INRIA Genostar Bioinformatics solutions

IOGMA® 3.11

Genetic Network Analyzer Tutorial

www.genostar.com

Genetic Network Analyzer Tutorial

Contact

Genostar
Miniparc Chartreuse
60, rue Lavoisier
38330 Montbonnot

http://www.genostar.com

info@genostar.com

Table of Contents

List of Figures

Part 1. Overview of GNA

1.1. What is GNA?

GNA (Genetic Network Analyzer) is a computer tool for the modeling, simulation and analysis of genetic regulatory networks. The aim of GNA is to assist biologists and bioinformaticians in constructing a model of a regulatory network using knowledge about regulatory interactions in combination with gene expression data.

GNA consists of a simulator of qualitative models of genetic regulatory networks in the form of piecewise-linear differential equations, a class of models originally introduced by Leon Glass and Stuart Kauffman. The simulator has been implemented in Java and has been applied to the analysis of various regulatory systems, including the networks controlling the initiation of sporulation in *B. subtilis* and carbon starvation in *E. coli*. The simulator can be used in combination with standard model-checking tools in order to analyze important model properties.

GNA has been developped at INRIA Grenoble-Rhône-Alpes, initially by Hidde de Jong and Michel Page. The following persons have contributed to the current and previous versions of the software: Grégory Batt (now at INRIA Rocquencourt), Bruno Besson (now at Genostar SA), Estelle Dumas, Céline Hernandez (now at the Swiss Institute for Bioinformatics, Geneva) and Pedro Monteiro (now at IST, Portugal). Jean-Luc Gouzé (INRIA Sophia Antipolis) and Tewfik Sari (Université de Haute-Alsace, Mulhouse) have made contributions to the qualitative simulation method underlying the tool, while Johannes Geiselmann (Université Joseph Fourier, Grenoble) and Delphine Ropers have worked on applications to actual regulatory networks. GNA has recently been transferred to Genostar, but remains freely available for non profit academic research purposes.

General information on the **qualitative modeling and simulation of genetic regulatory networks** can be found in:

- H. de Jong (2002), Modeling and simulation of genetic regulatory systems: A literature review, *Journal of Computational Biology*, 9(1):69-105.
- H. de Jong, D. Ropers (2005), Qualitative approaches towards the analysis of genetic regulatory networks, Z. Szallasi, V. Periwal, J. Stelling (eds), *System Modeling in Cellular Biology: From Concepts to Nuts and Bolts*, MIT Press, Cambridge, MA. 125-148.

More information on the **qualitative simulation method and its extensions** can be found in:

- H. de Jong, J.-L. Gouzé, C. Hernandez, M. Page, T. Sari, and J. Geiselmann (2004), Qualitative simulation of genetic regulatory networks using piecewise-linear models, *Bulletin of Mathematical Biology*, 66(2):301-340.
- G. Batt, H. de Jong, M. Page, J. Geiselmann (2008), Symbolic reachability analysis of genetic regulatory networks using qualitative abstractions, *Automatica*. 44(4):982-989.
- G. Batt, D. Ropers, H. de Jong, J. Geiselmann, R. Mateescu, M. Page, D. Schneider (2005), Validation of qualitative models of genetic regulatory networks by model checking: Analysis of the nutritional stress response in *Escherichia coli*, *Bioinformatics*, 21(Suppl 1):i19-i28.
- R. Casey, H. de Jong, J.-L. Gouzé (2006), Piecewise-linear models of genetic regulatory networks: Equilibria and their stability, *Journal of Mathematical Biology*. 52(1):124-152.
- H. de Jong, M. Page (2008), Search for steady states of piecewise-linear differential equation models of genetic regulatory networks, *ACM/IEEE Transactions on Computational Biology and Bioinformatics*. 5(2): 208-222.
- P.T. Monteiro, D. Ropers, R. Mateescu, A.T. Freitas, H. de Jong (2008), Temporal logic patterns for querying dynamic models of cellular interaction networks, *Bioinformatics*, 24(16) :i227-i233.
- R. Mateescu, P.T. Monteiro, E. Dumas, H. de Jong (2010), CTRL: Extension of CTL with regular expressions and fairness operators to verify genetic regulatory networks, Theoretical Computer Science, accepted for publication. Special issue CMSB 2008.

- P.T. Monteiro, E. Dumas, B. Besson, R. Mateescu, M. Page, A.T. Freitas, H. de Jong (2009), A service-oriented architecture for integrating the modeling and formal verification of genetic regulatory networks, *BMC Bioinformatics*, 10:450.
- V. Baldazzi, D. Ropers, Y. Markowicz, D. Kahn, J. Geiselmann, H. de Jong (2010), The carbon assimilation network in Escherichia coli is densely connected and largely sign-determined by directions of metabolic fluxes, *PLoS Computational Biology*, 6(6) : e1000812.

A previous version of **GNA** is described in:

- H. de Jong, J. Geiselmann, C. Hernandez, and M. Page (2003), Genetic Network Analyzer: Qualitative simulation of genetic regulatory networks, *Bioinformatics*, 19(3):336-344.

The **application of GNA** is illustrated in:

- H. de Jong, J. Geiselmann, G. Batt, C. Hernandez, and M. Page (2004), Qualitative simulation of the initiation of sporulation in *Bacillus subtilis*, *Bulletin of Mathematical Biology*, 66(2):261-300.
- D. Ropers, H. de Jong, M. Page, D. Schneider, J. Geiselmann (2006), Qualitative simulation of the carbon starvation response in Escherichia coli, *Biosystems*, 84(2):124-152.
- J-A. Sepulchre, S. Reverchon, W. Nasser (2007), Modeling the onset of virulence in a pectinolytic bacterium, *Journal of Theoretical Biology*, 44(2):239-257.
- A. Usseglio Viretta, M. Fussenegger (2004), Modeling the quorum sensing regulatory network of human-pathogenic *Pseudomonas aeruginosa*, *Biotechnology Progress*, 20(3):670-678.

An introduction to the use of GNA, with modeling tricks, can be found in:

- G. Batt, B. Besson, P. Ciron, H. de Jong, E. Dumas, J. Geiselmann, R. Monte, P.T. Monteiro, M. Page, F. Rechenmann, D. Ropers, Genetic Network Analyzer: A Tool for the Qualitative Modeling and Simulation of Bacterial Regulatory Networks. In D. Thieffry et al (eds), *Bacterial Molecular Networks: Methods and Protocols*, Human press, to appear.

GNA uses the following open-source software: JGraph, SAT4J, and CUP.The model-checker web server at INRIA runs NuSMV.

1.2. Projects using GNA

GNA has been used in several research projects, including:

- COBIOS: Engineering and Control of Biological Systems: A New Way to Tackle Complex Diseases and Biotechnological Innovation (2007-2009), NEST-2005-2 PATHFINDER.
- EC-MOAN: Scalable Modeling and Analysis Techniques to Study Emergent Cell Behavior; Understanding the E. coli Stress Response (2007-2009), NEST-2005-2 PATHFINDER.
- MetaGenoReg: Towards an Understanding of the Interrelations between Metabolic and Gene Regulation: E. coli Carbon Metabolism as a Test Case (2006-2009), ANR Biologie systémique (BIOSYS).
- Hygeia: Hybrid Systems for Biochemical Network Modeling and Analysis (2005-2007), NEST-2003-1 ADVENTURE.

Part 2. Using GNA

2.1. Defining a network

The initial step in modeling a genetic regulatory network consists of the unambiguous definition of the network structure and the representation of this information in visual form. The genes, proteins, and interactions making up the network are obtained from the experimental literature and completed with plausible hypotheses. Notice that the collection of this information may take a sizeable amount of the total time invested in the modeling project, as it demands reflection on the precise biological questions to be addressed, followed by a careful evaluation of the available experimental data.

The network editor of GNA supports the modeler in defining and representing the network structure. It takes its inspiration from the visual modeling language MIM (Molecular Interaction Maps) developed by Kohn and colleagues, and adheres as much as possible to the visualization standard proposed in the Systems Biology Graphical Notation (SBGN) project. The network editor is built upon the IogmaNetwork library developed by Genostar.

Define simple genetic regulatory network

In order to illustrate the construction of a genetic regulatory network by means of the network editor, we consider a small network of three genes. Each gene codes for a protein that controls the expression of one or more target genes by binding to the promoter region. In particular, gene *b* codes for a protein that activates the transcription of gene *a*, while gene *a* and gene *c* code for proteins that form a heterodimer inhibiting the transcription of gene *b*. In addition, protein A activates gene *c*, while protein C inhibits the expression of its own gene.

The network can be created from the elements shown in the toolbar, among which we find genes and promoters, proteins, complex formation reactions, and activating and inhibiting interactions. Upon selecting a network element, the user is asked to specify the name of the gene or the protein. The position of the network elements in the editor can be changed by selecting an element and dragging it to another location using the mouse or the keyboard arrows. The network lay-out can be modified by adding or removing breakpoints in the lines representing the complex formation reactions and regulatory interactions. These breakpoints are added by means of mouse clicks while holding the **Shift** key down and can be moved like other network entities. The result for the example network is shown in Figure 2.1. To create reactions with multiple inputs and/or outputs, create first a simple reaction, then use the reaction tool again between the entity you want to add and the targeted part of the reaction (either its start, or its end).

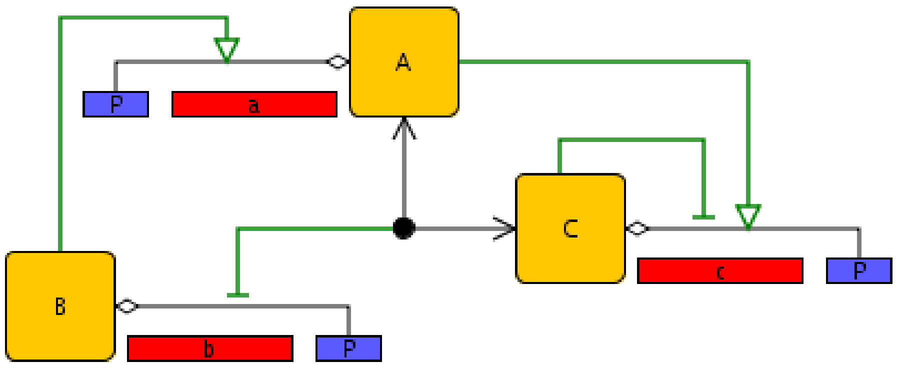


Fig. 2.1. Schema of simple genetic regulatory network

The network thus constructed is saved in a so-called **project**. You can save the project by choosing **Save project** in the **File** menu.When saving the schema, the lay-out is stored as well. You can also start a new project or load an existing project by choosing the corresponding options in the **File** menu.

Define more complex regulatory networks

In Figure 2.1 only a limited range of biochemical mechanisms involved in gene regulation was used, notably activation and inhibition of gene expression by transcription factors and the formation of protein complexes through dimerization. The network editor allows a much wider range of biochemical mechanisms to be described, as illustrated in Figure 2.2. Examples include:

- Enzymatic reactions (the reaction transforming E into F, catalyzed by D);
- Post-translational modifications (the modification of B into B*, induced by an external signal);
- Active degradation of proteins and mRNA (the degradation of C, regulated by B*).

The types of network elements and biochemical reaction mechanisms supported by the network editor are shown in the menus and the toolbar (see Section 2.8).

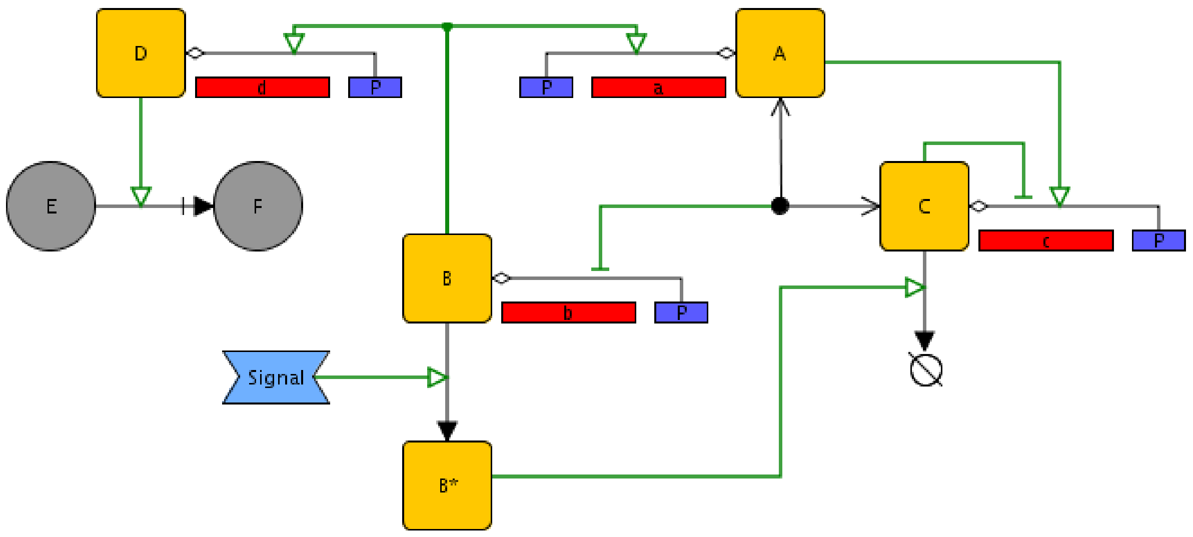


Fig. 2.2. Schema of a more complex genetic regulatory network

Reduction of regulatory network

Figure 2.2 gives an impression of the diversity of the elements composing a genetic regulatory network. For our purposes, it is convenient to have a more restricted view of a genetic regulatory network, as consisting of genes, gene products (proteins and mRNAs), and the direct and indirect regulation of gene expression exerted by the gene products. This results in a reduced network structure where boxes represent indirect regulatory interactions mediated by, for example, protein-protein interactions, metabolism, and signal transduction. The construction of these boxes is based on the assumption that the latter processes are typically fast on the time-scale of gene expression. A detailed explanation of this reduction step is beyond the scope of this tutorial, but more details can be found in publications listed in the literature section.

The reduction process is automatic and the user can see the result at any time by pressing the **Show reduced network** button in the toolbar. Figure 2.3 shows the result for the simple three-gene network in Figure 2.1. The grey circles represent the proteins and the arrows the direct and indirect regulatory interactions controlling their synthesis and degradation. Notice that the reduction step has introduced a box mediating the influence of A and C on the expression of the gene encoding protein B. This influence is exerted through the heterodimerization of A and C, an interaction no longer explicitly represented but hidden in the box. The user can easily switch back and forth between the full and reduced network representation, thus making it possible to verify the pattern of indirect interactions induced by particular hypotheses on the biochemical mechanisms involved in the regulation of a gene.

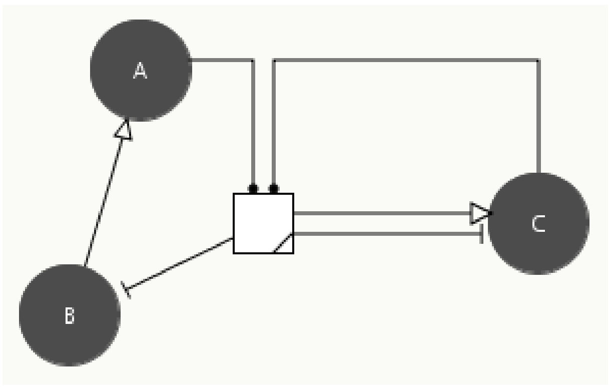


Fig. 2.3. Schema of reduced genetic regulatory network, obtained from Figure 2.1

2.2. Building a model

The definition and reduction of the genetic regulatory network under study gives rise to a schema summarizing the relevant genes, proteins, and interactions. In the next step, this schema is transformed into a GNA model of the network, by adding more information on the regulation of the genes. A GNA model has the form of a system of **piecewise-linear differential equations (PLDEs)**, completed by a set of inequality constraints. The formulation of the model will be illustrated here, by means of the simple example introduced in the previous section.

