Graph Grammars as a Formal Foundation for Biological Process Diagrams

Ramon Medrado[1] and Leila Ribeiro[2]

Instituto de Informática, Universidade Federal do Rio Grande do Sul, Brazil,
{ramon.medrado,leila}@inf.ufrgs.br
WWW home page: http://www.inf.ufrgs.br/∼rgmedrado

Abstract. Biological pathways represent interactions between complex chemical entities that occur in cells. There are some graphical notations to describe biological pathways. Among them, process diagrams have been widely used. A process diagram is essentially a graph in which vertices and edges represent biological components as well as relationships among these components, and there is a graphical notation associated with each different element. In this paper we give a formal foundation for biological process diagrams, by defining their (concrete and abstract) syntax and semantics using a formalism called graph grammars. We first build a graph called BioProc Graph, describing the meta-model of process diagrams. Instances of this BioProc graph are concrete process diagrams modeling biological pathways. They can be constructed by (graph) rules defining the syntax of the process diagram language. The semantics of the resulting diagrams can also be expressed by (graph) rules that change the amount of substances present in the given pathway. Moreover, since we have a meta-model formally defined as a graph, we can use it as a basis for model evolution using (graph) rules.

1 Introduction

One of the topics of investigation in systems biology is the complex interactions representation's between chemical entities (proteins, substrates, metabolites, etc.) that happen at a molecular level in cells. Such interactions, called biological pathways, are very important for the vital functions of living organisms. There are 4 types of biological pathways [1]: metabolic, signaling, regulatory and molecular interaction pathways.

Biological pathways are complex to describe and, most important, understand in an intuitive way. There is a need for description languages that generate models with predictive value, that is, that enable analysis methods to verify whether the model corresponds to the real one, as well as to predict behaviours, either as a stand alone pathway or as a part of a complex network. Such models would increase the understanding of living systems by relating the basic molecule's behaviour to complex behaviours.

Although it is possible to build mathematical models (like differential equations) to describe the behaviour of a system from scratch just by observing the system, using mathematical language directly makes the understanding of the proposed model more difficult, specially for biologists. A visual representation widely used by biologists to describe pathways is the Kitano's process diagrams notation [2]. Here we formalize this visual language, defining its abstract and concrete syntaxes, as well as providing a formal semantics for the generated diagrams. Moreover, we show how to (formally) describe evolutions of process diagrams. This is relevant since models are usually constructed incrementally, and a formal specification of how a model shall be transformed provides a basis for analysis of the evolution process itself. Our approach is based on Graph Grammars, a formalism that provides natural representations for models that are inherently based on graphs (as it is the case for biological processes diagrams). We use graph grammars to model the three different aspects discussed above: syntax, semantics and evolution.

This paper is structured as follows: Section 2 shortly reviews the Kitano's notation to describe biological pathways and presents the metamodel of process diagrams, including abstract and concrete representations; the rules of the syntax are shown in Section 3; Section 4 defines the semantics of process diagrams using Stochastic Graph Transformation Systems; Section 5 discusses the idea of model evolution, and finally Section 6 presents our conclusions.

2 A Meta Model for Process Diagrams: BioProc Graph

Kitano et al [2] describes a biological process diagram as a set of state nodes (entities of the biological process, such as proteins, RNA or genes) and a set of transition arcs (modulations of the reactions, such as association, dissociation, etc.). The graphical symbols used in the Kitano notation can be found in [2].

Although there is a formal description of the underlying graphs that represent process diagrams, this description covers only part of the aspects of the diagrams. Thus, it can be considered as a semi-formal description. Moreover, rules for construction of such diagrams as well as their meaning (semantics) were only informally described. Formal descriptions eliminate ambiguities and doubts that may occur in specifications and serve as basis for formal reasoning, allowing the constructions of more accurate models. By having a more faithful model of reality, more effective predictions can be made by analysis of the model.

A visual language defines the set of all visual sentences (diagrams) which can be derived from its the grammar. We will formalize the process diagrams language's. The rest of this section defines the metamodel for process diagrams. This metamodel will be used in the next section as a basis for the grammar rules that define this visual language. The process diagrams metamodel will be given by a special kind of graphs and a related visual representation. Thus, a graph will be an abstract representation of a process diagram. We will use rules over graphs to define the grammar rules that generate all (syntactically correct) process diagrams. Then, a formal semantics for this language will be presented, assigning meaning to each process diagram.

