Proceedings of the 2nd International Workshop on Biological Processes & Petri Nets (BioPPN2011)
online: http://ceur-ws.org/Vol-724 pp.21-35

GReg : a domain specific language for the modeling of genetic regulatory mechanisms

Nicolas Sedlmajer, Didier Buchs, Steve Hostettler,
Alban Linard, Edmundo Lopez, and Alexis Marechal

Université de Genève, 7 route de Drize, 1227 Carouge, Switzerland

Abstract. Chemical and biological systems have similarities with *IT-systems* as they can be observed as sequences of events. Most available tools propose simulation frameworks to explore biological pathways (*i.e.,* sequences of events). Simulation only explores a few of the most probable pathways in the system. On the contrary, techniques such as model checking, coming from IT-systems analysis, explore all the possible behaviors of the modeled systems, thus helping to identify interesting pathways. A main drawback from most model checking tools in the life sciences domain is that they take as input a language designed for computer scientists, that is not easily understood by non-expert users. We propose in this article an approach based on Domain Specific Languages. It provides a comprehensible language to describe the system while allowing the use of complex and powerful underlying model checking techniques.

1 Introduction

Because of their stochastic and combinatorial nature, many biological systems such as cellular and supra-cellular interactions are very hard to investigate. Current practice is mainly limited to the use of *in vivo* and *in vitro* experiments. Investigation through formal models of biological systems is currently a rather restricted research field, unlike what has been done in other natural sciences such as chemistry and physics. There is clearly an emerging field of research where future experiments can be partially performed *in silico*, *i.e.,* by means of techniques from computer science. One of the main approaches of biological modeling is the so-called regulatory networks [17,5]. The main idea of biological modeling according to the regulatory network approach is to model interbiological reactions through a set of interdependent biological rules. This can be seen as a set of discrete modules having strong interconnections. The occurrence of interesting events in the biological system can be represented as logical properties expressed on the state of these modules. This is very similar to the kind of properties computer scientists validate on hardware and software systems (deadlocks, error states,...).

Among the tools available in this domain, the main analysis approach for regulatory networks is *simulation*. Simulation is generating and analyzing a limited sample of possible system behaviors. This technique is not convenient when the

main purpose of the research is to look for rare or abnormal behaviors (*e.g.,* cancer). The main approach in this case is to use *model checking* instead of simulation. Model checking consists in generating and analyzing the complete set of possible states of the system. Naturally, this technique suffers from the drawback of the enormous number of possible states of biological systems.

It is interesting to note that this problem is well-known to the model checking community in computer science, where it is called the *state space explosion* [18]. There is a parallel between cellular interactions and software systems in that the state space explosion is mainly due to their concurrent nature. Therefore, we can apply techniques that have been developed for the model checking of hardware and software systems to biological interactions. Approaches based on a symbolic encoding of the state space are particularly well-suited for this [4,9].

In this paper we show a work in progress in our group. We present Gene Regulation Language (GReg), our first attempt to build a framework for modeling and analyzing biological systems based on formal modeling and reasoning. Advanced techniques for defining Domain Specific Languages, giving their semantics and analyzing them using symbolic model checking are presented. First, Section 2 describes precisely the biological domain considered by GReg, then Section 3 outlines the state of the current research in the field and Section 4 describes in detail the creation and usage of Domain Specific Languages. The following two chapters describe GReg itself. Section 5 describes the language designed for expressing biological mechanisms and the corresponding queries, and Section 6 provides a simple example taken from the literature. Finally, Section 7 concludes the article and discusses the future research perspectives in this area.

2 Chemical and biological models covered by GReg

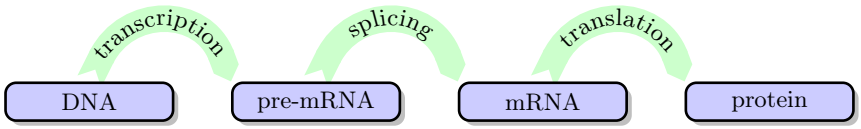

Fig. 1: DNA to protein process

The purpose of GReg is to describe genetic regulatory mechanisms controlling the DNA to protein process (Figure 1). This process comprises three steps: transcription, splicing and translation. The regulation of each step can be modeled using standard chemical reactions, presented in Section 2.1. For the transcription initiation we use the genetic regulatory mechanism model, presented in Section 2.2. Finally we define a cell network in Section 2.3. We separate these three domain models to clearly distinguish chemical from biological concepts.

We use the same definition of molecule level as presented in [17], if a molecule has n distinct actions, we define $(n + 1)$ levels. This allows us to use a discrete formalism to efficiently model the gene expression [17], which is in fact the

concentration of the gene products. The lowest level is the lowest transcription rate of a gene.