The model-building process consists of four steps:

1. Definition of state and input variables

2. Definition of parameters

3. Definition of state equations

4. Definition of inequality constraints

The model-building process can be automated to some extent by means of a model-building wizard. Each of the above steps, and their integration into the model-building wizard, is discussed in detail below.

Most of the time, you will not build a model from scratch, but start from an earlier model. In that case, you choose **Open project...** in the **File** menu to load the earlier model you have decided to use as a template. You may import models from earlier GNA versions (text files with the extension `.gna`) by choosing **Import model...** in the **File** menu. It is also possible to import a model in SBML format, by choosing **Import model from SBML...** in the same **File** menu.

Definition of state and input variables

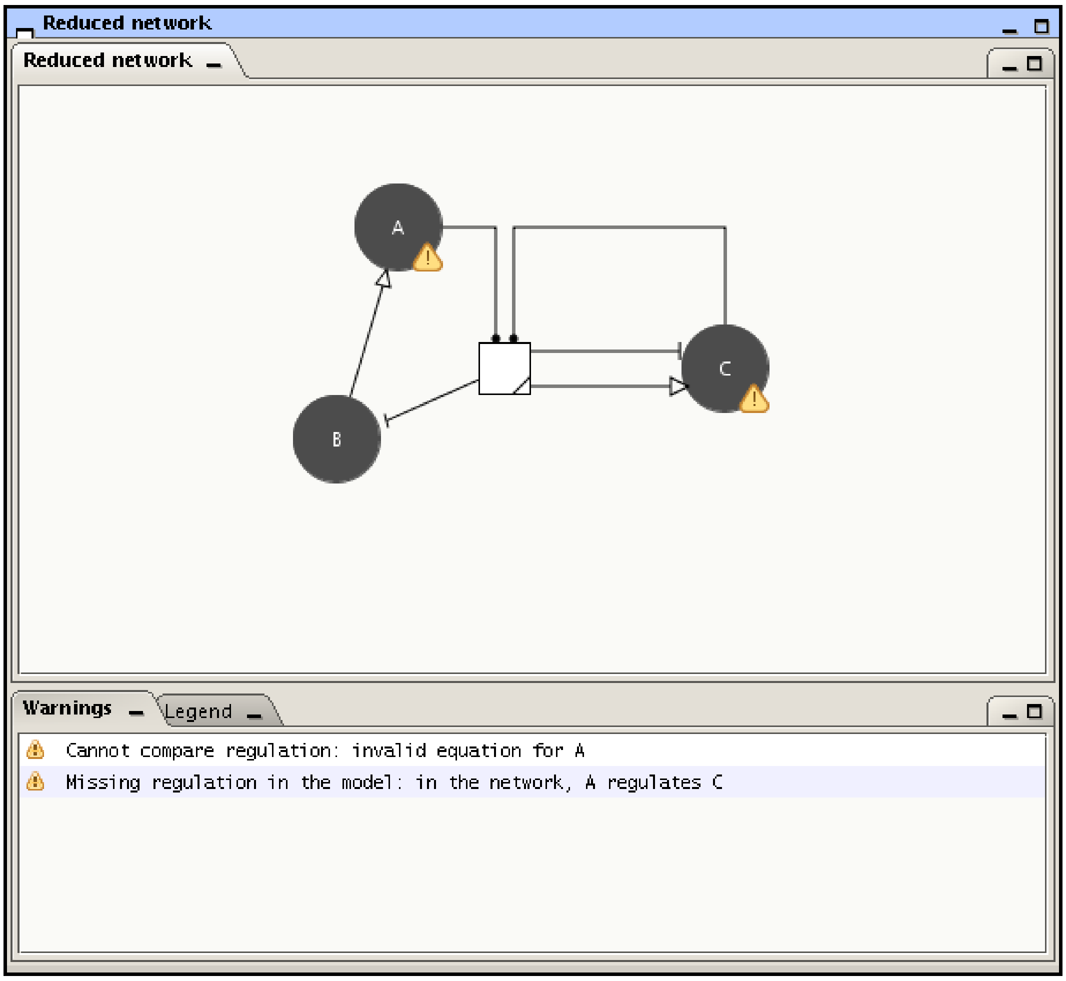
In order to complete the network into a PLDE model, you have to click the **Model** tab in the left-upper corner of the application. GNA opens a desktop with two components: the **project tree** and the **Reduced network** window. The project tree lists the variables of the model as well as the initial conditions, atomic propositions and properties, while the Reduced network window shows the network of direct and indirect genetic regulatory interactions as obtained through reduction (see Figure 2.4).

The Reduced network window is also a way to check the consistency of the model definition with the network schema. It will graphically indicate the differences between both representations. The warnings do not impact the modeling process and can be ignored, but it is usually advisable to check if the differences are intentional or not. Warnings are produced in the following cases:

- A network element does not exist as a model variable;

- A variable created in the model has no corresponding element in the network;
- The regulation of a variable differs between the network and the model (*e.g.*, in Figure 2.4 the regulation of C by A cannot be found in the state equation for C);
- The equation for a state variable is invalid.

The Legend tab allows you to quickly understand the meaning of the different entities of the reduced network. The Reduced network window will not be present if you work on a model for which no network has been defined previously.



```
Cannot compare regulation: invalid equation for A
Missing regulation in the model: in the network, A regulates C
```

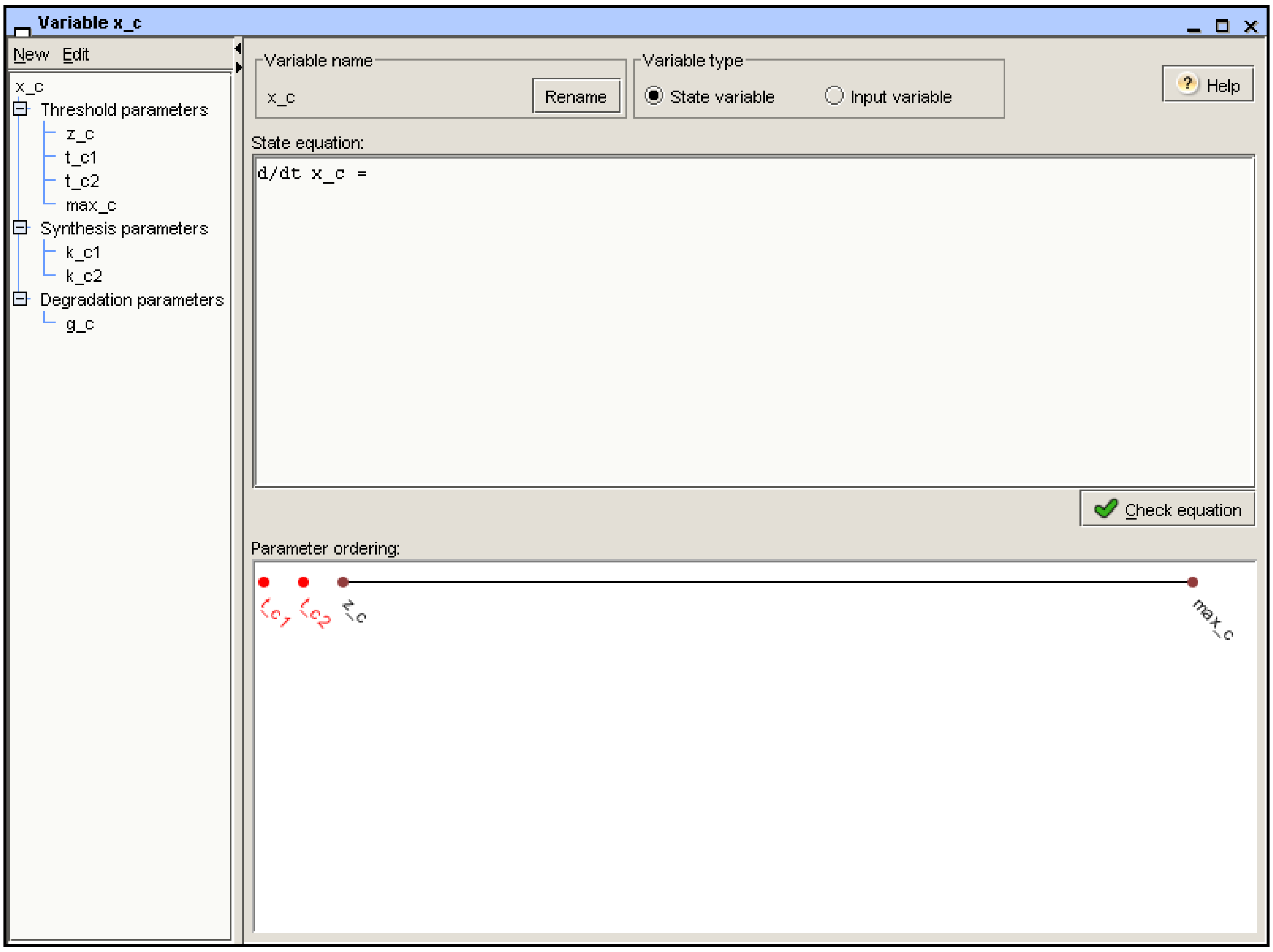
Fig. 2.4. Reduced network window with some examples of warnings

The genes in the network correspond to variables in the model, representing the concentrations of the proteins encoded by the genes. The variables are created by choosing the option **New variable** in the **Edit** menu. You can also press the right button of your mouse when the cursor is on the model item in the Project window. In the remainder of this page, we will omit the shorthand commands for menu options, which are listed in the overview of GNA functions. The creation of a variable is accomplished by specifying its name and leads to the opening of the Variable window. In the case of the example network, three variables are defined: x_a, x_b, and x_c. After their creation, variables can be renamed or deleted, by means of the options **Rename variable** and **Delete variable** in the **Edit** menu.

Two different types of variables are distinguished: **state variables** and **input variables**. While the former vary as a function of the other concentration variables in the model, the latter are determined by factors outside the scope of the system. In addition, GNA assumes that they are constant. In the Variable window, a variable can be defined as an input variable. This hides a number of fields in the Variable window that are irrelevant for input variables.

Definition of parameters

The Variable window also allows you to specify a number of parameters. Upon creation of a state or input variable, a **zero parameter** and a **maximum parameter** are automatically defined in the **parameter tree** on the left. The zero and maximum parameters indicate a lower and upper bound for the concentration of the protein, respectively. By selecting these parameters in the parameter tree, the default name assigned by GNA can be changed. For the variable x_c, we have a zero parameter zero_c and a maximum parameter max_c, as shown in Figure 2.5 below.



x_c
- Threshold parameters
  - z_c
  - t_c1
  - t_c2
  - max_c
- Synthesis parameters
  - k_c1
  - k_c2
- Degradation parameters
  - g_c

Variable name: x_c

Variable type: [✓ State variable] Input variable

State equation:

```
d/dt x_c =
```

Parameter ordering:

Fig. 2.5. Variable window after creation of the parameters for the state variable x_c

The parameter tree may also contain a number of **threshold parameters** for the variable. A threshold indicates the concentration above or below which a regulatory protein has an influence on the expression of the target gene. As a protein may be involved in several regulatory interactions, it may have several different thresholds The thresholds are created by pressing the **New** button of the tree menu, which asks the name of the threshold and inserts the newly created threshold into the tree. Once created, a threshold parameter can be deleted or renamed. In the case of x_c, we define thresholds t_c1 and t_c2, arising from the two interactions in which the protein encoded by gene *c* is involved. If two thresholds are equal, only a single threshold is defined (in other words, all thresholds are different).

Finally, if the variable is a state variable, you can specify so-called **synthesis** and **degradation parameters**. These indicate the rate of synthesis and degradation, respectively, of the protein whose concentration is represented by the variable. The synthesis and degradation parameters are specified in the parameter tree. The same buttons **New** and **Edit** allow the parameters to be created and maintained. Because there must always be at least one degradation parameter, GNA creates one by default. In the example, we introduce two synthesis parameters for x_c, k_c1 and k_c2, representing two possible synthesis rates of protein C, respectively. In addition, we introduce the degradation parameter g_c. The result is shown in Figure 2.5.

Definition of state equations

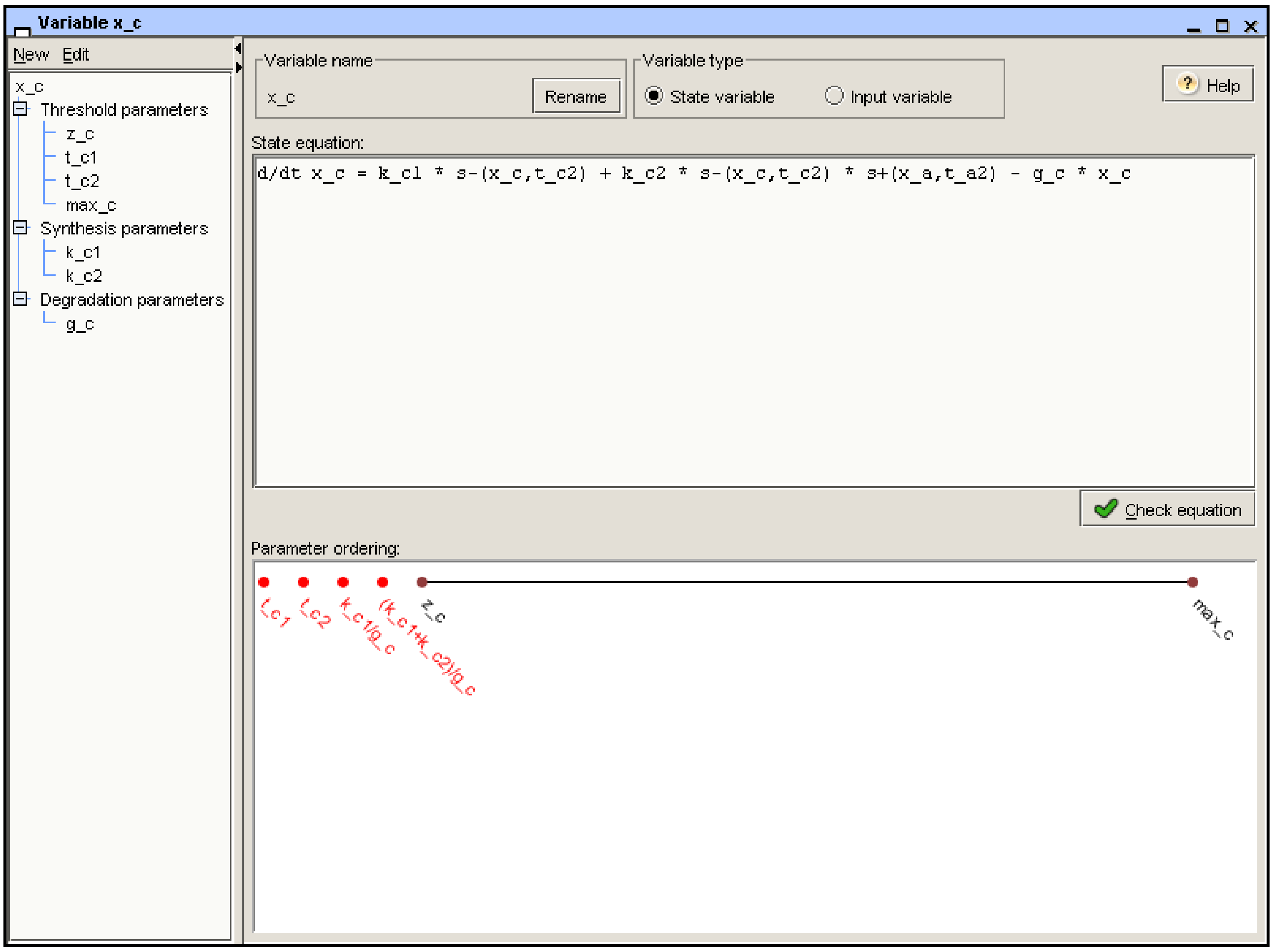
Next, still in the Variable window, we add information on the regulatory logic of the synthesis and the degradation of the proteins. That is, we specify **state equations**, which define the rate of change of the protein concentrations as a balance between the synthesis and the degradation rates of the protein. The state equations are entered in the State equation field, according to the syntax of GNA models specified in the appendix.

