarchical models of biological systems. Their approach makes excessive use of patterns in the description of hierarchies, so the presented slicing technique may be transferable and useful in this context.

Bibliography

[1] Doug K. Allen, Igor G. L. Libourel, and Yair Shachar-Hill. Metabolic flux analysis in plants: coping with complexity. *Plant, Cell & Environment*, 32(9):1241–1257, 2009. doi: 10.1111/j.1365-3040.2009.01992.x. 63

[2] Oana Andrei. *Un calcul de réécriture de graphes: applications à la biologie et aux systèmes autonomes*. PhD thesis, Institut National Polytechnique de Lorraine, 2008. 20

[3] Oana Andrei and Hélène Kirchner. A biochemical calculus based on strategic graph rewriting. In *The Third International Conference on Algebraic Biology, AB08, Hagenberg Austria*, 2008. 20

[4] Gil Benkö, Christoph Flamm, and Peter F. Stadler. Generic properties of chemical networks: Artificial chemistry based on graph rewriting. In *Advances in Artificial Life*, volume 2801 of *Lecture Notes in Computer Science*, pages 10–19. Springer Berlin / Heidelberg, 2003. doi: 10.1007/978-3-540-39432-7_2. 9

[5] Gil Benkö, Christoph Flamm, and Peter F. Stadler. A graph-based toy model of chemistry. *J. Chem. Inf. Comput. Sci.*, 43(4):1085–1093, 2003. doi: 10.1021/ci0200570. 8, 9

[6] Michael L. Blinov, James R. Faeder, Byron Goldstein, and William S. Hlavacek. Bionetgen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291, 2004. doi: 10.1093/bioinformatics/bth378. 18

[7] Michael L. Blinov, James R. Faeder, Byron Goldstein, and William S. Hlavacek. A network model of early events in epidermal growth factor receptor signaling that accounts for combinatorial complexity. *Biosystems*, 83(2–3):136–151, 2006. doi: 10.1016/j.biosystems.2005.06.014. 13, 14

[8] Michael L. Blinov, Jin Yang, James R. Faeder, and William S. Hlavacek. Graph theory for rule-based modeling of biochemical networks. In Corrado Priami, Anna Ingólfsdóttir, Bud Mishra, and Hanne Nielson, editors, *Transactions on Computational Systems Biology VII*, volume 4230 of *Lecture Notes in Computer Science*, pages 89–106. Springer Berlin / Heidelberg, 2006. doi: 10.1007/11905455_5. 15, 16, 18

[9] N. M. Borisov, A. S. Chistopolsky, J. R. Faeder, and B. N. Kholodenko. Domain-oriented reduction of rule-based network models. *IET Systems Biology*, 2(5):342–351, 2008. doi: 10.1049/iet-syb:20070081. 21

[10] Muffy Calder, Vladislav Vyshemirsky, David Gilbert, and Richard Orton. Analysis of signalling pathways using continuous time markov chains. *Transactions on Computational Systems Biology VI*, 4220:44–67, 2006. 18

[11] Yang Cao, Daniel T. Gillespie, and Linda R. Petzold. Accelerated stochastic simulation of the stiff enzyme-substrate reaction. *Journal of Chemical Physics*, 123(14):144917, 2005. doi: 10.1063/1.2052596. 18

[12] Luca Cardelli. Brane calculi. In Vincent Danos and Vincent Schachter, editors, *Computational Methods in Systems Biology*, volume 3082 of *Lecture Notes in Computer Science*, pages 257–278. Springer Berlin / Heidelberg, 2005. doi: 10.1007/978-3-540-25974-9_24. 20

[13] Ron Caspi, Tomer Altman, Joseph M. Dale, Kate Dreher, Carol A. Fulcher, Fred Gilham, Pallavi Kaipa, Athikkattuvalasu S. Karthikeyan, Anamika Kothari, Markus Krummenacker, Mario Latendresse, Lukas A. Mueller, Suzanne Paley, Liviu Popescu, Anuradha Pujar, Alexander G. Shearer, Peifen Zhang, and Peter D. Karp. The MetaCyc database of metabolic pathways and enzymes and the BioCyc collection of pathway/genome databases. *Nucleic Acids Research*, 38(suppl 1):D473–D479, 2010. doi: 10.1093/nar/gkp875. 1

[14] Federica Ciocchetta and Jane Hillston. Bio-PEPA: a framework for the modelling and analysis of biological systems. *Theoretical Computer Science*, 410(33):3065–3084, 2008. 18, 20