2.1 Chemical compartment model

A chemical compartment $\kappa = \langle M_\kappa, R \rangle \in K$, is composed of a non-empty set of molecules ($\varnothing \neq M_\kappa \subseteq Molecules$) and a set of reactions ($R \subseteq Reactions$). Two compartments may be separated by a membrane (*i.e.,* selective barrier), thus allowing molecule transfer between them.

A chemical reaction $\rho = \langle Re, Pr, Ca, k, type_\rho \rangle \in R$, where $Re, Pr, Ca \subseteq \mathcal{M}(M_\kappa) \times K$, the reactants ($Re$), products ($Pr$) and catalysts ($Ca$) are defined as the association of a multiset of one molecule ($\mathcal{M}(M_\kappa)$) with a given compartment ($\kappa \in K$).

Catalysts (Ca) have the particularity to appear as reactants and products in a chemical reaction. We have chosen to define the catalysts in a distinct set, therefore no catalyst can appear as reactant or product:
$\forall m \in M_\kappa, m \in Ca \implies m \notin Re \wedge m \notin Pr$.

A reaction may be either irreversible or reversible, $type_\rho \in \{irr, rev\}$. In reversible reactions an equilibrium constant (k) is defined with the ratio of both direction rates.

2.2 Genetic regulatory mechanism model

A genetic regulatory mechanism $\mu = \langle \Gamma_\mu, C \rangle$, is composed of a non-empty set of genes (Γ_μ). Genes are organized into one or more chromosomes (C). A genetic regulatory mechanism is contained in a chemical compartment, this specification will be presented in Section 2.3.

A chromosome $c = \lambda_1, \ldots \lambda_n \in C$ is a sequence of loci. A gene may have different version (*i.e.,* alleles) at a given chromosome location (*i.e.,* locus). And a locus $\lambda = \langle \Gamma_\lambda \rangle$ defines a non-empty set of genes ($\varnothing \neq \Gamma_\lambda \subseteq \Gamma_\mu$) that are located at a given locus. Then the set of all possible chromosomes in a mechanism is the Cartesian product of the set of genes at each locus:
$Chromosomes = \Gamma_{\lambda_1} \times \cdots \times \Gamma_{\lambda_n}$.

A gene $\gamma = \langle M_\gamma, \Sigma \rangle \in \Gamma_\mu$ is a portion of DNA that codes for at least one molecule ($\varnothing \neq M_\gamma \subseteq M_\kappa$), and may contain some regulation sites ($\Sigma \subseteq Sites$).

A regulation site $\sigma = \langle M_\sigma, type_\sigma \rangle \in \Sigma$ defines the non-empty set of regulatory molecules ($\varnothing \neq M_\sigma \subseteq M_\kappa$) associated to the regulation site σ of a given gene. Note that when using anti-termination sites, genes order matters, therefore chromosomes must be defined.

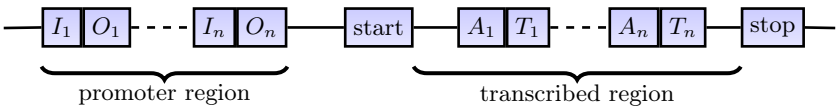

Fig. 2: Idealized gene structure

Our idealized gene structure (Figure 2) is composed of two regions: promoter and transcribed. Note that the exact position in the gene of each regulatory site is not specified, *i.e.,* we are mainly interested in its regulatory role. The type of a regulation site is $type_\sigma \in \{\mathbf{I},\mathbf{O},\mathbf{A},\mathbf{T}\}$.

Initiation (I) the portion of DNA where bound activators increase the rate of the transcription process. It is located in the promoter region;

Operator (O) the portion of DNA where bound repressors block the transcription process. It is located in the promoter region;

Anti-termination (A) the portion of DNA where activators continue the transcription process. It is located in the transcribed region. These sites may also allow the transcription of the next gene;

Termination (T) (also called attenuator) the portion of DNA where repressors stop the transcription process. It is located in the transcribed region; thus, it produces a reduced RNA.

2.3 Cell network

The cell network model is designed to model the interactions of different cells with their environment and also with their inner components (*i.e.,* organelles). This model authorizes the construction of currently not observed cells, *e.g.,* prokaryote with nucleus, eukaryote with multiple nuclei, *etc.* The validity of the specification is delegated to the user (*i.e.,* domain expert).

This model defines three chemical compartments, therefore they inherit both sets M_κ and R.

Organelle, $\omega = \langle M_\kappa, R, \mu \rangle$ is the lowest compartment in the compartment hierarchy. An organelle may contain a mechanism (μ), *e.g.,* nucleus, mitochondrial DNA, *etc.*

Cell, $\phi = \langle M_\kappa, R, \mu, \Omega \rangle$ contains a possibly empty set of organelles (Ω). The model of a prokaryote cell would define a mechanism (μ), by cons an eukaryote cell would define instead an organelle with a mechanism (*i.e.,* nucleus).