The **synthesis term** of the state equation is a sum of **step-function expressions**, each weighted by a synthesis parameter. Recall that, in the case of protein C, we defined two synthesis rates, `k_c1` and `k_c2`, respectively. Suppose that the protein is not produced at all, if the concentration of protein C is above its threshold `t_c2`. That is, for `x_c > t_c2`, negative autoregulation of the gene sets in. On the other hand, for `xc < t_c2`, the protein is produced at a rate which is assumed to depend on the concentration of protein A. If the latter concentration is below the threshold `t_a2`, *i.e.* `x_a < t_a2`, then gene *c* is expressed at a low rate `k_c1`. However, if `x_a > t_a2`, then RNA polymerase is stabilized by protein A, and gene *c* is expressed at a high rate `k_c1 + k_c2`. The following synthesis term captures this regulatory logic: `k_c1 * s-(x_c,t_c2) + k_c2 * s-(x_c,t_c2) * s+(x_a,t_a2)`, where `s-(x_c,t_c2)` and `s+(x_a,t_a2)` are step-function expressions. That is, `s-(x_c,t_c2)` evaluates to 1, if `x_c < t_c2`, whereas it evaluates to 0, if `x_c > t_c2`. The case of `s+(x_a,t_a2)` goes analogously, while bearing in mind that `s+(x_a,t_a2) = 1 – s-(x_a,t_a2)`.

The **degradation term** is defined analogously to the synthesis term, except that we demand that an unregulated, spontaneous degradation term is included, in order to ensure that the degradation rate of the protein is always positive. If we only have spontaneous degradation or growth dilution, as in the example of Figure 2.6, then the degradation term is simply a degradation parameter multiplied by the state variable (*e.g.*, `g_c * x_c`). In case the degradation of the protein is also regulated by other proteins, one or more step-function expressions, each weighted by a degradation parameter, are added to the degradation parameter accounting for spontaneous degradation. The resulting sum is multiplied by the state variable. For example, above a threshold concentration `t_b2`, protein B might favor degradation of protein C, by binding to it and thus presenting it to the proteolytic machinery. This can be represented by the degradation term `(g_c1 + g_c2 * s+(x_b,t_b2)) * x_c`, where `g_c1` is a low, spontaneous degradation rate, and `g_c1 + g_c2` a high, regulated degradation rate.

In general, the step-function expressions in the synthesis and the degradation terms may take many forms, depending on the number of proteins involved and the regulatory logic they implement. Further examples of step-function expressions, sometimes quite complex, can be found in the model of the initiation of sporulation in *B. subtilis* and the model of the carbon starvation response in *E. coli*.

The state equation is a differential equation whose righthand-side is the difference of the synthesis and degradation terms. In the example, the state equation for protein C is `d/dt x_c = k_c1 * s-(x_c,t_c2) + k_c2 * s-(x_c,t_c2) * s+(x_a,t_a2) – g_c * x_c`, as shown in Figure 2.6. Once entered in the State equation field, the syntax and semantics of the state equation can be checked by clicking the **Check equation** button. It is then verified that the equation satisfies the syntax of GNA models and that it is consistent with the parameter definitions in the other Variable windows.



Variable x_c

x_c
- Threshold parameters
  - z_c
  - t_c1
  - t_c2
  - max_c
- Synthesis parameters
  - k_c1
  - k_c2
- Degradation parameters
  - g_c

Variable name: x_c

Variable type: [✓ State variable] Input variable

State equation:

```
d/dt x_c = k_c1 * s-(x_c,t_c2) + k_c2 * s-(x_c,t_c2) * s+(x_a,t_a2) - g_c * x_c
```

Parameter ordering:

Fig. 2.6. Variable window after definition of the state equation for the variable x_c

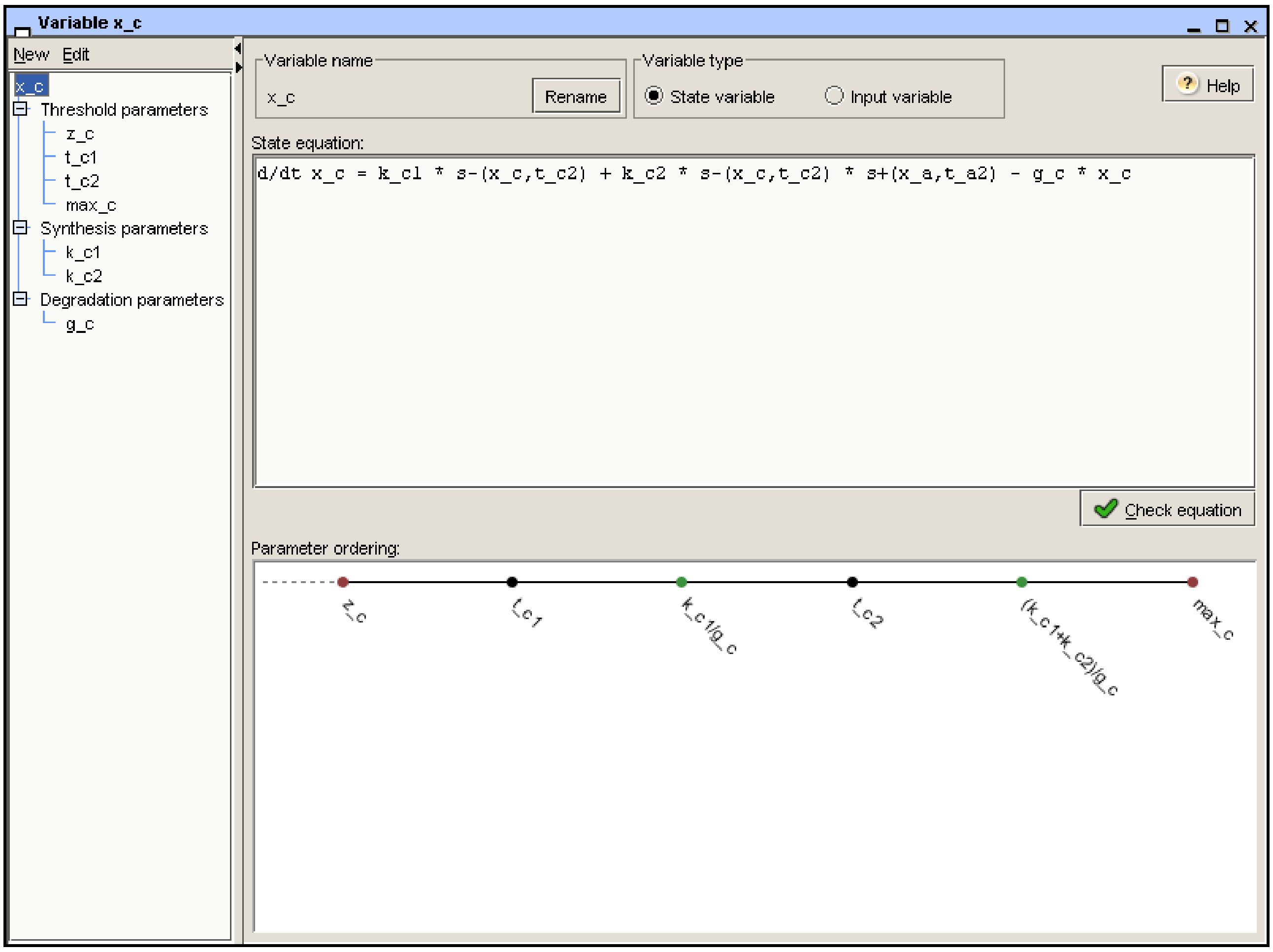
Definition of inequality constraints

The simulation method does not require precise numerical values for the threshold, synthesis, and degradation parameters. Instead, inequality constraints on the parameter values are specified to predict the qualitative dynamics of the regulatory system (see the section on running simulations). More precisely, GNA demands the user to order the so-called **threshold parameters** and **focal parameters**. We briefly explain the biological meaning of these parameters and their ordering by means of the example.

Protein C has two thresholds, `t_c1` and `t_c2`. Assume that the protein influences the expression of gene *b* at lower concentrations than it influences the expression of its own gene. This gives rise to the inequality `t_c1 < t_c2`. By default, `zero_c < t_c1` and `t_c2 < max_c`. The thresholds divide the phase space into regions in which the step functions evaluate to either 0 or 1, so that the state equations there reduce to linear and uncoupled differential equations. Inside a region, each protein concentration monotonically converges towards a so-called **focal parameter**, given by a quotient of synthesis and degradation parameters of the protein. The state equation above, for instance, implies that in a region in which `x_a` lies below `t_a2`, and `x_c` below `t_c2`, `x_c` converges towards the focal parameter `k_c2/g_c`. The behavior of the system on the threshold planes separating the regions can be determined from the behavior in the adjacent regions, as explained in the publications on the qualitative simulation method.

The possible focal parameters in the different regions of the phase space can be ordered with respect to the threshold parameters. Intuitively speaking, this defines the strength of gene expression in the regions of the phase space in a qualitative way, on the scale of ordered threshold concentrations. In the case of variable `x_c`, the possible focal parameters are 0, `k_c1/g_c`, and `(k_c1 + k_c2)/g_c`. At the highest expression level, corresponding to the focal parameter `(k_c1 + k_c2)/g_c`, protein C should be able to repress its own synthesis. This implies that `t_c2 < (k_c1 + k_c2)/g_c < max_c`. In addition, we set `t_c1 < k_c1/g_c < t_c2`.

The ordering of the threshold and focal parameters, and thus the specification of the inequality constraints, is achieved in the Parameter ordering field. It contains an axis on which the previously defined threshold, zero, and maximum parameters, as well as the focal parameters automatically generated from the state equation are positioned. The elements of the list can be ordered by means of the mouse, by dragging them up and down the axis. Elements can also be moved out of the ordered list, by dropping them left from the zero parameter. Notice that this results in a total order on the threshold and focal parameters, contrary to what was the case in previous versions of GNA (in which focal parameters did not need to be totally ordered). As a consequence, when models prepared with previous versions of GNA (version 6.0 or below) are loaded in GNA 8, the program issues a warning that the parameter order is not total and appends the unordered focal parameters at the left of the zero parameter. Figure 2.7 shows the ordered threshold and focal parameters for the variable x_c.



Variable name: x_c

Variable type: [✓ State variable] Input variable

State equation:

```
d/dt x_c = k_c1 * s-(x_c,t_c2) + k_c2 * s-(x_c,t_c2) * s+(x_a,t_a2) - g_c * x_c
```

Parameter ordering:

Fig. 2.7. Variable window after definition of the inequality constraints for the variable x_c

After you have specified the inequality constraints, the GNA model of the genetic regulatory network is complete. You can save it by means of **Save project** or **Save project as...** in the **File** menu. The project is saved in an XML file with the extension `.xml` or `.gnaml`. You may also decide to export the model to a file with the `.gna` extension (former GNA files) by the means of **Export model...** in the **File** menu, or export it to SBML using **Export to SBML...**.

Examples of complete model files can be consulted in the `samples` directory of the GNA distribution.

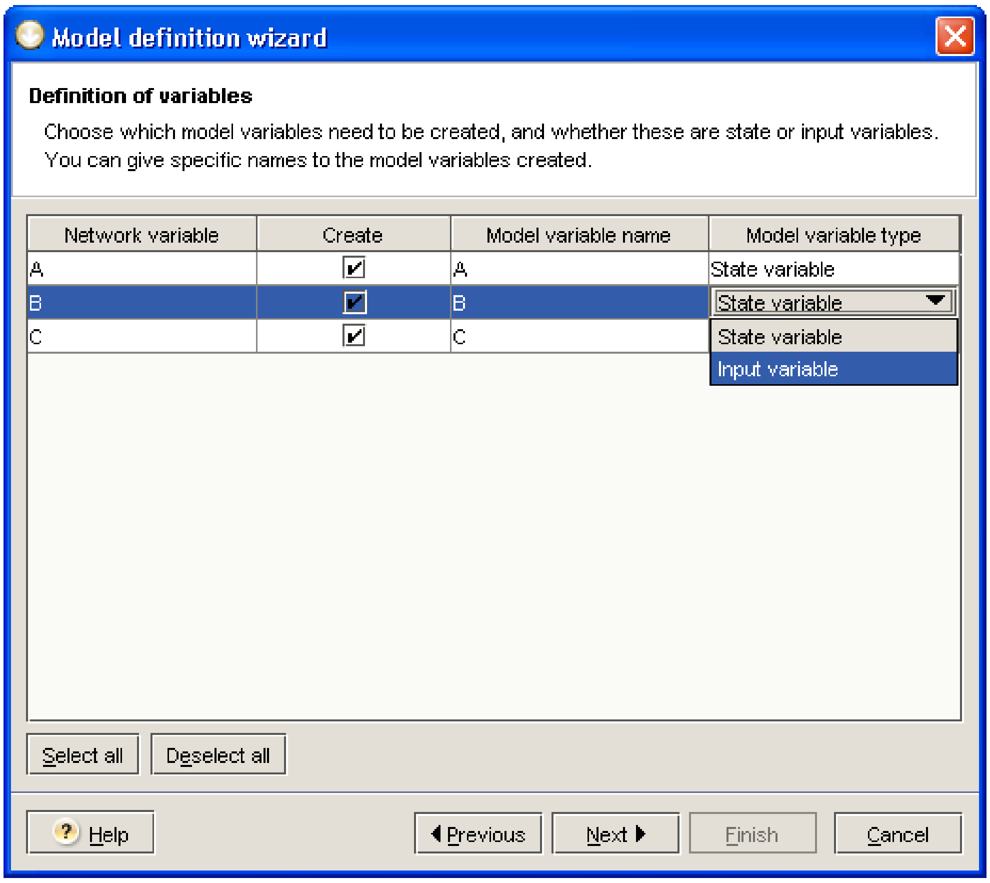
Definition of the model through the wizard

The first three steps of building a model (definition of state and input variables, definition of parameters and definition of state equations) can also be handled through a **wizard** that allows you to semi-automatically complete these steps. The model definition wizard is available if you have previously defined a network. It is launched the first time you switch from the network to the model editor mode. Even if the wizard is capable of dealing with only a resttricted range of regulating mechanisms, and thus may not give correct results for complex networks, it is usually worth the effort to run it in order to prepare a basic - perhaps partial - model that will facilitate and speed up the rest of the modeling process.

The first step of the wizard allows you to decide which variables will be created in the model (see Figure 2.8). The default proposal is based on the following rules:

- Genes and external signals will be used to create corresponding model variables;
- Among these variables, those regulated by other variables will be tagged as state variables;
- Other variables, not regulated, will be tagged as input variables and will not have a state equation associated. This will allways be the case for external signals.

You can change this default if it does not correspond to what you want, as well as the name of the model variables.