[15] Holger Conzelmann. *Mathematical Modeling of Biochemical Signal Transduction Pathways in Mammalian Cells: A Domain-Oriented Approach to Reduce Combinatorial Complexity*. PhD thesis, Fakultät Konstruktions-, Produktions- und Fahrzeugtechnik der Universität Stuttgart, 2008. URL `http://elib.uni-stuttgart.de/opus/volltexte/2009/3881/pdf/DissertationConzelmann.pdf`. 21

[16] Holger Conzelmann, Julio Saez-Rodriguez, Thomas Sauter, Boris N. Kholodenko, and Ernst D Gilles. A domain-oriented approach to the reduction of combinatorial complexity in signal transduction networks. *BMC Bioinformatics*, 7(1):34, 2006. doi: 10.1186/1471-2105-7-34. 21

[17] Holger Conzelmann, Dirk Fey, and Ernst D. Gilles. Exact model reduction of combinatorial reaction networks. *BMC Systems Biology*, 2:78, 2008. doi: 10.1186/1752-0509/2/78. 3, 21

[18] A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and J. Padberg. The category of typed graph grammars and its adjunctions with categories of derivations. In Janice Cuny, Hartmut Ehrig, Gregor Engels, and Grzegorz Rozenberg, editors, *Graph Grammars and Their Application to Computer Science*, volume 1073 of *Lecture Notes in Computer Science*, pages 56–74. Springer Berlin / Heidelberg, 1996. doi: 10.1007/3-540-61228-9_79. 19

[19] Andrea Corradini, Ugo Montanari, Franceses Rossi, Hartmut Ehrig, Reiko Heckel, and Michael Löwe. *Handbook of graph grammars and computing by graph transformation*, volume I, foundations, chapter Algebraic approaches to graph transformation part I: Basic concepts and double pushout approach, pages 163–245. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997. ISBN 98-102288-48. 9, 10, 16, 24

[20] Andrea Corradini, Tobias Heindel, Frank Hermann, and Barbara König. Sesqui-pushout rewriting. In Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro, and Grzegorz Rozenberg, editors, *Graph Transformations*, volume 4178 of *Lecture Notes in Computer Science*, pages 30–45. Springer Berlin / Heidelberg, 2006. doi: 10.1007/11841883_4. 27, 28, 33

[21] Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1):69–110, 2004. doi: 10.1016/j.tcs.2004.03.065. 8, 15, 16, 20, 21

[22] Vincent Danos, Jérôme Feret, Walter Fontana, and Jean Krivine. Abstract interpretation of cellular signalling networks. In Francesco Logozzo, Doron Peled, and Lenore Zuck, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 4905 of *Lecture Notes in Computer Science*, pages 83–97. Springer Berlin / Heidelberg, 2008. doi: 10.1007/978-3-540-78163-9_11. 3, 18, 21

[23] Vincent Danos, Jérôme Feret, Walter Fontana, Russel Harmer, and Jean Krivine. Abstracting the ODE semantics of rule-based models: Exact and automatic model reduction. In *16th International Static Analysis Symposium (SAS) 2009*, 2009. 21

[24] Vincent Danos, Jérôme Feret, Walter Fontana, Russel Harmer Harmer, and Jean Krivine. Rule-based modelling and model perturbation. *Transactions on Computational Systems Biology*, 11:116–137, 2009. 63

[25] Vincent Danos, Jérôme Feret, Walter Fontana, Russel Harmer, and Jean Krivine. Abstracting the differential semantics of rule-based models: exact and automated model reduction. *Logic in Computer Science*, 2010. 21, 56

[26] René David and Hassane Alla. Autonomous and timed continuous Petri nets. *Lecture Notes in Computer Science*, 674:71–90, 1993. doi: 10.1007/3-540-56689-9_40. 19

[27] René David and Hassane Alla. *Discrete, continuous, and hybrid Petri nets*. Springer Berlin Heidelberg New York, 2005. 19

[28] Peter Dittrich, Jens Ziegler, and Wolfgang Banzhaf. Artificial chemistries–a review. *Artificial Life*, 7(3):225–275, 2001. doi: 10.1162/106454601753238636. 7

[29] Roy Dyckhoff and Walter Tholen. Exponentiable morphisms, partial products and pullback complements. *Journal of Pure and Ap-*

plied Algebra, 49(1-2):103–116, 1987. ISSN 0022-4049. doi: 10.1016/0022-4049(87)90124-1. 27