Network, $\nu = \langle M_\kappa, R, \Phi \rangle$ represents the environment and contains one or more cells (Φ).

3 Related work

In this section we compare three well-known tools with our approach. We first define a few criteria such as the kind of analysis, the supported formalism and the supported exchange format. Table 1 presents a summary of the resulting comparison.

Domain language To be productive, the syntax of the input language should be as close as possible to the actual domain of the user. This input language can be textual (like in tools that use Systems Biology Markup Language (SBML) [8]) or graphical (like Systems Biology Graphical Notation (SBGN) [11]).

Simulation & Model Checking Although there are many tools adapted to biological process design and simulation, only a few of them allow exhaustive exploration of the state space. While simulation is very useful during model elaboration, an exhaustive search may help to discover pathological cases that would have never been explored by simulation.

Discrete & continuous Continuous models are closer to the real biological systems than discrete models, but unlike the latter they are not adapted for model checking techniques. Discrete formalisms allow a complete exploration of the state space while preserving the qualitative properties of the system, as mentioned in [17].

Exchange format The supported interchange format is an important feature as it allows us to bridge the gap between different tools and therefore enables the user to use the most adapted tool to hand. SBML is a common interchange format based on XML. It is used to describe biochemical reactions, gene regulation and many other topics.

Cell Illustrator [16] is an example of a commercial simulation tool for continuous and discrete domains. The graphical formalism is based on PN, called Hybrid Functional Petri Nets with extensions (HFPNe), which adds the notions of continuous and generic processes and quantities [12]. The XML-based exchange file format used in Cell Illustrator is called CSML.

Gene Interaction Network simulation a.k.a. GinSim [14] is a tool for the modeling and simulation of genetic regulatory networks. It models genetic regulatory networks based on a discrete formalism [6,13]. These models are stored using the XML-based format *ginml*. The simulation computes a state transition graph representing the dynamical behavior network. GINsim uses a graphical Domain Specific Language (DSL) called Logical Regulatory Graph (LRG) [5]. Models in LRG are graphs, where nodes are regulatory components (*i.e.,* molecules and genes) and arcs are regulatory interactions (*i.e.,* activation and repression) between the nodes.

Cytoscape [7,15] is an open source software platform for visualizing complex networks and integrating these with any type of attribute data. Cytoscape supports many file formats including PSI-MI and SBML. It has the advantage of adding features through a plug-in system. Many plug-ins are available for various domains such as biology, bioinformatics, social network analysis and the semantic web. Over 100 plug-ins are listed on the official website.

4 DSL approach

Model checking involves verifying whether a property holds on the whole set of possible states of a given model. To generate this complete state space, the model must be expressed in a formal language intended for this operation, like Petri Nets (PNs). This makes the model checking approach impractical for people who do not master these formal languages. We propose using DSLs for this purpose.

Tool	Cell Illustrator	GINsim	Cytoscape	GReg
Domain language	✓	✓	✓	✓
Simulation	✓	×	✓	×
State space	×	✓	×	✓
Model checking	×	×	×	✓
Discrete	✓	✓	✓	✓
Continuous	✓	×	✓	×
Exchange format	CSML	GINML	SBML,...	GReg

Table 1: Tool comparison table.

A DSL is a programming or specification language tailored for a given domain; it presents a reduced set of instructions closely related to this domain. Using a DSL has two main objectives. First, learning the language should be easy for someone with enough knowledge about the domain, even if this person does not have previous knowledge of other languages. Second, the number of errors made by a novice user should be drastically reduced as the expressivity of the language is reduced to the minimum.

The DSL semantics are defined by transformation into a target language, which is a formal language where complex operations (like model checking) can be performed. Usually, the scope of the target language is broader than the scope of the DSL. This allows using the same target language and its associated tools for different DSLs. Moreover, while creating a new DSL, it is often possible to use an already-existing language as a target, thus facilitating the language creation process. The results obtained in the target language are translated again into the DSL and returned to the user. This process is described in Figure 3. After the creation of the initial model, all the following steps must be fully automatic, to hide the underlying complexity from the end user.