Model definition wizard

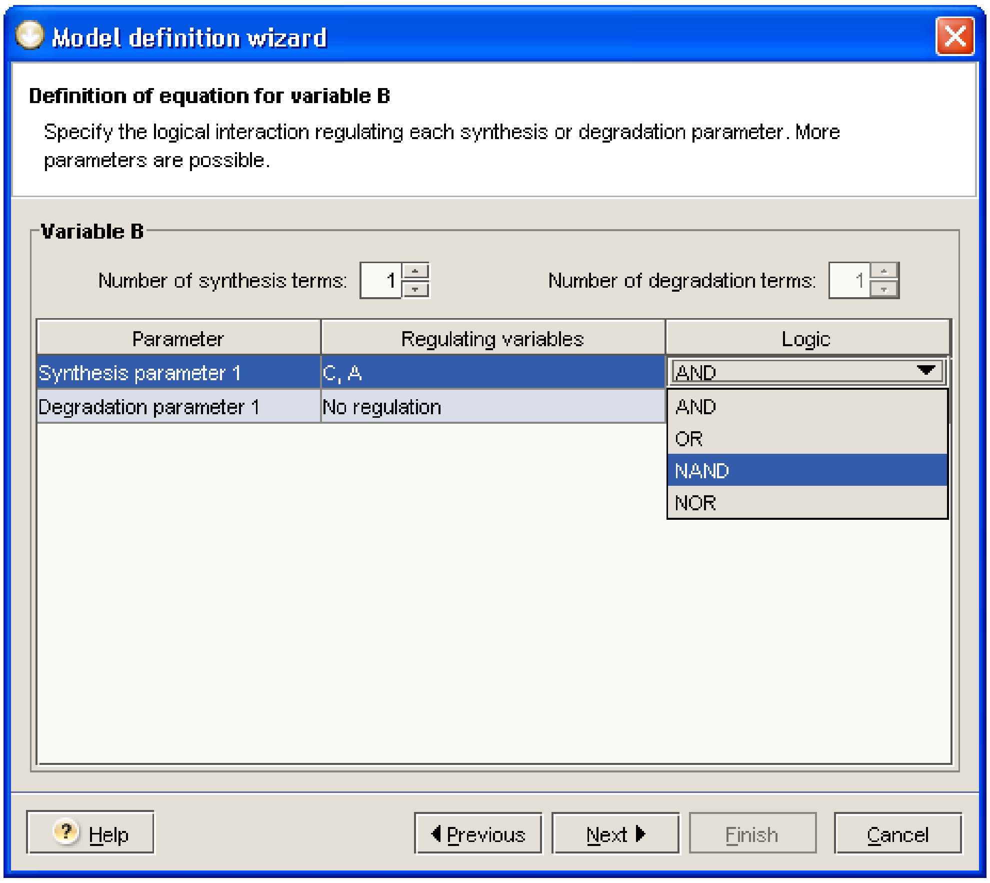
Definition of variables

Choose which model variables need to be created, and whether these are state or input variables.
You can give specific names to the model variables created.

Network variable	Create	Model variable name	Model variable type
A	✓	A	State variable
B	✓	B	State variable
C	✓	C	

Fig. 2.8. Wizard window for the definition of state and input variables

The second step of the wizard consists in defining an equation for each state variable you want to take into account (you may decide not to let the wizard algorithm generate an equation for some of the variables and define them yourself later on). For a specific state variable, the wizard takes all regulating variables into account to generate a consistent equation. You can specify various parameters, such as the number of synthesis and degradation parameters, the variables implied for each of these parameters, and the regulatory logic applied (you can choose between AND, NAND, OR and NOR). This is illustrated in Figure 2.9. When the options fit your needs, click on the **Next** button to reach the next step of the model-building-process.



Definition of equation for variable B

Specify the logical interaction regulating each synthesis or degradation parameter. More parameters are possible.

Variable B

Number of synthesis terms: 1   Number of degradation terms: 1

Parameter	Regulating variables	Logic
Synthesis parameter 1	C, A	AND
Degradation parameter 1	No regulation	

Fig. 2.9. Wizard window for the definition of the state equation of B

When working with the wizard, you can go back one or more steps, or even cancel the process. When you have successfully completed all steps of the wizard, you still have to order the parameters for each variable. After this you can start specifying initial conditions and continue simulating and analyzing the model.

2.3. Specifying initial conditions

As explained in the section on model building, the threshold parameters divide the phase space into regions. These regions can be further refined by means of the focal parameters, so as to arrive at a partition of the phase space consisting of **domains** in which the concentration variables have a unique derivative sign pattern (see the publications on the simulation method for mathematical details). Stated in a different way, the resulting partition guarantees that the system behaves in a qualitatively-identical way everywhere in a domain.

Before starting the simulation, you must specify **initial conditions** in the form of an initial domain or an initial set of domains. This is achieved by selecting the **New initial conditions** in the **New** submenu of the **Edit** menu. This opens the Initial conditions window, in which you can specify (strict) lower and upper bounds on the variables by drawing appropriate intervals by means of the mouse. The lower and upper bounds consist of threshold or focal parameters. Opening the window also adds the new initial conditions to the project tree.

Suppose that we want to simulate the example network of Figure 2.1 from initial conditions `zero_a <= x_a < t_a1`, `zero_b <= x_b < t_b`, and `zero_c <= x_c < t_c1`. That is, we assume that the concentrations of the variables are between zero and their first threshold. Figure 2.10 shows the corresponding Initial conditions window.

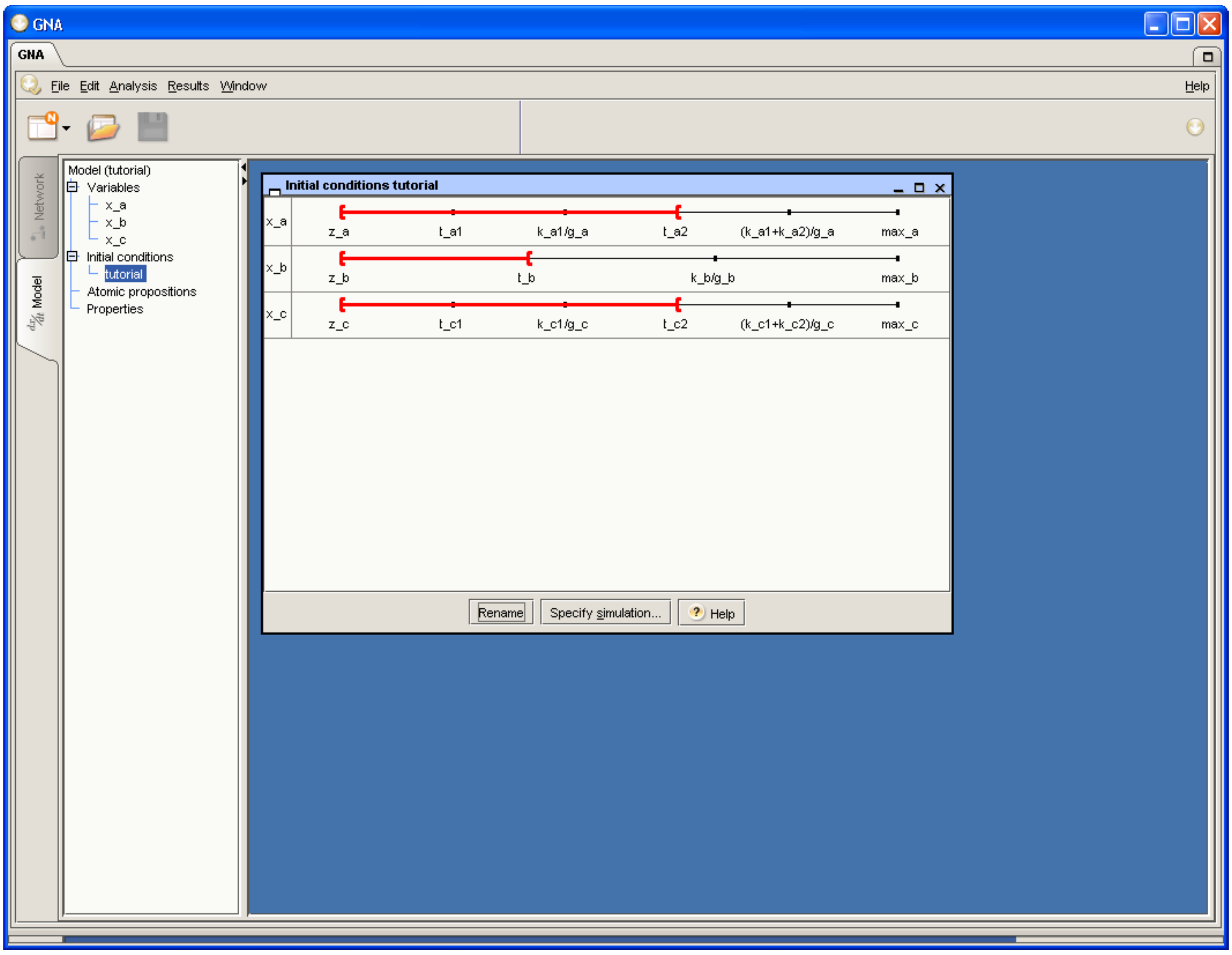


Fig. 2.10. Initial conditions window after definition of the initial conditions

The initial conditions used by GNA are saved in the same project file as the model. However, you can also import and export initial conditions separately as files with the extension `.dat`, compatible with earlier versions of GNA. To this end, use **Import initial conditions...** and **Export initial conditions...** in the **File** menu.

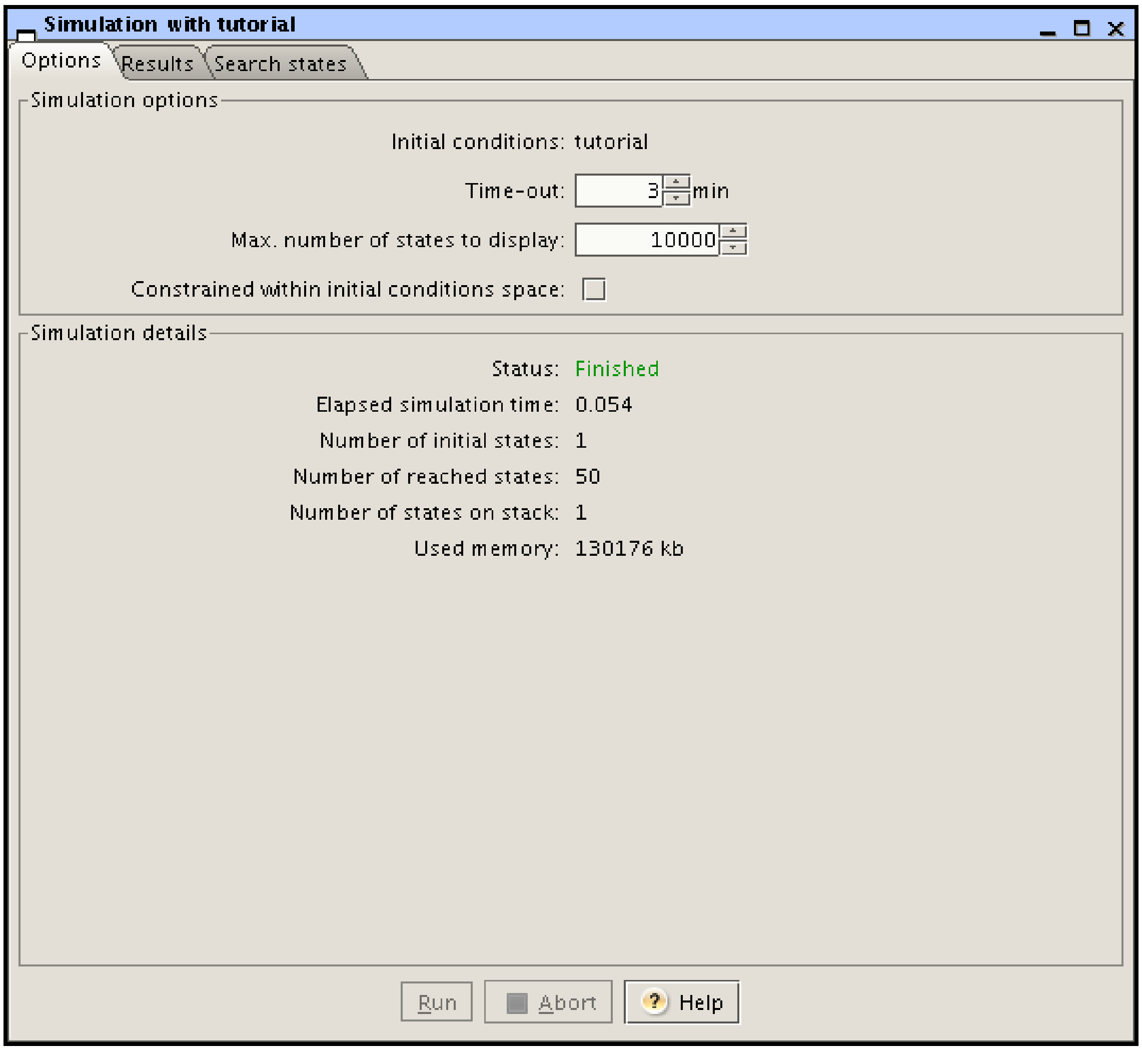
Examples of other initial conditions files can be consulted in the `samples` directory of the GNA distribution.

2.4. Running simulations

In every domain of the phase space, the system behaves in a qualitatively identical way, in the sense that the concentration variables have a unique derivative sign pattern in the domain (see the section on the specification of initial conditions). For this reason, a domain can be seen as representing a **qualitative state** of the system. A **transition** between qualitative states occurs, if the concentration variables have evolved in such a way that the system leaves one domain and enters another. The simulation algorithm of GNA recursively computes all qualitative states that are reachable through one or more transitions from the qualitative state(s) corresponding to the initial domain(s). This results in a **state transition graph**, summarizing the qualitative dynamics of the system. The graph contains **qualitative steady states** or **qualitative cycles**, which may correspond to steady states and limit cycles, respectively, of the underlying piecewise-linear differential equations. Notice that, due to the mathematical techniques used to describe the dynamics of the system in threshold planes, the steady states located on the thresholds of the concentration variables may be sets instead of points (see the GNA publications for mathematical details).

Once you have defined a model of the genetic regulatory network and appropriate initial conditions, GNA allows you to perform a qualitative simulation of the network. First, you need to define the simulation parameters, either by choosing **Specify simulation** in the **Analysis** menu, or by pressing the corresponding button in the Initial conditions window. This opens the Properties page of the Simulation window, in which you can set the time-out parameter and the maximum number of states of the state transition graph that will be displayed. In addition, you can specify whether the simulation should be constrained to the initial conditions only or not. That is, by checking this option GNA only computes the transitions between the qualitative states corresponding to domains included in the initial conditions, and does not consider transitions to other states. This gives rise to a reduced state transition graph, summarizing the qualitative dynamics of the system in a specific region of the phase space, for instance the dynamics of the system around a steady state.

The actual simulation is launched by choosing **Run** in the Simulation window or in the **Analysis** menu. The program displays run-time statistics in the Simulation window, as shown in figure Figure 2.11, and allows you to abort the simulation.



Initial conditions: tutorial
Time-out: 3 min
Max. number of states to display: 10000
Constrained within initial conditions space: ☐

Status: Finished
Elapsed simulation time: 0.054
Number of initial states: 1
Number of reached states: 50
Number of states on stack: 1
Used memory: 130176 kb

Fig. 2.11. Simulation window after running the model of the genetic regulatory network in Figure 2.1

A qualitative simulation can also be launched from the console using the MS-DOS, Linux, Unix, or MacOS command:

```
% gna [-p] [-v] [-q] [-e <plugin-directory>]
    [-f] [-i <file>] <project/model>
```

The argument `project/model` is mandatory and refers to the project file (extension `.gnaml`) or the model file used in previous GNA releases (extension `.gna`). The `-p` option (also accessible through `--presenta-`

tion-mode) is set to *user* when GNA is started from the command line by the user, and to *system* when GNA is called from within another application. The trace level determines the amount of trace information generated during the simulation, and can be adjusted through the -v (--verbose) and -q (--quiet) options. If the project does not contain any initial conditions, and the -i (--initial-conditions) option is not specified, the regulatory network is simulated for all possible initial conditions. That is, GNA generates a state transition graph containing all possible qualitative states of the system. (However, note that in enumerating the possible qualitative states, GNA omits any qualitative states associated with domains in which an input variable assumes a threshold value).