[30] Andrée C. Ehresmann and Jean-Paul Vanbremeersch. Hierarchical evolutive systems: A mathematical model for complex systems. *Bulletin of Mathematical Biology*, 49(1):13–50, 1987. doi: 10.1007/BF02459958. 63

[31] Andrée C. Ehresmann and Jean-Paul Vanbremeersch. *Memory Evolutive Systems: Hierarchy, Emergence, Cognition*. Elsevier, 1 edition, 2007. 63

[32] Hartmut Ehrig and Barbara König. Deriving bisimulation congruences in the DPO approach to graph rewriting. In Igor Walukiewicz, editor, *Foundations of Software Science and Computation Structures*, volume 2987 of *Lecture Notes in Computer Science*, pages 151–166. Springer Berlin / Heidelberg, 2004. doi: 10.1007/978-3-540-24727-2_12. 45

[33] Hartmut Ehrig, Reiko Heckel, Martin Korff, Michael Löwe, Leila Ribeiro, Annika Wagner, and Andrea Corradini. *Handbook of graph grammars and computing by graph transformation*, volume I, foundations, chapter Algebraic approaches to graph transformation. Part II: single pushout approach and comparison with double pushout approach, pages 247–312. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997. ISBN 98-102288-48. 16

[34] Hartmut Ehrig, Reiko Heckel, Mercè Llabrés, Fernando Orejas, Julia Padberg, and Grzegorz Rozenberg. Double-pullback graph transitions: A rule-based framework with incomplete information. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Theory and Application of Graph Transformations*, volume 1764 of *Lecture Notes in Computer Science*, pages 341–348. Springer Berlin / Heidelberg, 2000. doi: 10.1007/978-3-540-46464-8_7. 27

[35] Steven Eker, Merrill Knapp, Keith Laderoute, Patrick Lincoln, and Carolyn Talcott. Pathway logic: Executable models of biological networks. *Electronic Notes in Theoretical Computer Science*, 71:144 – 161, 2004. ISSN 1571-0661. doi: 10.1016/S1571-0661(05)82533-2. WRLA 2002, Rewriting Logic and Its Applications. 19, 20

[36] James R. Faeder, Michaell Blinov, Byron Goldstein, and Williams Hlavacek. Rule-based modeling of biochemical networks. *Complexity*, 10(4):22–41, 2005. doi: 10.1002/cplx.20074. 15

[37] Jérôme Feret, Jean Krivine Vincent Danos, Russel Harmer, and Walter Fontana. Internal coarse-graining of molecular systems. *Proceedings of the National Academy of Sciences*, 106(16):6453–6458, 2009. doi: 10.1073/pnas.0809908106. 21, 56

[38] David Gilbert and Monika Heiner. From petri nets to differential equations - an integrative approach for biochemical network analysis. In *Petri Nets and Other Models of Concurrency - ICATPN 2006*, volume 4024 of *Lecture Notes in Computer Science*, pages 181–200. Springer, 2006. doi: 10.1007/11767589_11. 19

[39] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977. doi: 10.1021/j100540a008. 18

[40] Ralf Hofestädt. A Petri net application to model metabolic processes. *Systems Analysis Modeling Simulation*, 16:113–122, October 1994. ISSN 0232-9298. 19

[41] Minoru Kanehisa, Susumu Goto, Miho Furumichi, Mao Tanabe, and Mika Hirakawa. KEGG for representation and analysis of molecular networks involving diseases and drugs. *Nucleic Acids Research*, 38(suppl 1):D355–D360, 2010. doi: 10.1093/nar/gkp896. 1

[42] Adalbert Kerber, Reinhard Laue, Markus Meringer, and Christoph Rücker. Molecules in silico: A graph description of chemical reactions. *Journal of Chemical Information and Modeling*, 47(3):805–817, 2007. 12

[43] Ingrid M. Keseler, César Bonavides-Martínez, Julio Collado-Vides, Socorro Gama-Castro, Robert P. Gunsalus, D. Aaron Johnson, Markus Krummenacker, Laura M. Nolan, Suzanne Paley, Ian T. Paulsen, Martin Peralta-Gil, Alberto Santos-Zavaleta, Alexander Glennon Shearer, and Peter D. Karp. EcoCyc: A comprehensive view of Escherichia coli

biology. *Nucleic Acids Research*, 37(suppl 1):D464–D470, 2009. doi: 10.1093/nar/gkn751. 1