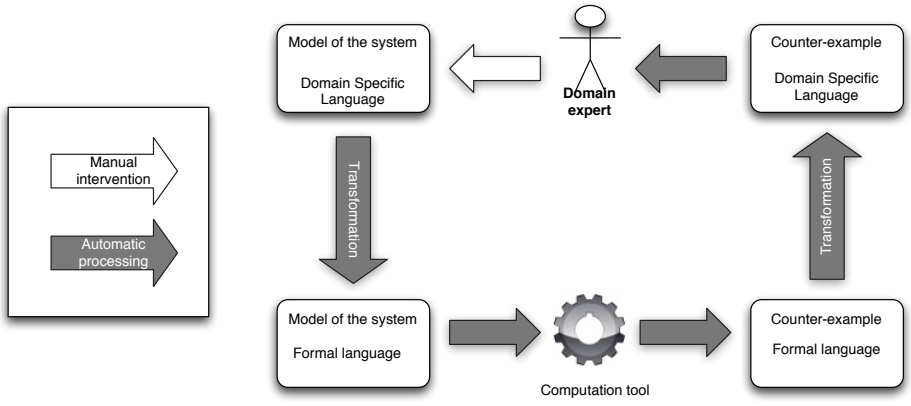

Fig. 3: DSL computational process

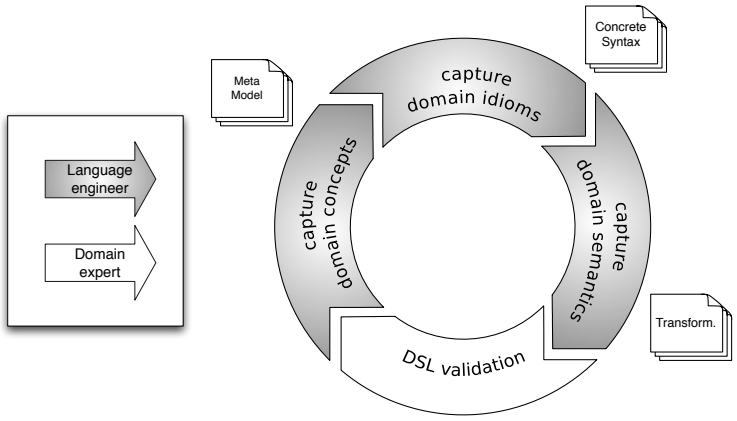

Fig. 4: DSL creation process

The person in charge of the DSL creation is a language engineer. This person should obviously have a certain knowledge about the language creation process, but he should also master the target platform language, in order to define efficient and correct transformations. Furthermore, he should be in contact with at least one domain expert, in order to settle the requirements and verify the correctness and completeness of the language created. The creation of a DSL follows a set of specific steps. First, the language engineer must identify the abstract concepts of the domain. These concepts include the basic elements of the domain, the interactions between these elements and the precise boundaries of the domain considered. Based on these concepts, the language engineer must define a set of expressions used to create a specific model in the domain, *i.e.,* a concrete syntax. Finally, the language engineer must define the semantics of the language created, usually by transformation to an existing platform. The whole process must be validated by one or more domain experts. Domain experts must validate the three steps of the DSL creation process: the domain must have been correctly defined, the expressions of the concrete syntax must be close to the already existing languages in the domain, and the execution must return the expected results. This creation/validation process often leads to an iterative development of the language. We show this entire process in Figure 4.

There exist various DSLs tailored for the biological processes, *e.g.,* SBML [8] and SBGN [11]. These two well-known languages cover a wide range of systems, mainly in the bio-chemical domain. GReg, instead, focuses on a more specific domain, which is genetic regulatory mechanisms. This domain has been described in Section 2.

While creating GReg, we used the Eclipse Modeling Project (EMP)[1] approach. We first created a metamodel of the domain using the Eclipse Modeling Framework (EMF), and we defined a concrete syntax with XText. XText provides a set of tools to create an editor for a given language, with some user-friendly features such as syntax highlighting, on the fly syntax checking (see Figure 5) and auto-completion. As a target platform we chose AlPiNA. AlPiNA[3]

is a model checking tool for Algebraic Petri Nets (APNs). It aims to perform efficient model checking on models with extremely large state spaces, using Decision Diagrams (DDs) to tackle the state explosion problem. AlPiNA's input languages were also defined using the EMP approach. This allowed us to use Atlas Transformation Language (ATL) transformations, which is a tool dedicated to define model to model transformations. GReg is thus fully integrated in the Eclipse/EMP framework.

The modular structure of GReg's definition would allow us to replace the target domain while keeping exactly the same language. If needed, we could, for example, define a transformation to SBML using a model to text transformation tool like XPand.

5 GReg : Gene Regulation Language

GReg is a Domain Specific Language designed to describe genetic regulatory mechanisms. We built it in order to illustrate the DSL approach, and the benefits it provides to research in the life sciences domain. Throughout this section, we introduce the GReg language using an excerpt of the lac operon model [10]. We also introduce the GReg Query Language (GQL) language, used to specify the queries to be executed in the model specified in GReg.