As will be explained in section Section 2.7, the simulation results can be analyzed by means of a model checker. The command **gna** is able to generate model-checker specific output, as defined by customized plug-ins. The option -e (--export) provides the plug-in directory to be used. If the simulation results are in the form of an explicit state transition graph, one can additionally filter the instantaneous states of the state transition graph through the -f option (--filter-instantaneous-states).

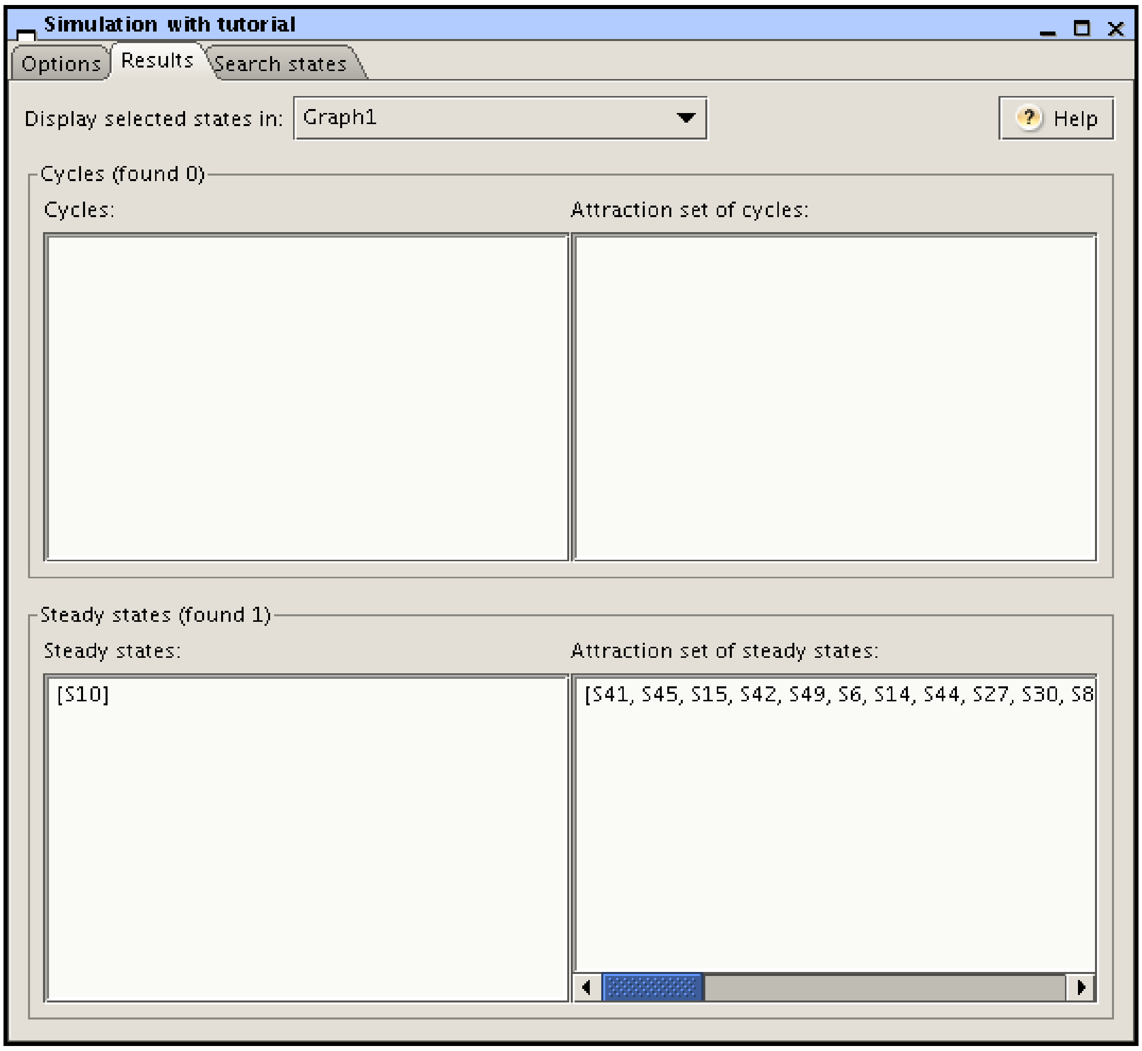
The command

```
% gna --help
```

displays the meaning of the arguments.

2.5. Interpreting simulation results

The Results page in the Simulation window summarizes the results obtained for a qualitative simulation (Figure 2.12). It displays the qualitative steady states and qualitative cycles that are reachable from the initial qualitative states, as well as the **attraction set** of each qualitative steady state or qualitative cycle (*i.e.*, the set of qualitative states from which the attractor is reachable). The Results page in the Simulation window also gives the name of the Transition graph window, containing the state transition graph resulting from the simulation (Figure 2.13). In the example, the state transition graph consists of 50 qualitative states, associated with domains lying between threshold or focal planes (regulatory domains), or domains lying on one or more threshold or focal planes (switching domains). In order to graphically distinguish the qualitative states associated with regulatory and switching domains, the former have an emphasized border. In addition, qualitative steady states are shaded.



Display selected states in: Graph1

Cycles (found 0)

Steady states (found 1)

[S10]

[S41, S45, S15, S42, S49, S6, S14, S44, S27, S30, S8

Fig. 2.12. Simulation window displaying results of a qualitative simulation of the genetic regulatory network in Figure 2.1: Qualitative steady states, qualitative cycles, and their attraction sets

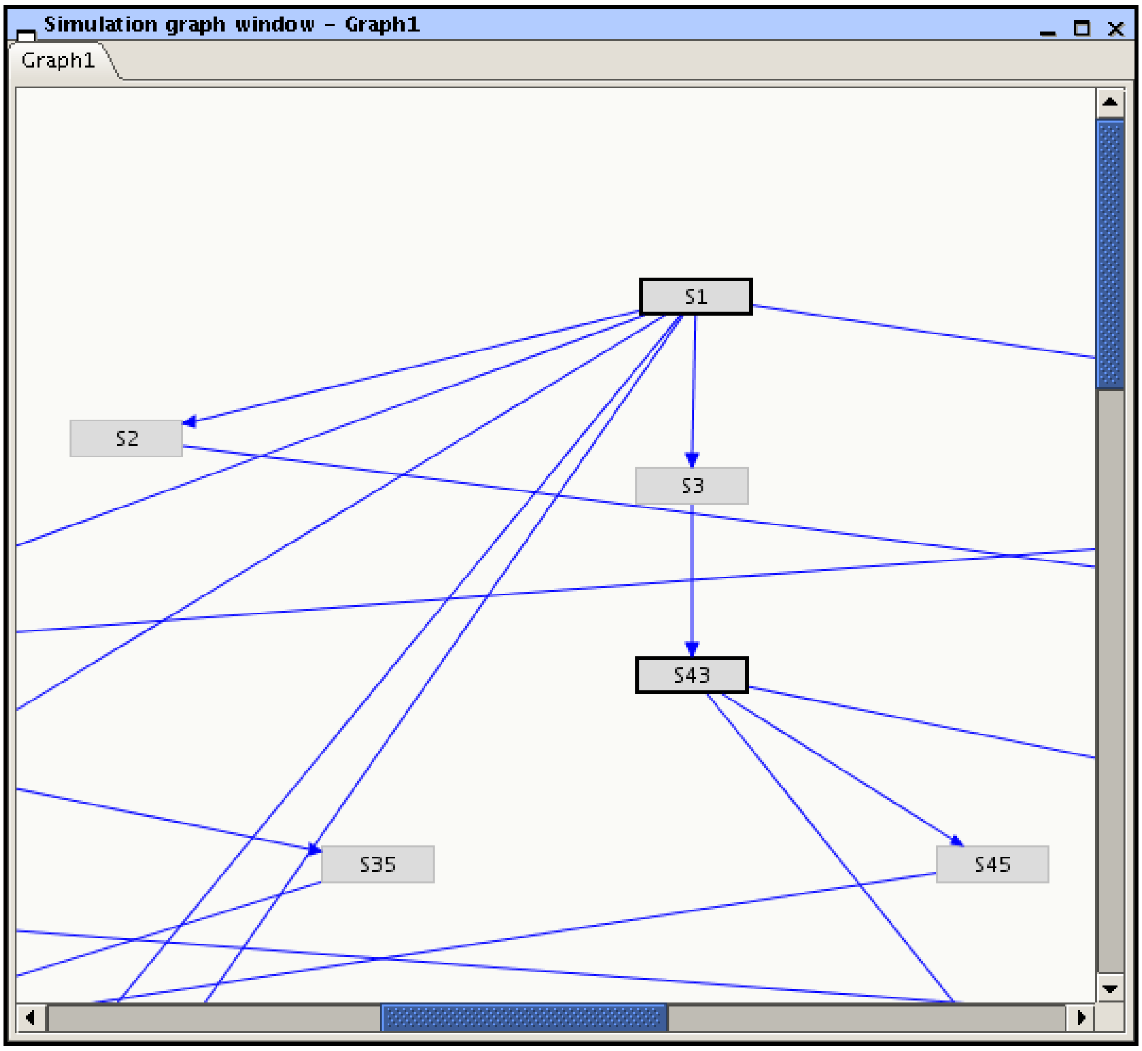


Fig. 2.13. Results of a qualitative simulation of the genetic regulatory network in Figure 2.1: Fragment of the state transition graph

The state transition graph contains more detailed information on the qualitative dynamics of the system, which can be analyzed by means of the graphical user interface. First, by hovering the mouse cursor on a qualitative state, the properties characterizing the state, notably its location in the phase space, are displayed in textual form. Press the **F2** key if you want to focus on it. Second, you can select a path in the state transition graph and use the option **Display variables in selected path** in the **Results** menu to show the qualitative evolution of the protein concentrations along the selected path. The states are selected by means of the mouse, while pressing the **Ctrl** key. A click on a selected state, again while pressing the **Ctrl** key, allows this state to be deselected. In Figure 2.14, you can see a path selected in the state transition graph of Figure 2.13. Notice that the order in which the states have been selected determines the order in the concentration profiles shown. If the selected order is consistent with the successor relations in the state transition graph, then the path can be interpreted as a temporal order of events. More precisely, the concentration profile then shows how the derivative signs of the variables evolve over time, giving rise to a sequence of maxima and minima for each of the variables. This information can be straightforwardly compared with available gene expression data. Similarly, the option **Display focal sets in selected path** in the **Results** menu shows the qualitative evolution of the values towards which the state variables locally converge.

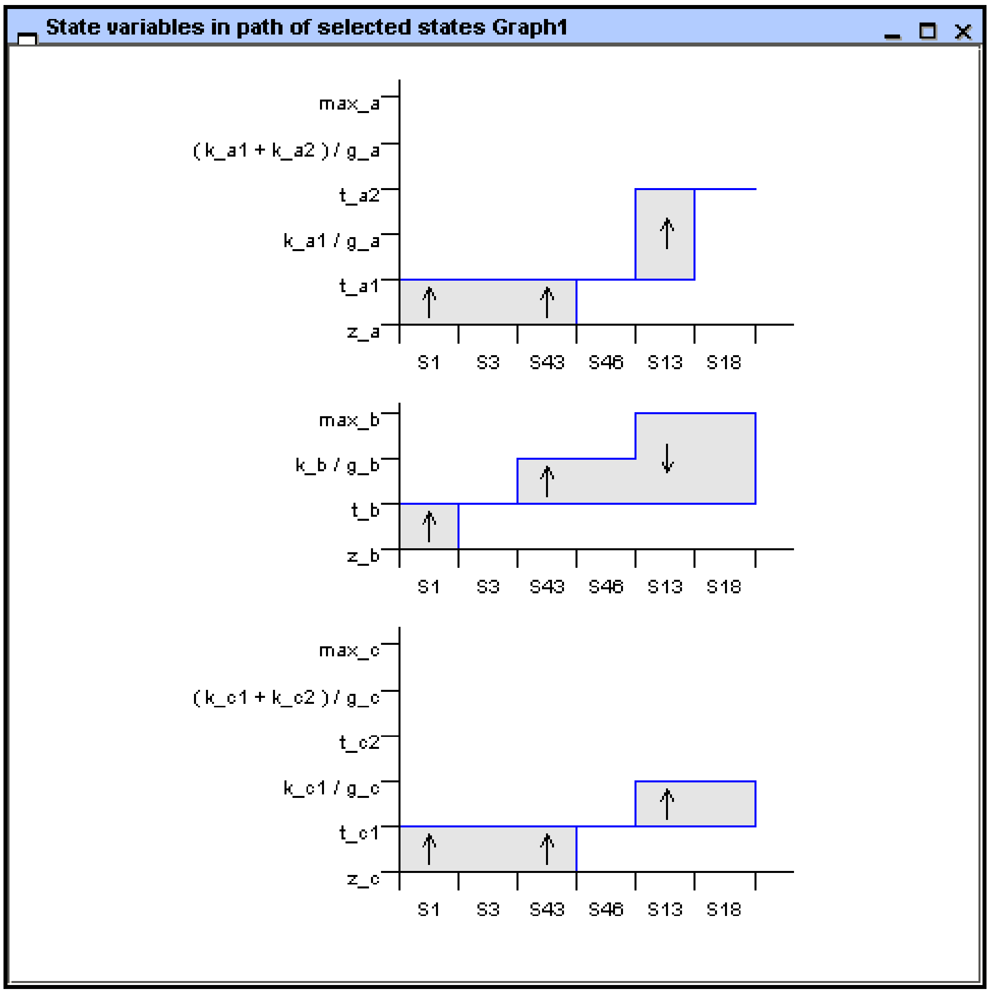


Fig. 2.14. Qualitative evolution of the variables x_a, x_b, and x_c in a path selected in the state transition graph of Figure 2.13

For more complex genetic regulatory networks, the state transition graph produced by GNA may contain hundreds or thousands of states. Depending on the display limit set in the Properties page of the Simulation window, only part of the state transition graph is then shown. The graphical user interface provides a few functions to analyze this graph. First, the zoom functions in the **Results** menu - more conveniently accessed through the right mouse button (see the overview of GNA functions) - allow you to increase or decrease the visible part of the graph. Second, you can select a subset of the state transition graph, display it in a separate tab, and expand it using several levels of successors and/or predecessors. Third, a useful option is to reduce the graph by eliminating qualitative states that correspond to domains that are instantaneously traversed, and therefore have no practical relevance. This option, called **Filter instantaneous states** in the **Results** menu, eliminates 32 of the 50 qualitative states of the state transition graph in our example. Fourth, the Search states page in the Simulation window allows you to identify parts of the state transition graph that satisfy certain search criteria. Finally, once a single state is selected it is possible to navigate through the transition graph using the keyboard arrows (maintain the **Alt** key pressed to add new states to the selection).

