

UNIVERSITY OF ATHENS
M.Sc. in BIOINFORMATICS
ATHENS 2006

Assignment in
COMPLEX ADAPTIVE SYSTEMS

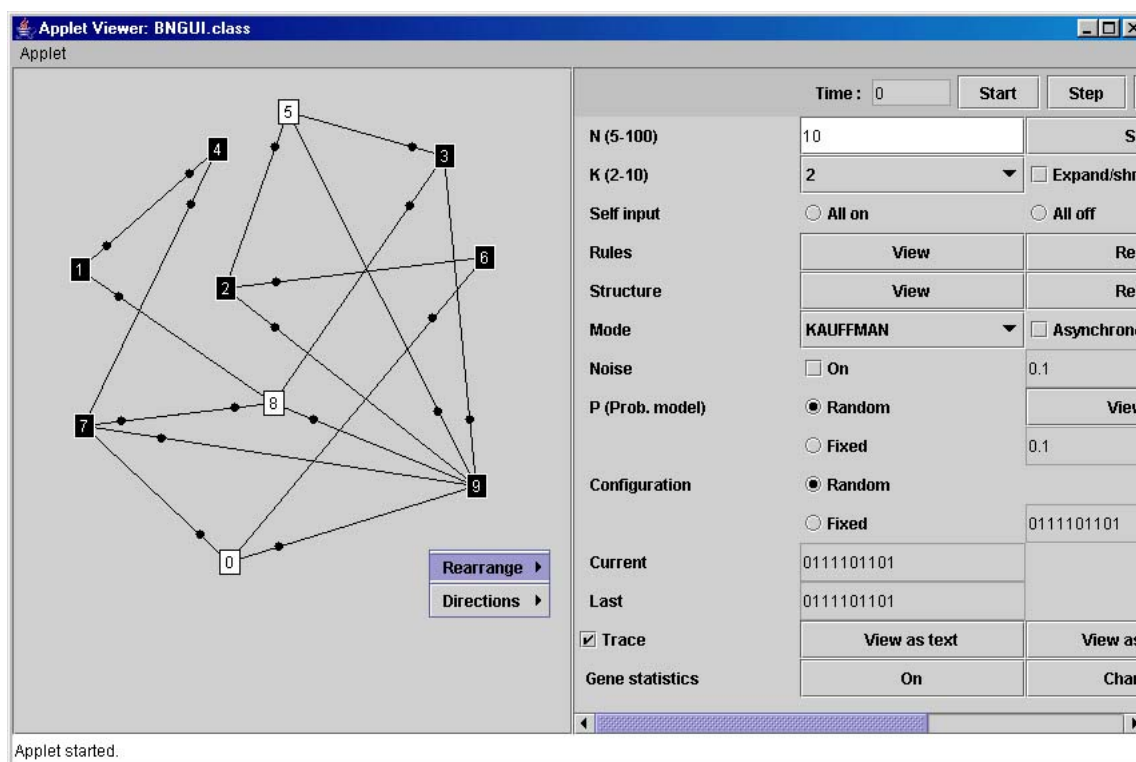
Topic
Boolean networks

Elpida Tzafestas

1 Objective of the project

The objective is experimentation with various boolean network models, Kauffman (N -K) and variants. More precisely, the student has to carry out attractor studies, perturbation studies, execution timing studies (synchronous, asynchronous), as well as behavioral studies of other gene types. See detailed instructions in section 3.

2 Implementation of the testbed

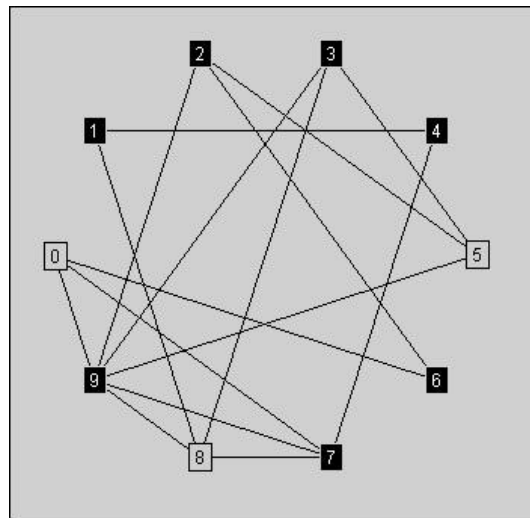


2.1 User's manual

The applet's screen shows a gray area on the left where the simulated network lies. Genes are visualized in black or white, depending on their value (true/1 or false/0, respectively). The user may drag a gene to another position on the screen to better visualize the network. For visualization purposes, the user may also automatically rearrange the genes on the screen by using the popup menu that is shown in the above snapshot. The "rearrange" option allows to reposition all genes either randomly or regularly around a circle (as vertices of a N-gon). The "directions" option allows to manipulate the visibility parameters for the directions of the gene-gene connections.