We first show how to describe a regulation mechanism. Listing 1 shows the overall structure of a GReg mechanism specification. The mechanism is named (`lac_operon`). It specifies the **molecules** occurring in the mechanism, and the chemical **reactions** between these molecules. The GReg description also specifies the **genes** with their properties and organization into **chromosomes**.

```
mechanism lac_operon is
  molecules
  -- declaration of molecules
  reactions
  -- declaration of chemical reactions
  chromosomes
  -- declaration of chromosomes
  gene -- declaration of a gene
  -- declaration of other genes
end lac_operon
```

Listing 1: GReg mechanism specification

The **molecules** section of a GReg description specifies the molecules occurring in the mechanism. For instance, in Listing 2, the `lac_operon` mechanism uses molecules `lactose`, `allolactose`, `lacI`, `lacZ`, *etc.*

```
mechanism lac_operon is
  molecules
    lactose, allolactose,
    lacI, lacZ, lacY, lacA,
    cAMP, CAP
  ...
end lac_operon
```

Listing 2: Molecules declaration

Molecules are only described by their names, as it is the only information relevant in our language. The DSL approach emphasizes specification of

only the required information for the particular domain. No molecules other than the ones described here can be used in the mechanism. This constraint is useful for the user creating a GReg specification: spelling errors in molecule names are detected, see Figure 5.

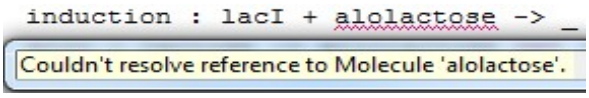

Fig. 5: Example of a spelling error in a molecule name (`allolactose`).

After the molecules declaration, a GReg description specifies the chemical reactions that take part in the mechanism. Listing 3 presents the **reactions** part of the mechanism.

```
mechanism lac_operon is
  ...
  reactions
    induction : lacI + allolactose  →  _
    allo      : lactose  →  allolactose cat lacZ
  ...
end lac_operon
```

Listing 3: Reactions declaration

In GReg, each reaction has a name, for instance `induction`. As usual in chemical notations, a reaction is a relation among (weighted) molecules. The molecule weight is usually called the stoichiometric coefficient. By default, stoichiometric coefficients are valued `1`.

A reaction can be either irreversible ($\rightarrow$) or reversible ($\leftrightarrow$). In our example, the reaction `induction` is a degradation of `lacI` and `allolactose`. A degradation is an irreversible reaction where products are not interesting, thus the reactants are simply removed from the system.

If needed, each direction of the reaction can be given a reaction rate (*i.e.,* probability). Each reaction can also have catalysts specified using the keyword **cat**. For instance `allo` is a reaction catalyzed by `lacZ`.

Listing 4 presents the **gene**s specification. For instance, `rep` is a minimal gene (*i.e.,* not regulated). A minimal gene defines at least the molecules it **codes**. If they are relevant, regulation sites are also specified in a section with the **sites** keyword. The `lac` gene defines a regulated gene with two regulation sites **I** and **O** , together with the molecule acting on them. Note that GReg also allows us to define several regulation sites for **I**, **O**, **A**, **T**.

```
mechanism lac_operon is
  ...
  gene rep
    codes lacI
  end rep
  gene lac
    codes lacZ, lacY, lacA
    sites
      I : cAMP and CAP = 1
      O : lacI @ 2
  end lac
end lac_operon
```

Listing 4: Genes declaration

As molecules may be present at different levels, the @ keyword allows the specification of the required molecules levels acting at a regulation site. By default, molecule levels are valued 1. As several molecules can act on one regulation site, GReg allows to combine molecules with Boolean operators for a regulation site. There are two operators defined : **and** and **or**. For instance the **I** site of `lac` gene specifies that `cAMP` **and** `CAP` are required to activate this site. The = keyword allows to specify the target level attributed to the gene once this site is active.

Note that for A sites, it is also possible to specify the next gene target level. As the role of a T site is to interrupt the transcription process, we allow the specification of the reduced set of produced molecules when these sites are active.

The **chromosomes** section is used to specify one or more chromosomes. A chromosome defines the sequence of loci. A locus is defined between two braces. Note that genes'order in each locus does not matter. This section is mandatory when taking into account A sites.

```
mechanism lac_operon is
  ...
  chromosomes
    c : {rep}, {lac,lac'}
  ...
end lac_operon
```

Listing 5: Chromosomes declaration

Listing 6 shows an example of GQL specification. The **use** keyword imports the `lac_operon` mechanism from another file, allowing us to reference the molecules declared in `lac_operon` mechanism from a query specification. A GQL file specifies the **levels** and the **queries**.