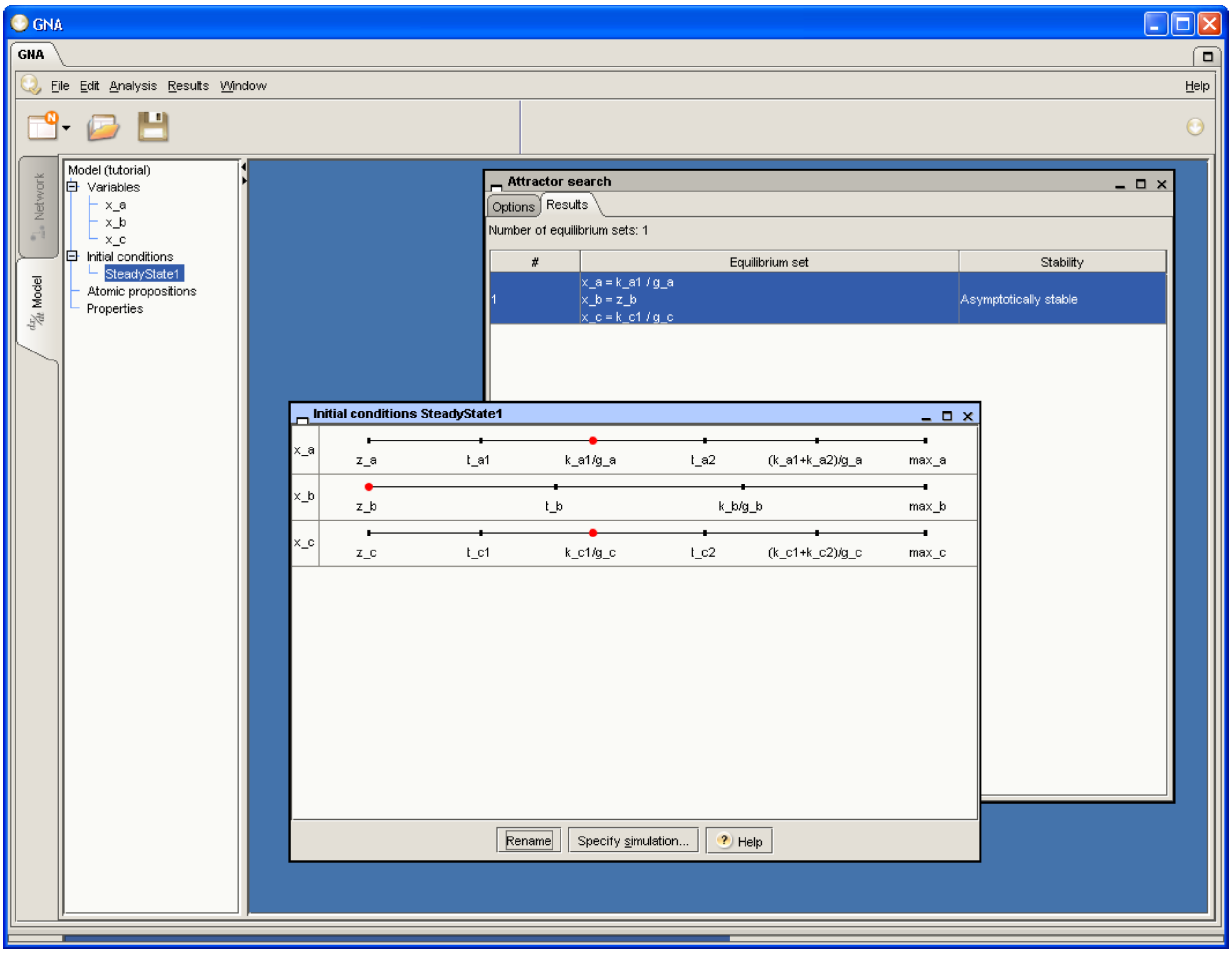
For really large models, the above functions may not be sufficient, and other solutions need to be tried. GNA offers two functionalities that are of particular interest for the analysis of large state transition graphs. First, you can export your graph to a model-checking tool and further analyze its properties by formulating queries in a temporal logic (see section on checking properties of the state transition graph). Second, instead of generating the entire state transition graph, you can use the attractor search option (see the section on searching for attractors).

2.6. Searching for attractors

A qualitative simulation results in a state transition graph, consisting of qualitative states and transitions between qualitative states. When dealing with models of complex genetic regulatory networks, the state transition graph may become too large to be handled by the graph analysis functions of GNA. In many cases it is not necessary to generate the state transition graph and it suffices to know which are the attractors of the system, that is, the qualitative steady states and qualitative cycles. The aim of the **attractor search** function of GNA is to locate the qualitative steady states of the system, without generating the state transition graph.

The **stability** of a qualitative steady state corresponds to the stability of a steady state, or more generally a steady state set, of the underlying piecewise-linear differential equations. The stability can be determined from the local topology of the state transition graph (see GNA publications). While a steady state not located on a threshold plane is always asymptotically stable, the stability of a steady state or steady state set located on a threshold plane can often be inferred from the incoming and outgoing transitions in the state transition graph. More precisely, if there is an incoming transition from all neighboring qualitative states, and no outgoing transition, then the equilibrium set is **asymptotically stable**. On the other hand, if there is at least one outgoing transition, then the steady state is **unstable**. In the other cases, the stability of the steady state cannot be determined from the topology of the state transition graph alone, and further mathematical analysis is required. The above notions of stability can be generalized to the steady state set sometimes occurring in piecewise-linear models (see GNA publications).

Once you have defined a model of the genetic regulatory network, GNA allows you to search for the steady states of the network. First you need to specify the search parameters, by choosing **Search attractors...** in the **Analysis** menu. This opens the Properties page of the Attractor search window, in which you can set the time-out parameter. The actual search for steady states is launched by choosing **Run** in the Attractor search window. The program displays run-time statistics in the Attractor search window and allows you to abort the attractor search.



Number of equilibrium sets: 1

#	Equilibrium set	Stability

1	x_a = k_a1 / g_a
	x_b = z_b
	x_c = k_c1 / g_c
		Asymptotically stable

Fig. 2.15. Attractor search window showing the results of the search for qualitative steady states in the model of the genetic regulatory network in Figure 2.1. The system has a single, asymptotically stable steady state. The attractor search also generates new initial conditions, consisting of the qualitative steady states that were found.

The Results page in the Attractor search window summarizes the results obtained (Figure 2.15). More particularly, the page contains a list of qualitative steady states, together with an indication of their stability (**asymptotically stable**, **unstable**, or **undetermined**). Sometimes several adjacent qualitative steady states correspond to a single steady state set in the underlying piecewise-linear model. In this case, GNA groups the qualitative steady states together. If GNA is not able to determine all steady states or group them together (a computationally expensive step) before the time-out is reached, the partial results are presented.

Each qualitative steady state is also added as an initial condition in the project tree. By performing a qualitative simulation constrained to neighborhood of the initial condition corresponding to a qualitative steady state, you can get an idea of the topology of the state transition graph around this state. This is especially useful if the stability of the steady state is left undetermined by GNA. Figure 2.15 shows that the example model only has a single qualitative steady state, which is not located on a threshold plane and therefore asymptotically stable. The initial conditions `SteadyState1` are generated.

An attractor search can also be launched from the console using an MS-DOS, Linux, Unix, or MacOS command, like for a qualitative simulation:

```
% gna-attractor <project/model>
```

The argument `project/model` refers to the file with the model. Typing

```
% gna-attractor --help
```

in the console displays the meaning of the arguments.

2.7. Checking properties of the state transition graph

Another way to deal with large state transition graphs is to analyze the graph by means of formal verification techniques, implemented in computer tools called model checkers. Examples of the kind of properties one might want to test are "*It is possible for a state with an `x_a` concentration equal to `t_xa2` to occur*" or "*If the system is in the state corresponding to the initial conditions `zero_a <= x_a < t_a1`, `zero_b <= x_b < t_b`, and `zero_c <= x_c < t_c1`, then it necessarily reaches a steady state*".

The use of the model-checking tools involves two steps:

1. The specification of the property to be verified by defining atomic propositions and formulating so-called temporal-logic formulas;

2. The actual verification of the property by a model checker.

We will illustrate these steps by means of one of the above-mentioned properties for the example network.

Specification of atomic propositions

An **atomic proposition** is used to characterize qualitative states. It indicates, for each variable in the model, its value, the value toward which it converges and the sign of its derivative. In addition, it specifies state descriptors indicating, *e.g.*, whether the state is a steady state.

An atomic proposition can be created by means of **New atomic proposition** in the Edit menu: a new atomic proposition is created and the corresponding Atomic proposition window opens (see Figure 2.16). The window is composed of 4 tabs:

- the Value tab, where the user can choose (strict) lower and upper bounds for the value of each variable (following the same principle as the specification of the initial conditions);

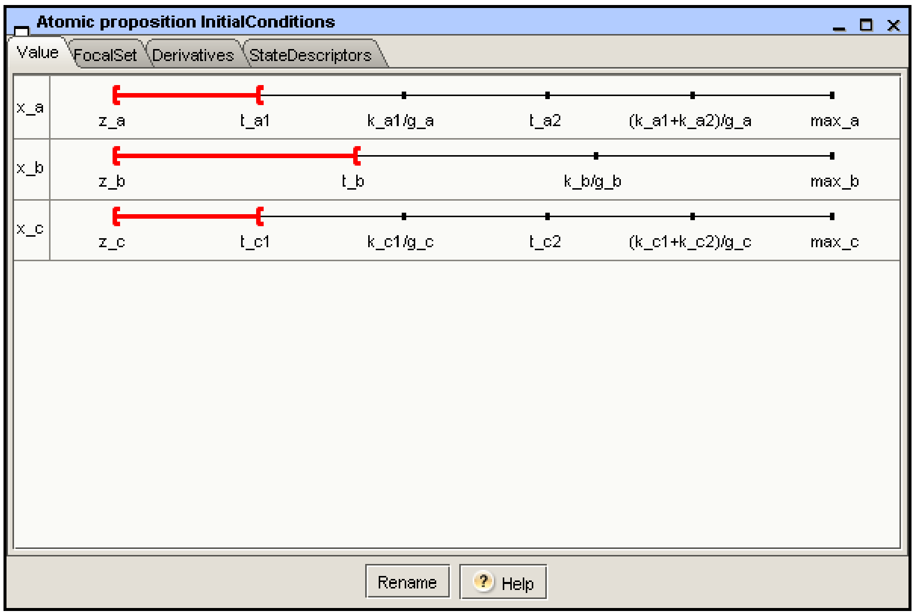
- the FocalSet tab, where the user can choose (strict) lower and upper bounds for the focal value of each variable. Within each region of the state space, the system converges towards a so-called focal set (see the publications on GNA);

- the Derivatives tab, where for each variable, the user can select the appropriate derivative sign (Undefined, Zero, Positive, Negative, All);

- the StateDescriptors tab contains check boxes that can be selected if the user wants to specify that the state is a steady state, an initial state, in a strongly connected component, or in a terminal cycle of the state transition graph.

A newly-created atomic proposition has a default specification that does not impose any specific constraints, and is therefore **true** for all states in the state transition graph.

In order to verify the property "*If the system is in the state corresponding to the initial conditions `zero_a <= x_a < t_a1`, `zero_b <= x_b < t_b`, and `zero_c <= x_c < t_c1`, then it necessarily reaches a steady state*", we need to create two atomic propositions. One is called `InitialConditions` and specifies the initial conditions `zero_a <= x_a < t_a1`, `zero_b <= x_b < t_b`, and `zero_c <= x_c < t_c1`. The other is called `SteadyState` and specifies that the state is a steady state (see Figure 2.16).



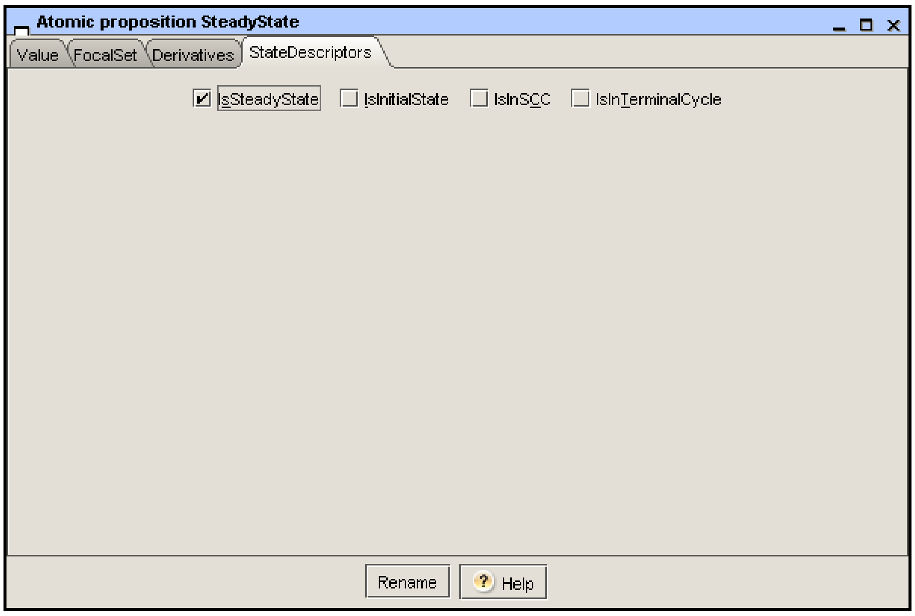


Fig. 2.16. Atomic proposition windows, showing the specification of the atomic propositions `InitialConditions` and `SteadyState`.

Formulation of temporal logic formulas

The atomic propositions are included in the temporal-logic property, defined by means of a special editor. The user can call the editor by opening the **Edit** menu and choosing the **New property** item. It opens the Property editor window which is composed of two tabs, corresponding to two different ways to formulate a property. The first is a pattern-based wizard that allows you to formulate a property without any knowledge of temporal logic. The second is a free-text interface that provides all the flexibility of the temporal logic supported by GNA, called CTRL. This logic subsumes standard temporal logics like CTL and LTL (see GNA publications).

A pattern is a high-level query template that captures recurring questions and can be automatically translated into temporal logic.

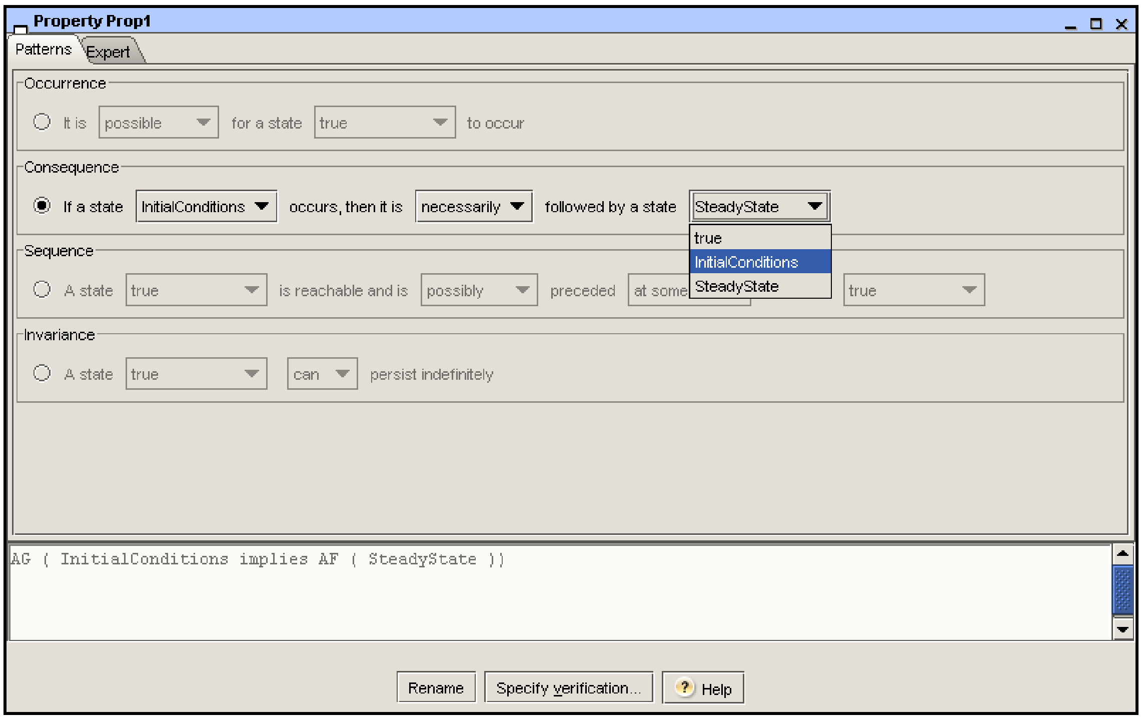
Four different patterns, each with some variants, are available to the user. Together they cover most of the frequently-asked questions by modelers (see GNA publications).