By default, the network edges are represented with a small ball near the destination- gene of the connection, for example in the above network the gene '2' takes input from genes '6' and

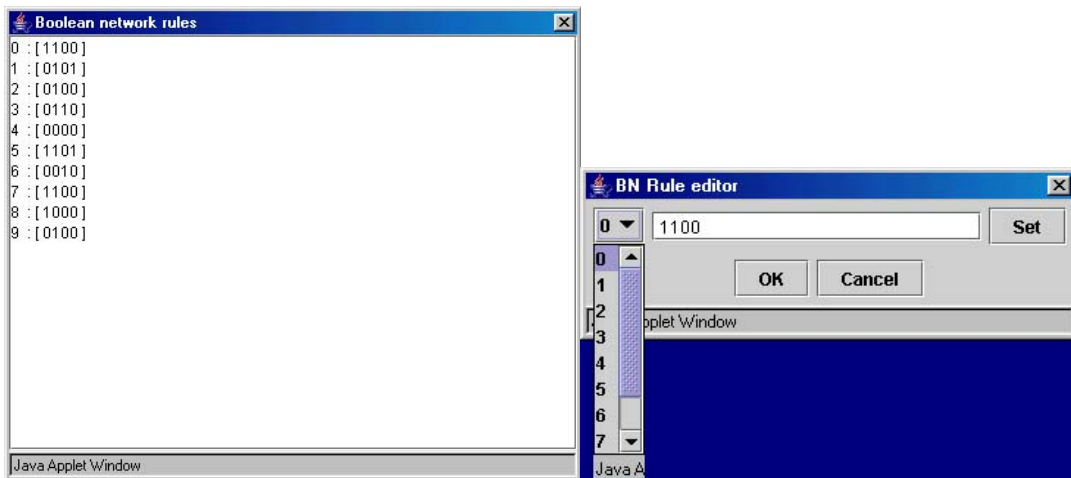
'9'. The following figure shows the same network as above rearranged cyclically and with all connection directions invisible (hidden).



The right side of the applet includes a number of simulation control components.

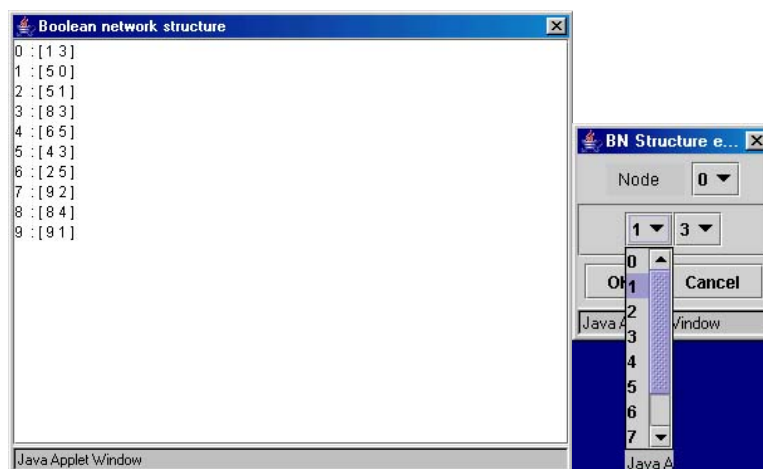
- **The simulation clock.**
- **Three simulation control buttons.**
 The **Start/Stop** button starts/stops the simulation and in every cycle executes the whole network. The **Step** button advances the simulation one cycle further. Finally, the **Restart** button restarts the simulation, i.e. reinitializes the clock to 0 and resets the network and the statistics observers.
- **Definition of network N (number of genes).**
 It shows the current N of the network. The user may change it and press the **Set** button to reinitialize the network.
- **Definition of network K (connectivity).**
 It shows the current K of the network. If the user selects another value in the selector ("combo box", from 1 to 10), then the network is automatically reinitialized. When the **Expand/shrink** checkbox is selected, then any network reinitialization with a different N and/or K will reuse the current structure and expand or shrink appropriately (in a random way) to reach the new values of N and K.
- **Self-input.**
 It indicates whether a gene can take itself as input. This feature may be **On** (all genes take themselves as input), **Off** (no gene takes itself as input) or **Random** (all genes take themselves as input).
- **Three buttons for control and visualization of the network rules.**
 The **View** button opens a text dialog that shows the rules of the network : for each gene, a series of bits appear that are the outputs of the gene for each logical (boolean) combination of the inputs of the gene (see following figure left). The **Reset** button