The abstract representation of a process diagram will be given by a graph. We chose special graphs, in which nodes and edges have special labels. The basis of our approach is the definition of attributed hypergraphs [5], that are graphs in which each edge may have many source and target vertices, and in which the vertices and edges are attributed, that is, values may be assigned to them.

Definition 1 (Graph). *An (attributed hyper) graph is a tuple* $G = (V_G^G, V_D^G, E_G^G, E_{NA}^G, E_{EA}^G, (source_j^G, target_j^G)_{j \in \{G,NA,EA\}})$, *where*

- *V_G^G and V_D^G are sets, called sets of* graph vertices *and* data vertices, *respectively;*
- *E_G^G, E_{NA}^G and E_{EA}^G are sets, called sets of* graph edges, vertex *and* edge attributes, *respectively;*
- *$source_G^G : E_G^G \to V_G^{G*}$, $target_G : E_G^G \to V_G^{G*}$ are total functions that assign lists of vertices as source and target for each graph edge (V_G^{G*} denotes the set of all lists over the set V_G^G);*
- *$source_{NA}^G : E_{NA}^G \to V_G^G$, $target_{NA} : E_{NA}^G \to V_D^G$ are total functions that define source and target vertices for each vertex attribute;*
- *$source_{EA}^G : E_{EA}^G \to E_G^G$, $target_{EA}^G : E_{EA}^G \to V_D^G$ are total functions that define source and target vertices for each edge attribute.*

Graphs are related by morphisms which map vertices and edges in a compatible way, that is preserving the source and target of each edge.

Definition 2 (Graph Morphism). *Consider two graphs G_1 and G_2 $G_i = (V_G^{G_i}, V_D^{G_i}, E_G^{G_i}, E_{NA}^{G_i}, E_{EA}^{G_i}, (source_j^{G_i}, target_j^{G_i})_{j \in \{G_i, NA, EA\}})$ for $k = 1, 2$. A (partial) graphs morphism $g : G_1 \to G_2$ is a tuple $f = (f_{V_G}, f_{V_D}, f_{E_G}, f_{E_{NA}}, f_{E_{EA}})$ where $f_{V_i} : V_k^{G_1} \to V_k^{G_2}$ e $f_{E_j} : E_j^{G_1} \to E_j^{G_2}$ for $k \in \{G, D\}$ and $j \in \{G, NA, EA\}$ are (partial) functions, such that f commutes with all source and target functions, that is, for all edge e and corresponding source and target functions $f_V \circ source(e) = source \circ f_E(e)$ and $f_V \circ target(e) = target \circ f_E(e)$. If all components of a morphism are total, we say that the morphism i total.*

Process diagrams have a particular structure, and therefore we will use a typing mechanism for vertices and edges to distinguish among the different kinds of elements that appear in these diagrams. Formally, this concept of typing will be given by defining a graph in which vertices and edges represent the types of elements, called *graph of types*. Types of elements of each graph will be defined by mapping this graph to the typed graph by a morphism.

Definition 3 (Typed graph). *Let TG be a graph, called* graph of types. A graph typed over TG is a tuple $(G, type)$ where G is a graph and $type : G \to TG$ is a total graph morphism.

Process diagrams are a special class of typed graphs, where each node represents one chemical element and each edge represents a each chemical reaction. Now we define the graph of types that will be used to represent such diagrams. First, we define the set of (graph) vertices and edges (Def. 4), then the attributes (Def. 5), and finally the source and target functions (represented in Figure 2). More details about the informal semantics of each element of a process diagram can be found in [2] and [3].