The **levels** section is used to define the combination of levels. A combination of levels is a partial or total definition of molecule levels, while unspecified molecules may match any possible value. For instance `l1` specifies only the level of `lacZ` among all molecules defined in `lac_operon`. The **exists** query returns true if predefined level exists. The **paths** query is used to retrieve all paths from the state space matching the sequence of predefined levels. The query `b` returns the path where `l2` is a direct successor of `l1`. But query `c` does not require that `l2` is a direct successor of `l1`.

```
use "lac_operon.greg"
levels
  l1 : lacZ = 1
  l2 : lacZ = 0, lacI = 2
queries
  bool a  : exists l1
  paths b : paths l1, l2
  paths c : paths l1 .. l2
```

Listing 6: GQL queries specification

Listing 7 shows an example of a GReg configuration specification given outside the mechanism, usually in a separate file. This allows to easily repeat experiments for the same mechanism with several initial quantities. The first section starts with the **initially** keyword and defines the genes or molecules initial levels. The second section starts with the **execute**

```
use "lac_operon.greg"
use "lac_operon.gql"
initially lac_operon has
  lactose = 1
  lacI = 1
execute
  if a then (b and c)
```

Listing 7: GReg specification

keyword and is used to specify which queries will be executed by the model checker.

6 Example

We present a simple example of a genetic regulatory mechanism taken from [17] with three genes. Gene Y is activated by the product of gene X. Genes X and Z are repressed by the products of genes Z and Y respectively. The products of genes X, Y and Z are molecules x, y and z respectively. Graphical (LRG) and textual (GReg) models derived from this example are given at Figure 6 and Listing 8 respectively.

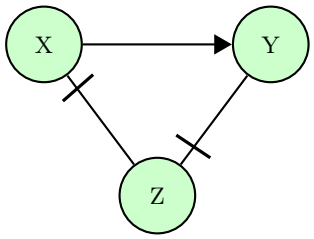

Fig. 6: LRG model of example

```
mechanism example is
  molecules x, y, z
  gene X codes x
    sites O : z
  end X
  gene Y codes y
    sites I : x
  end Y
  gene Z codes z
    sites O : y
  end Z
end example
```

Listing 8: GReg model of Figure 6

GReg models are transformed into APN models usable by AlPiNA. Note that the obtained APNs can always be unfolded into Place/Transition Petri Nets (P/Ts). We propose two different transformations:

- the first transformation produces a P/T shown in Figure 7, called Multi-level Regulatory Petri net (MRPN) in [5] ;
- the second transformation produces an APN shown in Figure 8, which is the folding of the corresponding MRPN and thus more compact.

From theses models AlPiNA is able to compute the state space and to identify all deadlock without requiring any additional input from the user.

A *deadlock* is a situation where no more events can occur in the system. Strictly speaking, in real biological systems there are no deadlocks but *livelocks*, a situation where events still occur but without changing the state of the system. The states where such situations occur are usually called stable states or attractors.

But we can also search for specific properties of the biological system. As previously mentioned, the state space might contain too many states to be used as it is. Therefore we propose a way to extract portions of state space (*e.g.*, subsets