resets randomly all rules in the network. Finally, the **Edit** button opens a specialized rule editor (see following figure right).



- **Three buttons for control and visualization of the network structure.**

The **View** button opens a text dialog that shows the structure of the network : for each (numbered) target gene, a series of numbers appear that are the numbers of the genes that are the inputs of the target gene (see following figure left). The **Reset** button resets randomly the structure of the network (all the connections). Finally, the **Edit** button opens a specialized structure editor (see following figure right).



- **Behavioral model (mode).**

Initially one of the following three :

- **KAUFFMAN model.**
- **PROBABILISTIC model.**
- **(Indicative) model with memory (MEMORIFUL).**

The code of the three models is given next.

```
public void execute(boolean inp[])
{
    int i, n = 0, k = inp.length;
    double diff;

    lastState = state;
    for (i=0;i<k;i++)
        n += (inp[i] ? powers2(k-i-1) : 0);

    switch (mode)
    {
        case KAUFFMAN:
        default:
            state = rule[n];
            break;
        case PROBABILISTIC:
            state = (Math.random() < P)
                ? (!rule[n]) : rule[n];
            break;
        case MEMORIFUL:
            if (rule[n])
                r += (rr*(1 - r));
            else
                r -= (rr*r);
            state = (r >= 0.5);
            break;
    }
}
```

- **Checkbox “asynchronous”.**

If it is selected, the genes execute asynchronously and in random order, otherwise they execute synchronously. The code for the two execution modes is given below.

```
public void execute()
{
    if (genes.isEmpty()) return;
    if (asynchronous) executeAsynchronous();
    else executeSynchronous();
}

public void executeAsynchronous()
{
    int i;

    // Create ordered = shuffled list of genes
    int ordered[] = new int[N];
    for (i=0;i<N;i++) ordered[i] = i;
    ordered = shuffle(ordered);

    for (i=0;i<N;i++)
    {
        int the_i = ordered[i];
        boolean inp[] = new boolean[K];
```

```

        for (int j=0;j<K;j++)
        {
            inp[j] = ((BooleanGene)genes.elementAt(
                inputs[the_i][j])).state();
            if (isNoisy)
                inp[j] = (Math.random()<noise)
                    ? (!inp[j]) : inp[j];
        }
        ((BooleanGene)genes.elementAt(the_i)).execute(inp);
    }
}

public void executeSynchronous()
{
    int i;
    boolean output[] = new boolean[N];
    for (i=0;i<N;i++)
        output[i] = ((BooleanGene)genes.elementAt(i)).
            state();

    for (i=0;i<N;i++)
    {
        boolean inp[] = new boolean[K];
        for (int j=0;j<K;j++)
        {
            inp[j] = output[inputs[i][j]];
            if (isNoisy)
                inp[j] = (Math.random()<noise)
                    ? (!inp[j]) : inp[j];
        }
        ((BooleanGene)genes.elementAt(i)).execute(inp);
    }
}

```

- **Noise controls.**

Checkbox **On** shows if there is noise in the network, whereas the text field indicates the amount (percentage) of noise present. The user may change this value and press the **Set** button, in which case the current value of noise will change.

- **Node probabilities controls for the PROBABILISTIC gene model.**

The **Random** choice is used to define that the genes will have different (random) selection probabilities, whereas the **Fixed** choice is used to define that the genes will have the same selection probability, that will be defined in the right text field and confirmed with the **Set** button. Finally, the **View All** button is used to open a text dialog that shows all node probabilities.