- Occurrence: It is (not) possible for a state satisfying an atomic proposition p to occur;

- Consequence: If a state satisfying an atomic proposition p occurs, then it is possibly (or necessarily) followed by a state satisfying an atomic proposition q;
- Sequence: A state statisfying an atomic proposition q is reachable and is possibly (or necessarily) preceded at some time (or all of the time) by a state p;
- Invariance: A state satisfying an atomic proposition p can (or must) persist indefinitely.

In the pattern wizard, you first click on the radio button for the pattern of your choice, and then use the combo boxes to choose existing atomic propositions and the specific pattern variant of interest (the default choice for the atomic propositions is true).

The temporal-logic translation of the property appears in the text area at the bottom of the window. This is illustrated by Figure 2.17 for the property "*If the system is in the state corresponding to the initial conditions* `zero_a <= x_a < t_a1`*,* `zero_b <= x_b < t_b`*, and* `zero_c <= x_c < t_c1`*, then it necessarily reaches a steady state*". The property is defined as a **Consequence** pattern, using the previously-defined atomic propositions `InitialConditions` and `SteadyState`.

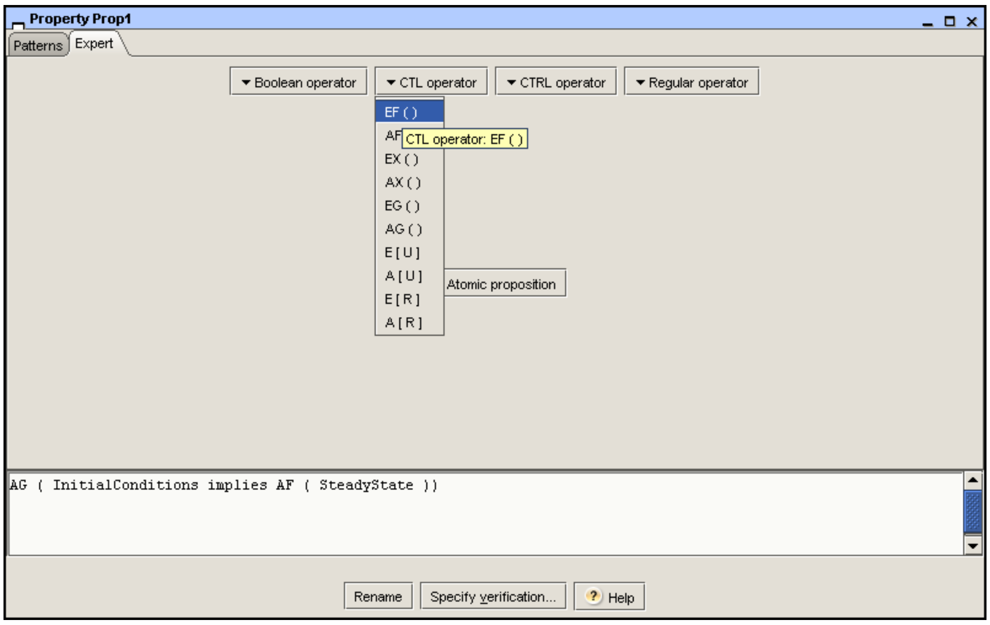


```
AG ( InitialConditions implies AF ( SteadyState ))
```

Fig. 2.17. Pattern wizard of the property editor, showing the pattern definition of the property "*If the system is in the state corresponding to the initial conditions* `zero_a <= x_a < t_a1`*,* `zero_b <= x_b < t_b`*, and* `zero_c <= x_c < t_c1`*, then it necessarily reaches a steady state*" and its translation into temporal logic.

If you are comfortable with the CTRL temporal logic, you can alternatively select the Expert tab. This allows you to directly enter the property by typing it in the text area. Another way to edit a property is by using the combo box system, selecting the corresponding operators and atomic propositions, which are directly inserted in the text area, as shown in Figure 2.18. To check if the temporal logic specification of the property is well-formed and compliant with the CTRL grammar, you can choose the Check syntax button. The syntax of CTRL formulas is defined in the appendix.

The temporal-logic properties and atomic propositions are saved in the same project file of the model. However, they can also be imported and exported as files with the extension `.prop`. To this end, use the **Import** and **Export** items in the **File** menu.



```
AG ( InitialConditions implies AF ( SteadyState ))
```

Fig. 2.18. Expert tab of the property editor, used for specifying the same property as in Figure 2.17

Verification of temporal logic formulas

Once the temporal logic formula has been completely specified, you can check its validity by means of the model-checking server located at INRIA. This is done in two different ways: either through the property editor, by clicking on the **Specify verification** button, or through the item with the same name in the Analysis menu. In both cases, a Verification window is opened for the property to be verified. An example is shown in Figure 2.19.

The Verification window asks you to specify the options of the model-checking process. The first option concerns the model checker. The choice of a model checker is constrained by the temporal logic used to specify the property and the available plugins. By default, the GNA distribution includes plugins for the NuSMV model-checker, which makes GNA capable of verifying properties specified in CTL. Plugins for other model-checkers are available on the GNA website.

After choosing a model-checker, the corresponding graph type is automatically selected. To each of the registered model checker is associated one of the following graph types:

- The **explicit** graph is an explicit export of the state transition graph previously obtained by running a simulation for a given set of initial conditions. If the graph is too big, it is not possible to use the model-checker web server;
- The **implicit** graph is a model-checker specific model description, containing all the necessary rules for the generation of the successors states by the model checker. This avoids having to run a simulation in GNA and ensures that a smaller file is sent through the web-server connection.

If the **explicit** graph is selected, you must choose one of the previously simulated state transition graphs. If the **implicit** graph is selected, you must choose one of the previously defined initial conditions.

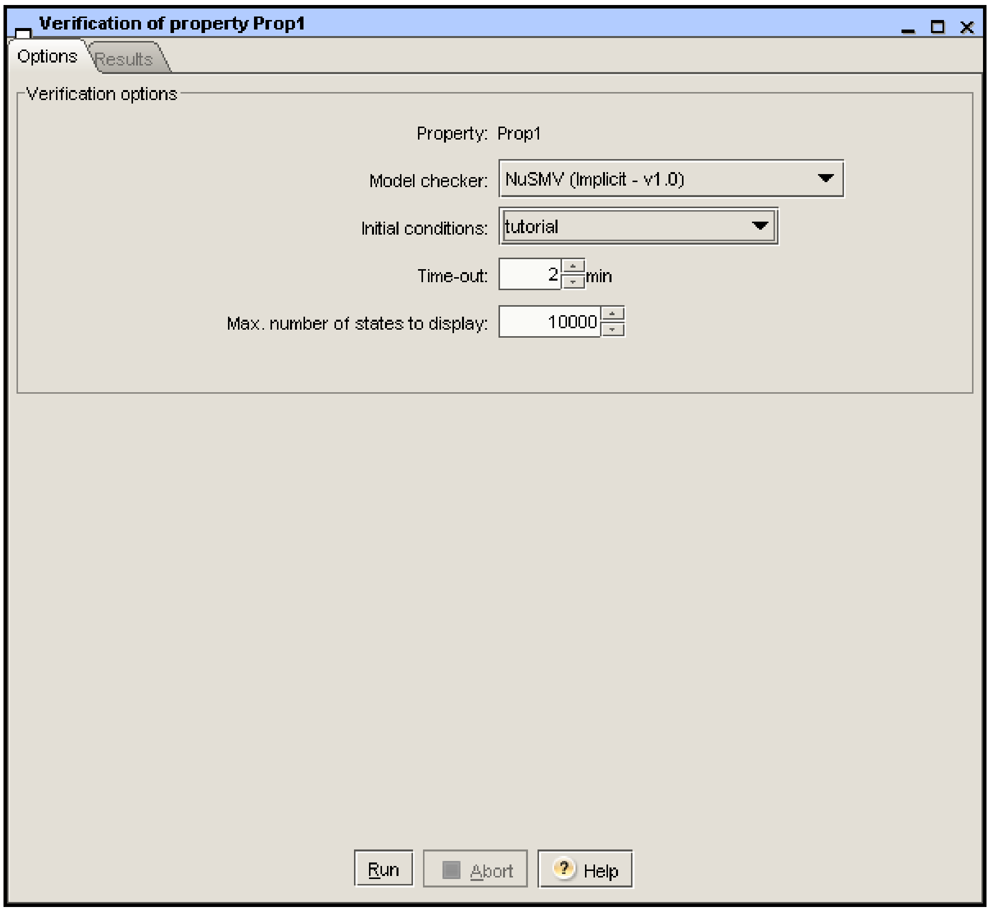
When clicking on the Run button, a verification request consisting of an (implicit or explicit) graph and a temporal-logic property is sent to the model-checker server. At any time you can abort the verification by clicking on the **Abort** button.

Once the verification is completed, a result is returned by the model-checker server. This consists of two elements, displayed in the Results tab of the frame:

- a verdict (true/false/error), representing the result of the property verification;

- a diagnostic graph, containing an example or a counterexample (if one exists). In the explicit case, this counterexample will be a subgraph of the state transition graph sent to the model checker.

If the verdict is **true**, the diagnostic graph represents a path that shows the property to be correct. On the contrary, if the verdict is **false**, the diagnostic graph represents a path that contradicts the property. Notice that some model checkers do not generate examples if the property is true (this is the case for NuSMV).



Property: Prop1
Model checker: NuSMV (Implicit - v1.0)
Initial conditions: tutorial
Time-out: 2 min
Max. number of states to display: 10000

Fig. 2.19. Verification window. Specification of the model checker options for the verification of the property defined in Figure 2.17

In case you do not wish to use the model-checker server, or if the model-checker server is not available, you can also directly test your property in the model checker of your choice. To this end, install the model checker locally on your computer and export the (explicit or implicit) graph in the appropriate format. With the default plugin in the GNA distribution, the **File** menu has an **Export** item that allows you to generate files that can be read by NuSMV (`.smv`). The next distribution will also include the possibility to export an incomplete model files, without inequality constraints. You can also generate the graph from the console using the command explained in Section 2.4. The supported formats are determined by the available plugins.

Examples of properties to be tested for several models in the GNA distribution can be consulted in the `samples` directory.

2.8. Functions of the graphical user interface

In this section, the functions of GNA are summarized. The overview is structured according to the menus in the toolbar (Figure 2.20, Figure 2.21), which is slightly different depending on whether you are editing the network or the model.

Network editor

Fig. 2.20. Menu of the graphical user interface (network editor)

File menu

This menu contains functions to load and save files.

- **New project**: Start a new project;
- **New model**: Start a new model (project without regulatory network);
- **Open project...**: Open project stored in `.gnaml` file;
- **Open recent file**: Open a recently opened file (last 10 files are listed);
- **Save project**: Save project to `.gnaml` file;
- **Save project as...**: Save project to `.gnaml` file under new name;
- **Import**
  - **Import model**: Import model stored in a `.gna` file;
  - **Import initial conditions**: Import initial conditions stored in `.dat` file;
  - **Import property...**: Import property stored in `.prop` file;
  - **Import from SBML**: Import model and initial conditions from an SBML Level 2 document;
- **Export**
  - **Export model...**: Export model to `.gna` file;
  - **Export initial conditions...**: Export initial conditions to `.dat` file;
  - **Export property...**: Export property to `.prop` file;
  - **Export to SBML**: Export model and initial conditions to an SBML document;

Some of the above import and export functions are accessible by clicking the right mouse button in the Project tree or in the corresponding windows.

Edit menu

This menu contains functions for editing the network.

- **New**
  - **Gene**: Create a new gene;
  - **Regulating entity**: Create new regulating entity;
  - **Degradation**: Create new degradation product for an entity;
  - **Complex**: Create new complex made of two or more entities;
  - **Conversion**: Create conversion link between two or more entities;

    - **Transport**: Create transport link between two entities;
    - **Activation**: Create activation link between an entity and a reaction;
    - **Inhibition**: Create inhibition link between an entity and a reaction;
    - **Catalysis**: Create catalysis link between an entity and a reaction.
- **Rename**: Rename selected entity;
- **Switch gene orientation**: Switch gene orientation from right to left to left to right, or reverse;
- **Add promoter**: Add new promoter to selected gene;
- **Delete**: Delete selected entity, reaction or regulation;
- **Preferences**: Edit preferences for GNA.

Most of the above functions are also accessible by clicking the right mouse button in the editor window or by means of the toolbar.

View menu

This menu contains functions to view and analyze the network.

- **Show whole network**: Display whole network as opposed to the reduced network;
- **Show reduced network**: Display reduced network;
- **Zoom in**: Zoom in on network;
- **Zoom out**: Zoom out of network.
- **Fit content**: Zoom in or out so that the whole network is displayed in the window
- **Reset zoom**: Zoom to 1:1 level

Help menu

- **Help**: Open help pages;
- **About GNA**: Display GNA version.

Model editor

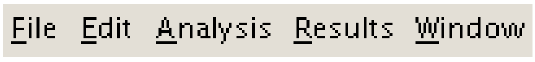


Fig. 2.21. Menu of the graphical user interface (model editor)

File menu

This menu contains functions to load and save files.

- **New project**: Start a new project;
- **New model**: Start a new model (project without regulatory network);
- **Open project...**: Open project stored in `.gnaml` file;
- **Open recent file**: Open a recently opened file (last 10 files are listed);

- **Save project**: Save project to `.gnaml` file;
- **Save project as...**: Save project to `.gnaml` file under new name;
- **Import**
  - **Import model**: Import model stored in a `.gna` file;
  - **Import initial conditions**: Import initial conditions stored in `.dat` file;
  - **Import property...**: Import property stored in `.prop` file;
  - **Import from SBML**: Import model and initial conditions from an SBML Level 2 document;
- **Export**
  - **Export model...**: Export model to `.gna` file;
  - **Export initial conditions...**: Export initial conditions to `.dat` file;
  - **Export property...**: Export property to `.prop` file;
  - **Export to SBML**: Export model and initial conditions to an SBML document;
  - **Export explicit graph to <model checker>...**: Export state transition graph in a file that can be read by the <model checker>;
  - **Export implicit graph to <model checker>...**: Export implicit graph defined by given initial conditions in a file that can be read by the <model checker>;

Some of the above import and export functions are accessible by clicking the right mouse button in the Project tree or in the corresponding windows.

Edit menu

This menu contains functions for editing the model, in particular the variables composing the model.

- **New**
  - **Variable**: Create a new variable;
  - **Initial conditions**: Create new initial conditions;
  - **Atomic proposition**: Create new atomic proposition;
  - **Property**: Create new property;
- **Edit**: Edit selected variable, initial conditions, atomic proposition or property;
- **Rename**: Rename selected variable, initial conditions, atomic proposition or property;
- **Delete**: Delete selected variable, initial conditions, atomic proposition or property;
- **View comment**: View comment for selected variable, initial conditions, atomic proposition or property;
- **Preferences**: Edit preferences for GNA.

Most of the above functions are also accessible by clicking the right mouse button in the Project window. The Variable window for a variable contains an equation editor for defining the state equation. The following key combinations can be used to edit the information in the State equation field:

- **Ctrl**+**X**: Cut and add to buffer;

- **Ctrl+C**: Copy and add to buffer;
- **Ctrl+V**: Paste buffer;
- **Ctrl+Z**: Undo;
- **Ctrl+Y**: Redo;
- **Ctrl+space**: Complete current name.