Definition 4 (BioProc types). *The set graph vertices and edges of BioProc, denoted by V^{Bio} and E^{Bio}, are:*

$$V^{Bio} = \{Protein, Receptor, Ion\_Channel, Truncated\_Protein, Simple\_Molecule, \quad (1)$$

$$Unknown\_Molecule, Phenotype, Homodimer, Ion, Gene, RNA, Antisense\_RNA, \quad (2)$$

$$Complex, Compartment, And, Or, Empty\_Set, Legends\_Concentrations\} \quad (3)$$

$$E^{Bio} = \{State\_Transition, Unknown\_Transition, Known\_Transition\_Ommited, \tag{4}$$
$$Translocation, Bidirectional\_Transition, Association, Dissociation, Truncation, \tag{5}$$
$$In\_Compartment, In\_And, In\_Or, In\_Complex\} \tag{6}$$

Vertex types represent the different types of substances involved in reactions (lines (1) and (2)). Moreover, we use special types (line (3)) like *Legends_Concentrations* to represent concentrations of substances. Eight edge types (lines (4) and (5)) correspond to the different kinds of edges in the process diagram notation. There are additional edge types (line (6)). For example, edge types *In_Compartment*/*In_Complex* are used to associate an instance of type *Compartment*/*Complex* to a list of substances; an edges of kind *In_And* associates an instance of type *And* to a list of substances (the idea is that all these substances must be present to activate the inhibition or stimulation of the corresponding reaction);*In_Or* has a similar role.

Each vertex and edge has associated attributes. Originally, graphs representing process diagrams (defined in [2]) had no attributes associated to vertices and edges. However, such attributes are needed to obtain models that are suitable for analysis, and thus must be part of the abstract syntax of the language. The choice of attributes used here was based on the database of biological pathways BioModels [4].They are the necessary parameters for simulation/analysis of biological processes, as well as parameters of the textual description of the system.

Definition 5 (Bio-Attribute types). *The types of attributes of the BioType graph are:*

$$V_D^{Bio} = \{Integer, Double, String, Boolean, List[String],$$
$$List[Integer], List[pair(Integer, Integer)], Boolean\_expression\}$$

The names of attributes of vertices and edges E_{NA}^{Bio} and E_{EA}^{Bio} are defined by

$$E_{NA}^{Bio} = \{name, amount, state, units, intervals, colors\}$$
$$E_{EA}^{Bio} = \{name\_react, rate\_law, react\_parameter\_tax,$$
$$participants, prod\_parameter\_tax, condition\_trigger\}$$

Intuitively, the attribute names have the following types and meaning:

– $name : String$ - scientific name of the participant substance;
– $amount : Integer$ - quantity of substance present in the biological pathway ;
– $state : Boolean$ - substance state, indicating whether it is normal or activated;
– $units : String$ - type of unit that is used to define concentration ranges;
– $intervals : List(pair(Integer, Integer))$ - defines concentration ranges which the attribute *amount* of each substance may assume. It is a list of pairs of integers, in which the first elements represent the minimum and the second the maximum values in the range;
– $colors : List(Integer)$ - defines the color associated with each concentration interval;
– $name\_react : String$ - scientific name to identify the reaction;

- $rate\_law : Double$ - indicates the speed with which the reaction occurs. It may be a constant rate or an expression depending on parameters such as the quantity of substances participating in the reaction;
- $react\_parameter\_tax : List(Integer)$ - list of parameters that represent the stoichiometric coefficients of each reagent of the reaction. The order of elements in this list must correspond to the order of source vertices of the corresponding edge;
- $participants : List(String)$ - list of labels that define the substance as part of the reaction is as simple participant, promoter or inhibitor of the reaction;
- $prod\_parameter\_tax : List(Integer)$ - list of parameters that represent the stoichiometric coefficients of products of the reaction. The order of elements in this list must correspond to the order of target vertices of the corresponding edge.
- $condition\_trigger$ - necessary condition for the occurrence of the reaction, for example when a substance reaches a certain amount. The type is $Boolean\_expression$, when no condition is specified its value is *true*;

Now we can define the type graph that can be used as abstract syntax for process diagrams.

Definition 6 (*Bio* Type Graph). *The graph of types that represents process diagrams, is defined by:*
$Bio = (V^{Bio}, V_D^{Bio}, E^{Bio}, E_{NA}^{Bio}, E_{EA}^{Bio}, (source_j^{Bio}, target_j^{Bio})_{j \in \{G,NA,EA\}})$ *where:*

- V^{Bio}, V_D^{Bio}, E^{Bio} ,E_{NA}^{Bio}, E_{EA}^{Bio} *are defined in. 4 and 5;*
- *the source and target functions are defined in Figure 2.*

In the following we denote by $Value_D^{Bio}$ the disjoint union of the sets corresponding to all data type names of V_D^{Bio}.