- **Current network configuration controls.**

The **Random** choice is used to define that the genes will have random boolean states, whereas the **Fixed** choice is used to define that the genes will have the particular configuration defined in the right text field and confirmed with the **Set** button.

- **Current.**

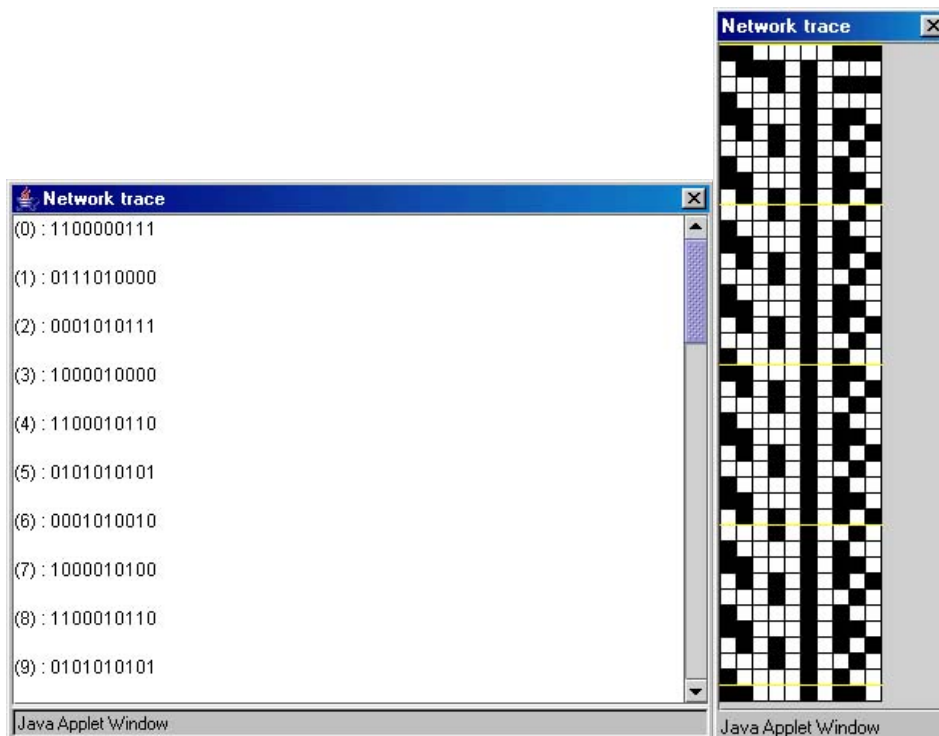
The current state of the network.

- **Last.**

The last (just before the current) state of the network.

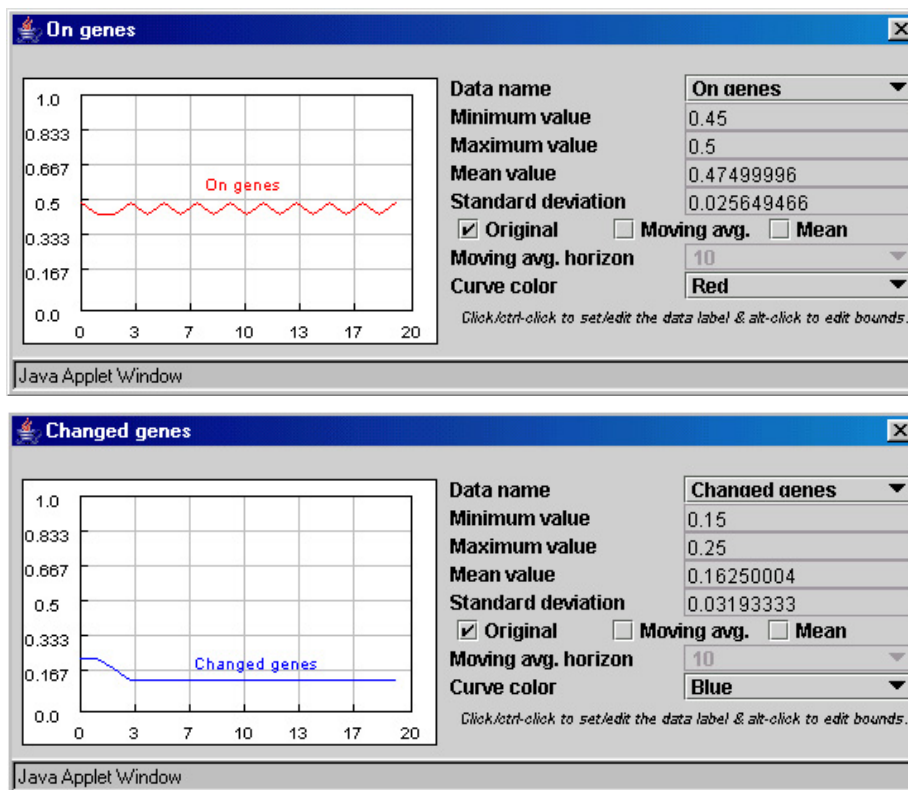
- **System trace (or history) controls.**