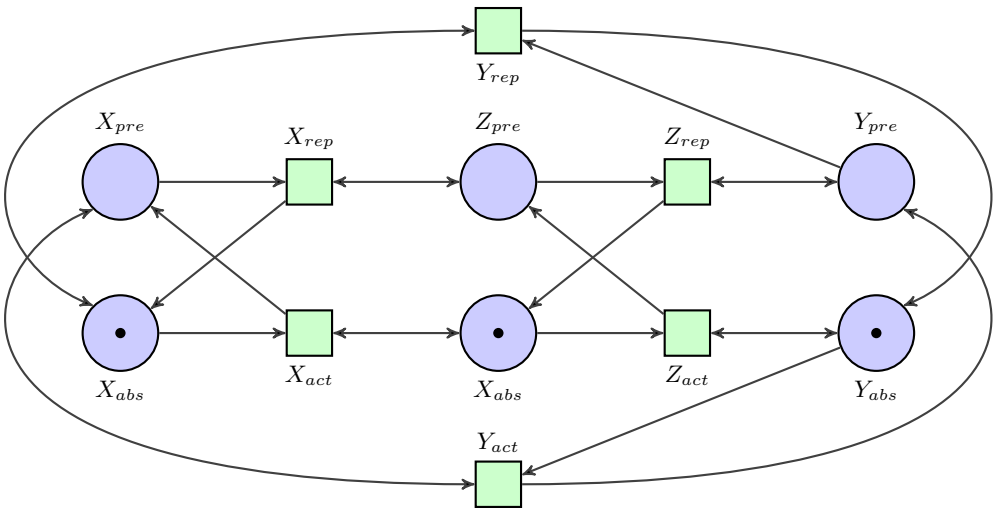

Fig. 7: PN model of example in Figure 6 and Listing 8

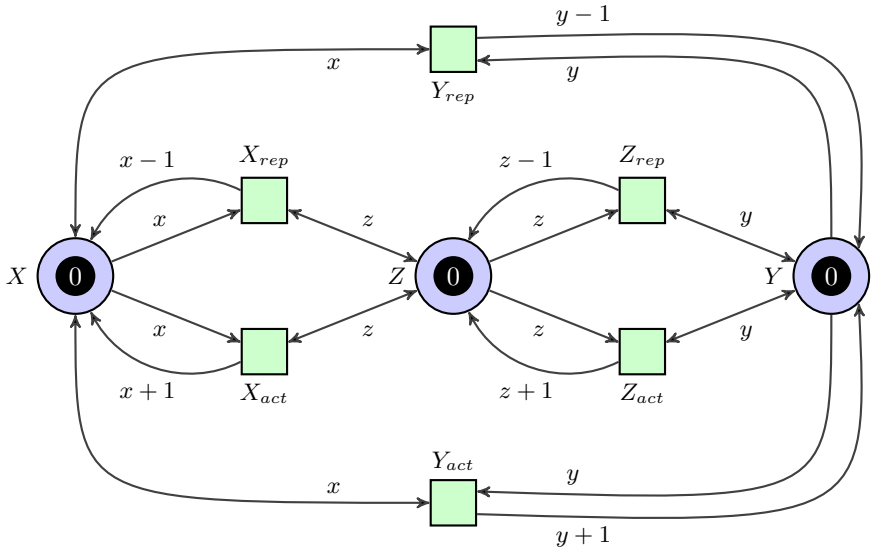

Fig. 8: APN model of example in Figure 6 and Listing 8

of states, paths, cycles, ...) through GQL queries. In Listing 9, we have defined one level (`l1`) and one query (**at**).

```
use "example.greg"
levels
  l1 : x = 1, y = 1
queries
  states s1 : at l1
```

Listing 9: GReg query example

This query returns the subset (`s1`) of states from the state space where the level of `x` and `y` is equal to one. To compute the set of states `s1`, the query is transformed into an AlPiNA property, shown in Listing 10.

```
s1 : exists($x in x, $y in y, $z in z :
 (($x equals suc(zero)) and ($y equals suc(zero))) = false
```

Listing 10: AlPiNA property expression of query in Listing 9

The example in Figure 6 and Listing 8 has eight states reachable from the initial marking, where all molecules are initially at level zero (`x,y,z`) = (0,0,0), see Figure 9. Model checking of the example PN finds two stable states: (1,1,0) and (0,0,1) and returns for `s1` the two states: (1,1,0) and (1,1,1).

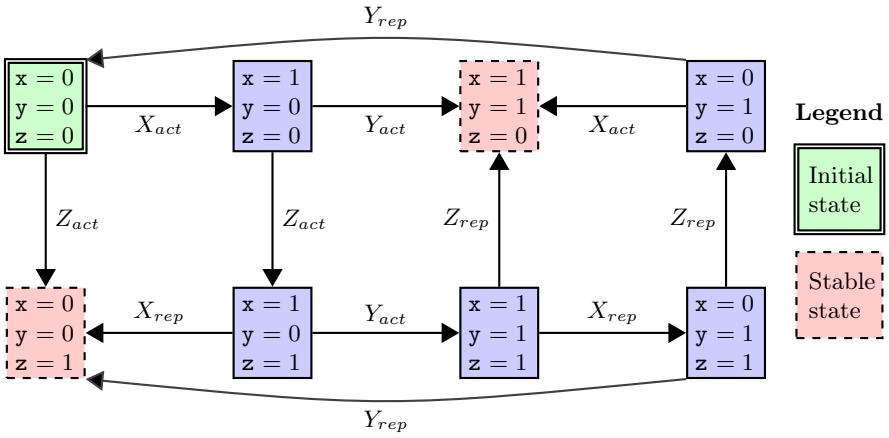

Fig. 9: State space of example in Figure 6 and Listing 8

7 Conclusion & future work

This paper introduces Gene Regulation Language (GReg), a language dedicated to the modeling of regulatory mechanisms. We explain the need to explore completely the sets of possible behaviors of a given model in order to detect rare events. GReg includes a query language used to express the properties of such events.

From a technical point of view, we explain that the techniques proposed are based on general principles borrowed from software modeling and verification. These techniques include the use of a dedicated DSL defined with a metamodeling approach and the translation of this language into a formal verification platform called AlPiNA.

The languages and transformations shown in this article have been implemented and tested on several toy examples. We also asked biologists to assess the expressivity and usability of the language. Although the first feedback seems promising, there is much room for improvement. We foresee three main axes of future development: improving the expressivity of the modeling and query languages, assessing the usability of the approach and exploring the mitigation of the state space explosion.

Extending the expressivity of GReg Concepts such as time and probabilities play an important role in biology and are therefore good candidates for a language extension. As the current underlying formalism, APNs, does not support these notions, such extension would require changing the target platform. Good examples of target formalisms are timed Petri nets and stochastic Petri nets. As mentioned in 4, the techniques used to create GReg allow changing the target language without changing the language itself. Note that we do not plan to add continuous concepts, used in languages such as HFPNe.