Definition 7 (BioProc). *Given the graph of types Bio, a BioProc is any graph $(G, type)$ typed over Bio, with $V_D^G = Value_D^{Bio}$. Given a BioProc $(G, type)$, whenever an attbribute $attr$ of a vertex or edge x will be considered, we will write $attr(x)$ for d such that there is a vertex attrInst and $source_i^G(attrInst) = x$, $target_i^G(attrInst) = d$ and $type(attrInst) = attr$, with $i \in \{NA, EA\}$*

The graph *Bio* can be considered as a meta-model for process diagrams. Any graph that can be mapped to it, called here *BioProc*, is an abstract representation of a process diagram. Note that using a graph of types (instead of just labels for vertices and edges, like in [2]) has the advantage that structural constraints are also represented. For example, if in the graph of types there is no edge of type X between vertices of types A and B, in any instance of this graph such kind of arcs may appear connecting vertices typed with A and B. This means that a grammar that generates *BioProc* only generates syntactically correct abstract descriptions of process diagrams.

The relation between abstract and concrete syntax can be seen in Figure 1. We assigned a graphical representation to each vertex, edge and attribute used in *BioProc* graphs. In Figure 2 we present one short representation of graph type of the language. The square in dot lines represents the attributes. To illustrate in a clear way how each edge relate your source and target types, we fetch the nodes in small blocks with types that has common characteristics: *simple_sub*, *composed_sub*, *genetic_elem* and *logic* represented respectively with the triangle, square, circle and diamond shapes. Each edge has types that can be associated with a source or target functions. If we associated one shape for one of this functions all types of the block can be associated with the edge.

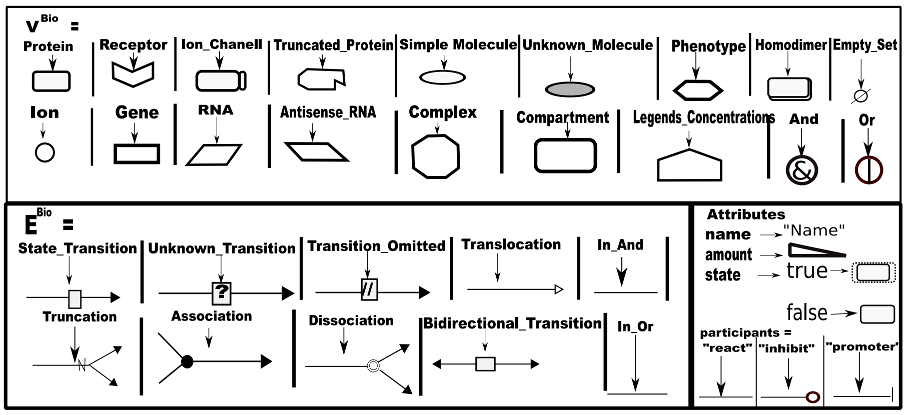


Fig. 1. Abstract and concrete syntax of nodes, edges and attributes.