Analysis menu

This menu contains functions to analyze the model defined.

- **Search attractors**: Search phase space for qualitative steady states of the model;
- **Specify simulation**: Specify simulation parameters for given initial conditions;
- **Specify verification**: Specify verfication parameters for given property.

The above functions are also accessible by clicking the right mouse button in the Project tree.

- **Run**: Run specified attractor search, simulation or verification;
- **Abort**: Abort current running attractor search, simulation or verification.

These function are also accessible from the buttons in each Attractor search, Simulation or Verification window.

Results menu (model editor mode)

This menu contains functions to view and analyze simulation results.

- **Deselect all states**: Deselect all states in the state transition graph;
- **Filter instantaneous states**: Generate the state transition graph from which all states that are instantaneously traversed have been removed. Display this graph in a new tab;
- **Subgraph spanning selected states**: Construct the subgraph spanning the selected states in the state transition graph. Display this graph in a new tab;
- **Expand selected subgraph**: Expand the state transition graph using several levels of successors and/or predecessors of the selected states. Display this graph in a new tab;
- **Display variables in selected path**: Display the qualitative evolution of the variables in a selected path in the state transition graph;
- **Display focal sets in selected path**: Display the qualitative evolution of the values toward which the state variables locally converge;
- **Zoom in graph**: Zoom in on state transition graph;
- **Zoom out graph**: Zoom out of state transition graph.
- **Fit graph**: Zoom in or out so that the whole state transition graph is displayed in the window
- **Reset zoom**: Zoom to 1:1 level

The above functions are also accessible by clicking the right mouse button in the Transition graph window. States in the Transition graph window can be selected by means of the mouse while pressing the **Ctrl** key. Deselection is achieved by clicking and pressing the **Ctrl** key. Some shortcuts allow to easily navigate within the graph: while maintaining the **Ctrl** and **Alt** keys pressed, you can drag the graph with the mouse ; when the **Ctrl** key is pressed, the mouse wheel allows to zoom in and out.

Windows menu (model editor mode)

This menu lists the active windows and allows you to make some of them invisible.

Help menu

- **Help**: Open help pages;
- **About GNA**: Display GNA version.

Part 3. Appendices

3.1. Syntax of GNA model

Model files have the extension `.gna` and can be edited using any text editor.

Syntactic conventions

Terminals are written in **bold**, whereas non-terminals are written in *italics*. Furthermore,

- | separates alternatives;
- (...) delimitates a block composed of several alternatives;
- [...] delimitates an optional block;
- [...]* denotes a block repeated any (possibly 0) number of times;
- [...]$^{+}$ denotes a block repeated at least once.

BNF of a model

The BNF syntax of the GNA models is given below. The axiom is *model*.

model ::= [*variable*]$^{+}$

variable ::= **state-variable:** *idf* [*variable-description*]$^{+}$ | **input-variable:** *idf* [*variable-description*]$^{+}$

variable-description ::=

synthesis-parameters: *idf* [, *idf*]* | **degradation-parameters:** *idf* [, *idf*]* | **threshold-parameters:** *idf* [, *idf*]* | **box-parameter:** *idf* | **zero-parameter:** *idf* | **state-equation:** *state-equation* | **threshold-inequalities:** [*threshold-inequality* ;]$^{+}$ | **equilibrium-inequalities:** [*equilibrium-inequality* ;]$^{+}$

state-equation ::= **d/dt** *idf* **=** *synthesis-term* **-** *degradation-term*

synthesis-term ::= *idf* | [*idf*+] *idf* ***** *step-expr* [+ *idf** *step-expr*]*

degradation-term ::= *idf* ***** *idf* | **(** *idf* [+ *idf* ***** *step-expr*]$^{+}$ **)** ***** *idf*

step-expr ::= **(1 -** *step-expr* **)** | *step-expr* ***** *step-expr* | **s+(** *idf* **,** *idf* **)** | **s-(** *idf* **,** *idf* **)**

inequality-term ::= *idf* | *focal-param-expression*

focal-param-expression ::= *synthesis-param-sum* **/** *degradation-param-sum*

synthesis-param-sum ::= *idf* | **(** *idf* [+ *idf*]$^{+}$ **)**

degradation-param-sum ::= *idf* | **(** *idf* [+ *idf*]$^{+}$ **)**

idf ::= (**a** |..| **z** | **A** |..| **Z**) [**a** |..| **z** | **A** |..| **Z** | **0** |..| **9** | **_**]*

Comments can be inserted into GNA models. The syntax of comments follows the rules for comments in languages like C: multi-line comments are placed between **/*** and ***/**, while single-line comments are placed after characters **//**.

Additional constraints

In addition to the above syntax, a number of constraints must be respected:

- Some keywords are reserved and cannot be used as identifiers: **true** and **false**. In addition to this, it is recommended not to use the following identifiers if you intend to use GNA's model checking tools: **equ**, **implies**, **or**, **and**, **not**, **ef**, **af**, **eg**, **ag**, **ex**, **ax**, **nil**, **e**, **a**, **u** and **r**.
- If the variable is an input variable, only the threshold-parameters, box-parameter, zero-parameter, and threshold-inequalities blocks can appear in a variable block.
- The synthesis-parameters block must appear exactly once in a variable block. Each synthesis parameter used in the right-hand side of the state equation in some variable block must be declared in the synthesis-parameters block. In previous versions of GNA, the terminal **production-parameters:** was used instead of **synthesis-parameters:**. This alternative terminal is still accepted by the current version.
- The degradation-parameters block must appear exactly once in a variable block.
- The threshold-parameters block must appear at most once in a variable block. (It can be omitted if the state or input variable has no associated threshold parameters.) The threshold parameters in the thresholds-parameters block of a variable `x` are the thresholds at which `x` regulates other variables. That is, the threshold parameters may appear in step functions in other variable blocks.
- The zero-parameter and box-parameter blocks must appear exactly once in a variable block.
- The state-equation block must appear exactly once in a variable block. The *idf* in the left-hand side of the state equation must be the variable being described.
- The synthesis-parameters, degradation-parameters, threshold-parameters, zero-parameter and maximum-parameter blocks must have been defined before the state equation and parameter-inequalities blocks.
- In the synthesis term of the state equation, each *idf* corresponds to a synthesis parameter of the variable being described.
- In the degradation term of the state equation, the rightmost *idf* in each alternative corresponds to the variable being described, while the others correspond to degradation parameters of that variable.
- In step-function expressions, the first *idf* is the regulator variable and the second *idf* a threshold parameter of this variable.
- The parameter-inequalities block must appear exactly once in a variable block. In order to make simulation possible, these inequalities must induce a total order on the threshold, focal, zero, and maximum parameters of the variable. The focal parameters occurring in the parameter inequality are determined from the state equation by taking all possible quotients of synthesis and degradation parameters, except 0. For instance, consider the following state equation: `d/dt x_a = k_a1 + k_a2 * s+(x_b,t_b) - g_a * x_a`.The possible focal parameters are in this case `k_a1/ g_a` and `(k_a1+ k_a2)/ g_a`.The user has to assure that the set of inequality constraints specified is consistent. Notice that the terminals **threshold-inequalities:** and **equilibrium-inequalities:** (**nullcline-inequalities:**) are no longer used in GNA 6.0.
- The first inequality term in a parameter equality must be the zero parameter, while the last inequality term must be the maximum parameter.

Examples of model files can be consulted in the `samples` directory of the GNA distribution.

3.2. Syntax of GNA initial conditions

The initial conditions file specifies the initial domain(s), and hence initial qualitative state(s), from which the genetic regulatory network is to be simulated. Initial conditions files have the extension `.dat` and can be edited using any text editor.

Syntactic conventions

Terminals are written in **bold**, whereas non-terminals are written in *italics*. Furthermore,

- | separates alternatives;
- (...) delimitates a block composed of several alternatives;
- [...] delimitates an optional block;
- [...]* denotes a block repeated any (possibly 0) number of times;
- [...]$^+$ denotes a block repeated at least once.

BNF of an initial conditions file

The BNF definition of the syntax of the GNA initial conditions is shown below. The axiom is *initial-conditions*.

initial-conditions ::= **initial-conditions:** [*initial-condition* **;**]$^+$

initial-condition ::= *idf* **<** *idf* | *idf* **>** *idf* |*idf* **<=** *idf* |*idf* **>=** *idf*

idf ::= (**a** |..| **z** | **A** |..| **Z**) [**a** |..|**z** | **A** |..| **Z** | **0** |..| **9** | **_**]*

Comments can be inserted into the files. The syntax of comments follows the rules for comments in languages like C: multi-line comments are placed between **/*** and ***/**, while single-line comments are placed after characters **//**.

Additional constraints

In the above syntax, for each of the alternative *initial-condition* expressions, the first *idf* is a state or input variable and the second *idf* either the box parameter, the zero parameter, or one of the threshold parameters of the variable. Initial conditions come in pairs for each variable, the first one giving its lower bound, the second one its upper bound. For the lower bound, the operator must be **>=**, if the right-hand side parameter is a zero parameter, and **>**, if it is a threshold parameter. For the upper bound, the operator must be **<=**, if the right-hand side parameter is a box parameter, and **<**, if it is a threshold parameter. The order of the threshold parameters, zero parameters, and box parameters given in the model file must be respected. Examples of initial conditions files can be consulted in the `samples` directory of the GNA distribution.

3.3. Syntax of CTRL property

The property file specifies a biological property enconded with a CTRL temporal logic formula, and the corresponding Atomic Propositions (describing restrictions on a given state) that are referred to inside this formula. Property files have the extension `.prop` and can be edited using any text editor.

Syntactic conventions

Terminals are written in **bold**, whereas non-terminals are written in *italics*. Furthermore,

- | separates alternatives;
- (...) delimitates a block composed of several alternatives;
- [...] delimitates an optional block;
- [...]* denotes a block repeated any (possibly 0) number of times;
- [...]$^+$ denotes a block repeated at least once.

BNF of a property

The BNF syntax of a property is given below. The axiom is *property*.

property ::= [*atomic-propositions*]* **property:** *idf* **formula:** *property-formula* **;**

atomic-propositions ::= **atomic-proposition:** *idf* [*subatomic-proposition*]$^+$

subatomic-proposition ::=

domain: [*idf inequality-term boundary-parameter* ;]$^+$ |

focal-set: [*idf inequality-term boundary-parameter* ;]$^+$ |

derivative: [**d/dt** *idf* (**Undefined** | **Zero** | **Positive** | **Negative** | **All**) ;]$^+$ |

state: [(**IsSteadyState;** | **IsInitialState;** | **IsInSCC;** | **IsInTerminalCycle;**)]$^+$

inequality-term ::= **>** | **>=** | **<** | **<=**

property-formula ::= [^;]*

boundary-parameter ::= *idf* | *focal-param-expression*

focal-param-expression ::= *synthesis-param-sum* **/** *degradation-param-sum*

synthesis-param-sum ::= *idf* | **(** *idf [+ idf]*$^+$ **)**

degradation-param-sum ::= *idf* | **(** *idf [+ idf]*$^+$ **)**

idf ::= **(a** |..| **z** | **A** |..| **Z)** [**a** |..| **z** | **A** |..| **Z** | **0** |..| **9** | **_**]*

Comments can be inserted into the files. The syntax of comments follows the rules for comments in languages like C: multi-line comments are placed between **/*** and ***/**, while single-line comments are placed after characters **//**.

Additional constraints

In addition to the above syntax, a number of constraints must be respected:

- The property formula is not checked when loading the property and the corresponding atomic propositions. For the grammar, the formula consists of everything until the semi colon. The syntax should be verified when trying to check the validity of the property.
- Atomic-propositions used inside a property formula must be defined. Their identifier is be the identifier appearing on the project tree.
- For the subatomic propositions domain and focal-set, the identifier *idf* represents a variable name that must be declared in the model. For each variable, the lower and upper bounds should be defined. If absent the lower bound will be *zero_var* and the upper bound *max_var*.
- For the subatomic proposition derivative, the identifier *idf* represents a variable name that must be declared in the model.
- The property block must appear exactly once in a variable block.

BNF of a property formula

The BNF syntax of a property formula is given below. The axiom is *specification*.

Some of these operators are boolean operators (**nil**, **not**, **true**, **false**, **and**, **or**, **equ**, **implies**), CTL and CTRL operators (**EF**, **AF**, **EX**, **AX**, **EG**, **AG**, **A**, **E**, **U**, **R**, **EF@**, **AF@**, **EG-|**, **AG-|**) and regular operators (**.**, **|**, *****,**+**). The CTRL grammar describing how these operators are articulated is shown below.

```
specification ::= equ-state-formula ;
equ-state-formula ::= equ-state-formula <equ> implies-state-formula
                      | implies-state-formula totototo;
implies-state-formula ::= implies-state-formula <implies> or-state-formula
                          | or-state-formula ;
or-state-formula ::= or-state-formula <or> and-state-formula
                     | and-state-formula ;
and-state-formula ::= and-state-formula <and> unary-state-formula
                      | unary-state-formula ;
unary-state-formula ::= <not> unary-state-formula
                        | temporal-operator unary-state-formula
                        | basic-state-formula ;
temporal-operator ::= <EF> <{> regular-formula <}>
                      | <AF> <{> regular-formula <}>
                      | <EG> <{> regular-formula <}>
                      | <AG> <{> regular-formula <}>
                      | ctl-unary-operator ;
ctl-unary-operator ::= <EX>
                      | <AX>
                      | <EF>
                      | <AF>
                      | <EG>
                      | <AG>;
basic-state-formula ::= <EF> <{> regular-formula <}> <@>
                        | <AF> <{> regular-formula <}> <@>
                        | <EG> <{> regular-formula <}> <-|>
                        | <AG> <{> regular-formula <}> <-|>
                        | atomic-formula
                        | <(> state-formula <)>
                        | ctl-binary-formula ;
ctl-binary-formula ::= <E> <[> state-formula <U> state-formula <]>
                      | <A> <[> state-formula <U> state-formula <]>
                      | <E> <[> state-formula <R> state-formula <]>
                      | <A> <[> state-formula <R>" state-formula <]> ;
atomic-formula ::= <TRUE>
                   | <FALSE>
                   | < " >[tmp ]*< " >
                   | < ' >[ tmp ]*< ' >
                   | identifier ;
regular-formula ::= concatenation-regular-formula ;
concatenation-regular-formula ::= concatenation-regular-formula <.> choice-regular-formula
                                  | choice-regular-formula ;
choice-regular-formula ::= choice-regular-formula <|> unary-regular-formula
                           | unary-regular-formula ;
unary-regular-formula ::= unary-regular-formula <*>
                          | unary-regular-formula <+>
                          | basic-regular-formula
                          | state-formula ;
basic-regular-formula ::= <NIL>
                          | <(> concatenation-regular-formula <.> choice-regular-formula <)>
                          | <(> choice-regular-formula <|> unary-regular-formula <)>
                          | <(> unary-regular-formula <*> <)>
                          | <(> unary-regular-formula <+> <)>
                          | <(> basic-regular-formula <)>;
tmp ::= [A-Z_0-9a-z ]
        | < \" >
        | < \' > ;
identifier ::= (<a> |..| <z> | <A> |..| <Z>) [<a> |..| <z> | <A> |..| <Z> | <0> |..| <9> | <_> ]* ;
```

3.4. CUP PARSER GENERATOR COPYRIGHT NOTICE, LICENSE AND DISCLAIMER

3.5. JGRAPH COPYRIGHT NOTICE, LICENSE, AND DISCLAIMER

3.6. SAT4J LICENSE

3.7. NuSMV LICENSE