The **View as text** button is used to open a text dialog that shows the trace of the network : for every moment in time, the state of the network is given (see next figure, left). The **View as image** button opens a visual dialog that shows the trace of the network as a “sheet” : row-*i* of the sheet from top to bottom shows the state of the network at time *i*, with the active genes (with boolean value = true or 1) showing as black, and the inactive ones as white. The order of the genes within the row is from left to right (see next figure, right).



- **Network statistics buttons.**

The **On** button is used to visualize the statistics of the percentage of active genes in every cycle, whereas the **Changed** button is used to visualize the statistics of the percentage of genes that changed value in every cycle. See following figure.



The above features of the testbed allow the user to experiment systematically with several boolean network models, in configurations or conditions of his own choice.

3 Instructions

The testbed includes four types of classes :

- **User interface classes.** Those comprise first the BNGUI class the implements the main system interface (applet) and the manipulation features for the simulated boolean network. Those comprise also the classes TextDialog (that is used to visualize textual data: visualization of network rules, structure and trace), BNRuleEditor, BNStructureEditor and TraceViewer (that are used respectively for the manipulation and editing of the network rules and structure and the visualization of the network trace as an image). The user does not need to know the internals of these classes, but rather it uses them as such.
- **Simulation class.** This is the BNSimulation class, that implements the simulation itself and connects the interface with the boolean network model (this class is responsible for running the simulation, collecting and visualizing the statistics and updating the interface). The user does not need to know the internals of this class, but rather it uses it as such.

- **Modeling classes.** Those are the classes BooleanNetwork, that implements the boolean network itself (gene connections, time execution models of the network, network structure and rules manipulation) and BooleanGene that implements the gene-component of the above network (that may follow one of many different behavioral models).
- **Auxiliary classes.** Those are the classes Observer, History, Curve, CurveViewer, ListModifierDialog, ListModifierListener and TextAreaDialog, that implement the statistical data recording as a time series that may be translated into a curve and visualized as a graph. The user does not need to know the internals of these classes, but rather it uses them as such. Only the classfiles are given for these classes.

What to do

1. Find all attractors for a small network of your choice (8-12 genes).

Hint. Try all possible input states (2^N states for a network of N genes, for example 256 states for N=8). Exploit the fact that a state lying in the basin of attraction of an attractor will necessarily lead to the same attractor, to avoid checking some network states. You can initially enumerate all possible states and then systematically erase all states that you check through simulation or that belong to the attractors that appear.

Deliverable. Network description text including a list of attractors (as a series of network states, for example 10010-11100-11001) and a list of states leading to each one of them. For example, for a network of 256 states with 3 attractors, each of the 256 states has to appear in one and only one of the three lists that correspond to the attractors.

2. For the above network find at least one structure variant (addition, deletion or modification of a connection) that creates a new attractor. If possible, find also at least one structure variant that removes one of the existing attractors.

Deliverable. Description text for the modified network and the new attractor together with at least one initial state leading to the new attractor. For attractor removal, you will either need to present systematically all the attractors of the modified network as before or to provide a clever explanation.

3. For the same network find at least one gene rule variant that creates a new attractor. If possible, find also at least one gene rule variant that removes one of the existing attractors.

Deliverable. Description text for the modified network and the new attractor together with at least one initial state leading to the new attractor. For attractor removal, you will either need to present systematically all the attractors of the modified network as before or to provide a clever explanation.

4. For the same network compare attractors in synchronous and asynchronous execution.

Deliverable. Description text for attractors and state lists as above.