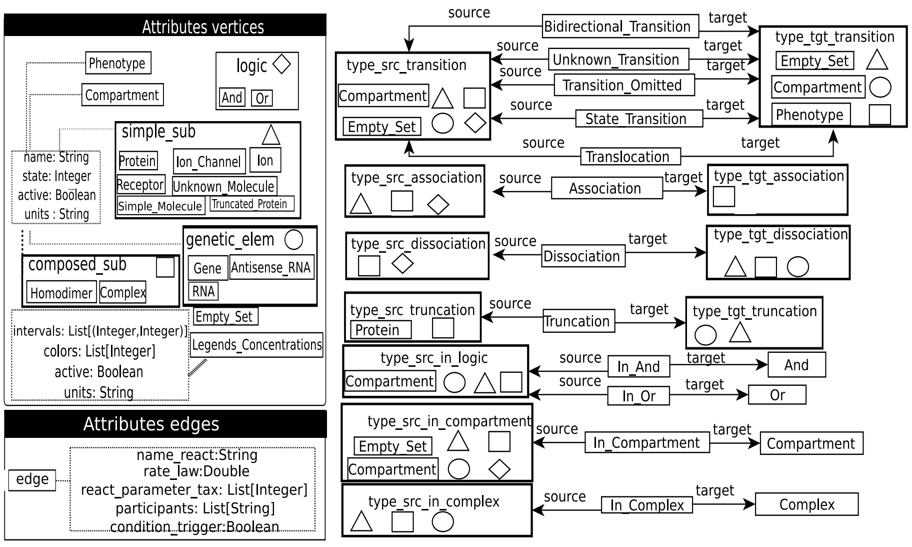
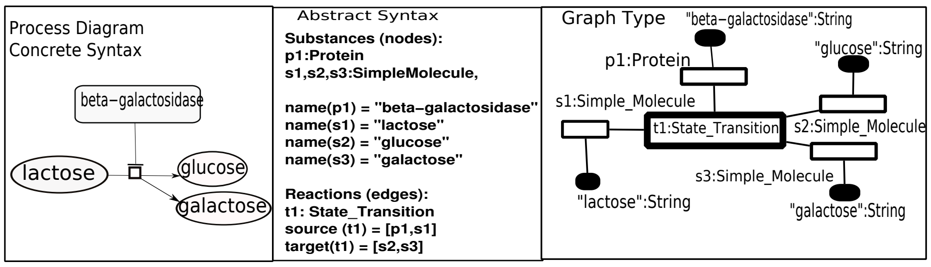


Fig. 2. Short graph type representation

To illustrate the relation between the two proposed syntaxes, we show an example of a reaction that cleaves lactose in glucose plus galactose and it is catalyzed by beta-galactosidase protein. Each graphical symbol of the substances included in the diagram is an instance of a vertex of the graph of types *Bio* ($s1, s2$ and $s3$ are instances of type *Simple_Molecule*, $p1$ is an instance of the type *Protein*). Each substance has an attribute name associated to it, which has the type String . The reactions are typed according to the graph edges of types. In the example the reaction $t1$ is of type *State_Transition*. The reaction combines a list of reactive substances by the function *source* and a list of products by the function *target*.



```
Substances (nodes):
p1:Protein
s1,s2,s3:SimpleMolecule,

name(p1) = "beta−galactosidase"
name(s1) = "lactose"
name(s2) = "glucose"
name(s3) = "galactose"

Reactions (edges):
t1: State_Transition
source (t1) = [p1,s1]
target(t1) = [s2,s3]
```

Fig. 3. Representation of Abstract and Concrete Syntax of a Process Diagram.

In the following sections we show how to use rules over BioProc graphs to describe syntax, semantics and evolution of pathways. Due to space limitations, we will not present the corresponding formal definitions, and will show just some examples to rules of each of these three kinds.

3 Constructing Process Diagrams

A production of a (string) grammar is defined as $L := R$ where L is called left hand side (LHS) and R right-hand side (RHS). A production can be applied if there is an instance of LHS in the string that is being transformed, and the effect of the application is to substitute the occurrence of the LHS by the occurrence of the RHS. Using graphs, the idea remains essentially the same, just that the notion of substitution is a bit more involved. Here we will not present the formal definitions of graph production and application, see [5] for the corresponding definitions.

The rules to define process diagrams are graph rules that describe how (syntactically) correct diagrams can be generated. Examples of these rules are depicted in Figure 4. Rule *Insert_Protein* can always be applied (the LHS is empty) and its effect is to generate a vertex representing a protein. Rule *Insert_Association* can be applied if two proteins and a complex have previously been generated, its effect is to connect them by an *Association* edge. Given a type graph, one can derive automatically the set of rules necessary to construct correct instances of this type graph [6]. If we have an associated concrete syntax, these rules can be used to automatically generate a corresponding graphical editor (see [6]).