Improving the usability of our tool Textual domain specific languages constitute a first step towards democratization of formal methods. Although highly efficient, textual languages are usually not as intuitive as graphical languages. On the other hand, graphical domain specific languages are especially good in the early phase of the modeling as well as for documentation, but they are often less practical when the model grows. The tools in EMP that were used to create GReg allow us to define a graphical version of the same language, thus keeping the best of both worlds.

Another way to ease the modeling phase is to allow import/export of models from/to other formalisms and standards such as SBML and to integrate it with *Cytoscape* through its plug-in mechanism.

Mitigating the state space explosion So far, we have done little experimentation in this area for biological processes. Nevertheless, we conducted several studies on usual IT protocols and software models that show that AlPiNA can handle huge state spaces[2]. This suggests promising results in the regulatory mechanisms domain.

The development of GReg is a work in progress, we would like to set up more collaborations with biologists interested in exploiting formal techniques from computer science to discover rare events. We think that we can make, in the near future, a useful contribution to life sciences based on advanced techniques borrowed from computer science.

References

1. Eclipse Modeling Project. `http://www.eclipse.org/modeling/`.
2. D. Buchs, S. Hostettler, A. Marechal, and M. Risoldi. AlPiNA: A Symbolic Model Checker. In *Petri Nets'2010: Applications and Theory of Petri Nets, Braga, Portugal*, volume 6128 of *Lecture Notes in Computer Science*, pages 287–296, 2010.
3. D. Buchs, S. Hostettler, A. Marechal, and M. Risoldi. AlPiNA: An Algebraic Petri Net Analyzer. In *TACAS'2010: Tools and Algorithms for the Construction and Analysis of Systems, Paphos, Cyprus*, volume 6015 of *Lecture Notes in Computer Science*, pages 349–352, 2010.
4. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation*, 98, 1992.
5. C. Chaouiya, H. Klaudel, and F. Pommereau. A Modular, Qualitative Modeling of Regulatory Networks Using Petri Nets. In *Modeling in Systems Biology*, volume 16 of *Computational Biology*, pages 253–279. Springer London, 2011.
6. C. Chaouiya, E. Remy, B. Mossé, and D. Thieffry. Qualitative Analysis of Regulatory Graphs: A Computational Tool Based on a Discrete Formal Framework. In *Positive Systems*, volume 294 of *Lecture Notes in Control and Information Sciences*, pages 830–832. 2003.
7. Cytoscape Consortium and Funding Agencies. `http://www.cytoscape.org/`.
8. M. Hucka, F. Bergmann, S. Hoops, S. Keating, S. Sahle, J. Schaff, L. Smith, and B. Wilkinson. The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core, 2010. Available from Nature Precedings `http://dx.doi.org/10.1038/npre.2010.4959.1`.
9. J-M. Couvreur and E. Encrenaz and E. Paviot-Adet and D. Poitrenaud and P-A. Wacrenier. Data Decision Diagrams for Petri Net Analysis. In *ATPN'02: International Conference on Application and Theory of Petri Nets*, volume 2360 of *Lecture Notes in Computer Science*, pages 101–120, 2002.
10. F. Jacob and J. Monod. Genetic Regulatory Mechanisms in the Synthesis of Proteins. *Journal of molecular biology*, 3:318–356, 1961.
11. S. Moodie, N. Le Novere, E. Demir, H. Mi, and A. Villeger. Systems Biology Graphical Notation: Process Description language Level 1, 2011. Available from Nature Precedings `http://dx.doi.org/10.1038/npre.2011.3721.4`.
12. M. Nagasaki, A. Doi, H. Matsuno, and S. Miyano. A Versatile Petri Net Based Architecture for Modeling and Simulation of Complex Biological Processes. *Genome Informatics*, 15(1):180–197, 2004.
13. A. Naldi, D. Berenguier, A. Fauré, F. Lopez, D. Thieffry, and C. Chaouiya. Logical Modelling of Regulatory Networks with GINsim 2.3. *Biosystems*, 97(2), 2009.
14. A. Naldi, C. Chaouiya, and D. Thieffry. GINsim. `http://gin.univ-mrs.fr/`.
15. P. Shannon, A. Markiel, O. Ozier, N.S. Baliga, J.T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: a Software Environment for Integrated Models of Biomolecular Interaction Networks. *Genome Research*, 13, 2003.
16. The University of Tokyo. CellIllustrator. `http://www.cellillustrator.com/`.
17. R. Thomas. Regulatory Networks seen as Asynchronous Automata: a Logical Description. *Journal of Theoretical Biology*, 153:1–23, 1991.
18. A. Valmari. The State Explosion Problem. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, pages 429–528, 1998.