[44] Stephen Lack and Pawel Sobociński. Adhesive categories. *Lecture Notes in Computer Science*, 2987:273–288, 2004. 24, 26, 33

[45] James Leifer and Robin Milner. Deriving bisimulation congruences for reactive systems. In Catuscia Palamidessi, editor, *CONCUR 2000 — Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 243–258. Springer Berlin / Heidelberg, 2000. doi: 10.1007/3-540-44618-4_19. 56

[46] Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag New-York, second edition, 1998. 24, 55

[47] Aneil Mallavarapu, Matthew Thomson, Benjamin Ullian, and Jeremy Gunawardena. Programming with models: modularity and abstraction provide powerful capabilities for systems biology. *Journal of the Royal Society Interface*, 6(32):257–270, 2009. doi: 10.1098/£rsif.2008.0205. 18

[48] Oliver Mason and Mark Verwoerd. Graph theory and networks in biology. *Systems Biology, IET*, 1(2):89–119, March 2007. doi: 10.1049/iet-syb:20060038. 8

[49] Charles E. Mortimer and Ulrich Müller. *Chemie*. Thieme, Stuttgart, 8 edition, 2003. 17

[50] Jens Niklas, Konstantin Schneider, and Elmar Heinzle. Metabolic flux analysis in eukaryotes. *Current Opinion in Biotechnology*, 21(1):63 – 69, 2010. ISSN 0958-1669. doi: 10.1016/j.copbio.2010.01.011. 63

[51] Kanae Oda, Yukiko Matsuoka, Akira Funahashi, and Hiroaki Kitano. A comprehensive pathway map of epidermal growth factor receptor signaling. *Molecular Systems Biology*, 1:2005.0010, 2005. doi: 10.1038/msb4100014. 13

[52] Francesco Parisi-Presicce. Transformations of graph grammars. In Janice Cuny, Hartmut Ehrig, Gregor Engels, and Grzegorz Rozenberg,

editors, *Graph Grammars and Their Application to Computer Science*, volume 1073 of *Lecture Notes in Computer Science*, pages 428–442. Springer Berlin / Heidelberg, 1996. doi: 10.1007/3-540-61228-9_103. 63

[53] Francesco Parisi-Presicce, Hartmut Ehrig, and Ugo Montanari. Graph rewriting with unification and composition. In *Graph-Grammars and Their Application to Computer Science*, pages 496–514, 1986. doi: 10.1007/3-540-18771-5_72. 63

[54] Christopher V. Rao and Adam P. Arkin. Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the gillespie algorithm. *Journal of Chemical Physics*, 118(11):4999–5010, 2003. doi: 10.1063/1.1545446. 18

[55] Venkatramana N. Reddy, Michael L. Mavrovouniotis, and Michael N. Liebman. Petri net representations in metabolic pathways. In *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology*, pages 328–336. AAAI Press, 1993. ISBN 0-929280-47-4. 19

[56] Aviv Regev, William Silverman, and Ehud Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Pacific Symposium on Biocomputing 6*, pages 459–470, 2001. 20

[57] Francesc Rosselló and Gabriel Valiente. Chemical graphs, chemical reaction graphs, and chemical graph transformation. *Electronic Notes in Theoretical Computer Science*, 127:157–166, 2005. doi: 10.1016/j.entcs.2004.12.033. 12, 37

[58] Gyula Svehla. Nomenclature of kinetic methods of analysis (iupac recommendations 1993). *Pure and Applied Chemistry*, 65(10):2291–2298, 1993. doi: 10.1351/pac199365102291. Commission on Analytical Nomenclature. 17

[59] Carolyn Talcott. Pathway logic. In Marco Bernardo, Pierpaolo Degano, and Gianluigi Zavattaro, editors, *Formal Methods for Computational Systems Biology*, volume 5016 of *Lecture Notes in Com-*

puter Science, pages 21–53. Springer Berlin / Heidelberg, 2008. doi: 10.1007/978-3-540-68894-5_2. 20

[60] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3:121–189, 1995. 23

[61] Mark Weiser. Program slicing. In *ICSE '81: Proceedings of the 5th international conference on Software engineering*, pages 439–449, Piscataway, NJ, USA, 1981. IEEE Press. ISBN 0-89791-146-6. 23

[62] Olaf Wolkenhauer. Systems biology: The reincarnation of systems theory applied in biology? *Brief Bioinform*, 2(3):258–270, 2001. doi: 10.1093/bib/2.3.258. 1