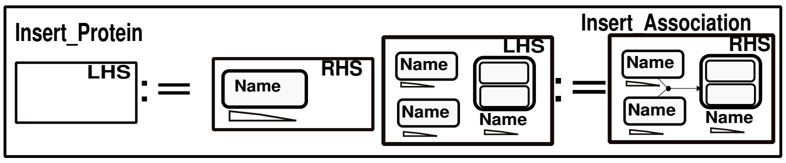


Fig. 4. Some Rules to Construct Process Diagrams

4 Semantics of Process Diagrams as Stochastic Graph Transformation Systems

A process diagram essentially represents a static structure: the structure of the pathway. The graph grammar presented in the previous section produces process diagrams in which each substance has an associated value, describing the current concentration of this substance in the system. Thus, BioProc graphs represent snapshots or states of a biological system. To model the dynamic aspects of biological pathways, it is necessary to associate an execution of simulation model to these diagrams. We propose to interpret each reaction as a rule, that changes the amount of the involved substances (reagents and products), leaving the rest of the graphical structure of the pathway intact. Of course, results of a simulation of such a rule system will be approximations with respect to the real system, since we are using discrete simulation.

The formal semantics of this underlying rule system is given by Stochastic Graph Transformations. This special kind of Graph Grammars was proposed by [7] as an extension of the original framework, in which a rate is associated to each transformation rule. Each rate represents an exponentially distributed application delay to be considered during execution. We also associate conditions to each rules, that are booleans expressions that must be true for a rule to be applied. We may use variables in the rules, as well as operations of the corresponding data types. To select a rule for application, besides finding an image of the graph of the left-hand side of the rule in the current state, we must also find an assignment of all variables appearing in its left- and right-hand sides to values, such that the conditions for the rule application become true.

Some rules of the semantics of process diagrams are shown in Figure 5. The first rule (*processAssoc*) shows the behavior of associations of substances. We start with an association of two proteins ($p1$ and $p2$) and a complex ($c1$) with respective amounts (x, y and z). The attribute *rate_law* is used as a delay of application of the rule. The values of attributes will change according to right hand side of the rule: $p1$ and $p2$ are subtracted by values of *react_parameter_tax* list $r1$, $r2$, respectively, and $c1$ is summed by *prod_parameter_tax* list value $prod1$. This rule can only be applied if the concentration of substances $p1$ and $p2$ is greater or equal to $r1$ and $r2$, respectively. The second rule (*processTrans*) is analogous, but shows the behavior when there is an inhibitor substance involved ($p1$). This reaction can only occur if $p1$ is not present, described by the equation $x = 0$ associated to this rule.

We can use Stochastic Graph Transformation Systems to build Markov Chains. Each state of graph grammar corresponds to a markovian state, where there are associated a probabilities of changing to another state and to stay in the same state. Its

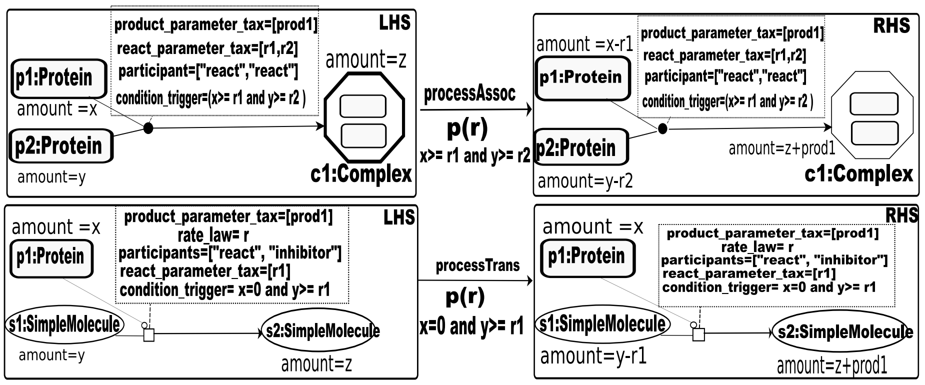


Fig. 5. Semantic Rules.

possible to analyze (via model checking) and simulate such systems using the PRISM tool [8].

5 Evolution of Process Diagrams

Models are constantly changing: new kinds of vertices/edges/attributes may show to be necessary, others may become obsolete, new graphical representations may be needed, etc. We can use the Bio Type graph as a basis for model transformation. Graph grammars have been successfully used as a tool to describe model transformations [9]. The idea is that we can use graph rules to describe at a metamodel level how models evolve. Some examples of rules that could be used to show the evolution of BioProc model are depicted in Figure 6. First, suppose that we would like to integrate two attributes into one. Rule *red_att_cols* integrates *intervals* (a list of pair structure) and *colors* (a list of values) creating a new attribute called *interval_colors*, that is a list of triples (the first two number describe the intervals and the third the color of associated to this edge). An another example could be the inclusion in the model of a boolean attribute to indicate if a protein is a enzyme or not.

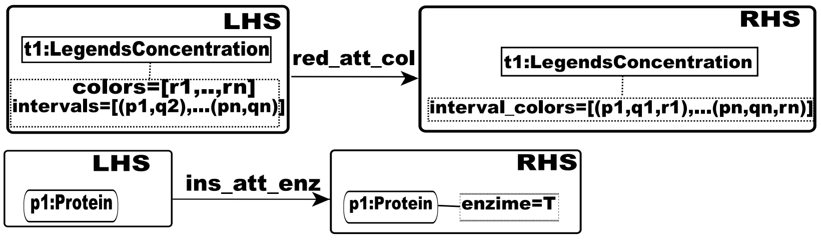


Fig. 6. Some Evolution Rules.

6 Conclusions and future work

In this paper we have developed a formal foundation for biological process diagrams, by defining their (concrete and abstract) syntax and semantics using the graph grammars formalism. This approach enables an user-friendly framework to manipulate biological pathways specifications, as well as takes advantage from the well-developed theory of graph grammars to analyze specifications [10]. The use of graph grammars to describe model evolution enables to improve the model of biological process diagrams with new properties, characteristics and relations not assumed at the time of definition of the model. This is interesting for communities like SBGN [11] where improvements are constantly being made in the descriptions of biological pathways. This paper defines the basis of our framework, that is the metamodel of process diagrams, and discussed how to use it to accomplish these three different tasks: definition of syntax, semantics and evolution. Future work includes implementation of the approach, by using and extending the existing tools, as well as validating it by realistic case studies.

.

References

1. Luciano, J.S., Stevens, R.D.: e-science and biological pathway semantics. BMC Bioinformatics **8 Suppl 3** (2007) 1–21
2. Kitano, H., Funahashi, A., Matsuoka, Y., Oda, K.: Using process diagrams for the graphical representation of biological networks. Nature Biotechnology **23**(8) (August 2005) 961–966
3. Kitano, H.: A graphical notation for biochemical networks. Biosilico **1**(5) (2003) 169–176
4. Le Novère, N., Bornstein, B., Broicher, A., Courtot, M., Donizelli, M., Dharuri, H., Li, L., Sauro, H., Schilstra, M., Shapiro, B., Snoep, J.L., Hucka, M.: Biomodels database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. Nucleic Acids Res **34**(Database issue) (January 2006)
5. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series). Springer (March 2006)
6. Bardohl, R.: GenGED - Visual Definition of Visual Languages based on Algebraic Graph Transformation. PhD thesis, Technische Universität Berlin (1999)
7. Heckel, R., Lajios, G., Menge, S.: Stochastic graph transformation systems. Fundam. Inf. **74**(1) (2006) 63–84
8. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: Probabilistic symbolic model checker. In Field, T., Harrison, P., Bradley, J., Harder, U., eds.: Proc. 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02). Volume 2324 of LNCS., Springer (2002) 200–204
9. Ehrig, H., Ehrig, K.: Overview of formal concepts for model transformations based on typed attributed graph transformation. Electr. Notes Theor. Comput. Sci. **152** (2006) 3–22
10. Rozenberg, G., ed.: Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations. In Rozenberg, G., ed.: Handbook of Graph Grammars, World Scientific (1997)
11. Novere, N.L., Moodie, S., Sorokin, A., Hucka, M., Schreiber, F., Demir, E., Mi, H., Matsuoka, Y., Wegner, K., Kitano, H.: Systems biology graphical notation: Process diagram level 1. Available from Nature Precedings : http://hdl.handle.net/10101/npre.2008.2320.1 (2008)