5. **Optionally (programming).** Implement a new gene model from the ones given below or similar :

- **Probabilistic as far as the rule is concerned.** For example, according to the Shmulevich model, a gene includes more than one rules and probabilistically selects one of them for execution, e.g. it selects one out of three rules with equal probabilities 33.33% each.
- **Probabilistic as far as the connections are concerned.** It is similar to the above, but the gene selects in every cycle K from its $>K$ real connections. For example, it includes $2K$ connections and selects in every cycle half of them to use for execution.
- **Gene with memory.** Think of what kind of memory could be meaningful in a gene network. Draw inspiration from the simple MEMORIFUL model that is given.

Compare the behavior of the original KAUFFMAN model with the behavior of your model on the basis of at least one criterion of your choice. For example, a (conceptually) good general criterion would be tolerance to perturbations, where small state variations do not induce major changes to the network attractor.

Deliverable. Description text as above and the modified BooleanGene.java file.

3.1 *Description of the BooleanNetwork class*

BooleanNetwork class defines the following primary data :

- **int N**
- **int K**
The number of genes of the network and its connectivity (number of connections per gene), according to the original N-K theory.
- **int[][] inputs**
The connectivity matrix of the network. For the gene 'i' (i from 0 to N-1) the value inputs[i][j] gives the input gene (again from 0 to N-1) at connection 'j' (j from 0 to K-1).
- **double noise**
The percentage of noise of the network (probability with which the regular output of the gene is reversed/negated).
- **boolean isNoisy**
Variable showing whether there is noise in the network.
- **boolean asynchronous**
Variable showing whether the network executes synchronously or asynchronously (when the variable takes the value false or true, respectively).
- **int selfInput**
Variable showing which of the following three self input models is used.

- **final static int SELF_INPUT_ON**
- **final static int SELF_INPUT_OFF**
- **final static int SELF_INPUT_RANDOM**

The three self input models for genes (always take themselves as inputs, never take themselves as input, randomly take themselves as inputs).

The most important methods implemented by the BooleanNetwork class are the following :

- **public void setN(int,boolean)**
- **public void setK(int,boolean)**
- **public void shrinkStructure(int,int,int)**
Reinitialization of the network with a new N or K passed as an argument (int) and with a boolean option that shows whether the network will shrink/expand as necessary or be reinitialized anew. For shrinking to a smaller N, the method **shrinkStructure** is used with arguments the old N and K and the new N.
- **public void resetStructure(int,int,int,int)**
General reinitialization method from an initial N-K pair to a new N-K pair.
- **public void execute()**
Execution of the network synchronously or asynchronously, according to the value of the **asynchronous** variable. It uses the following two methods.
- **public void executeAsynchronous()**
- **public void executeSynchronous()**
Execution of the network asynchronously or synchronously.
- **public float getOnGenes()**
- **public float getChangedGenes()**
Computation of the percentage of active genes and the percentage of genes that changed state in the last cycle.
- **public String getCurrentStateString(int)**
- **public String getLastStateString(int)**
Translation of the current or previous network state to a string (binary : 01001...). The int argument stands for the number of characters per String line.
- **public String getStructureString()**
- **public String getRulesString()**
Translation of the structure or the rules of the network to a String. In the first case, it is a series of numbers (the numbers of the input genes of each gene), whereas in the second case it is a series of rules for each gene given in the form of strings (binary : 01001...).

- **public void resetRules()**
Random reinitialization of all the network rules.
- **public void setRules(String[])**
Reinitialization of all the network rules on the given array of strings (one rule per gene).
- **public void setStructure(int[][])**
Reinitialization of the network structure on the given 2D array of ints (it initializes the `inputs[][]` variable).
- **public void setRandomConfiguration()**
- **public void setFixedConfiguration(boolean[])**
Reinitialization of the network state randomly or on the given array of booleans (one boolean per gene), respectively.
- **public void editRules()**
- **public void editStructure()**
Invocation of the editor for the rules and the structure of the network, respectively.

3.2 *Description of the BooleanGene class*

BooleanGene class defines the following data :

- **boolean state**
- **boolean lastState**
The current and previous state of the gene (true or false).
- **boolean[] rule**
The gene's rule in the form of a truth table. The numbering of the inputs follows the order with which the inputs are given in the network structure description (variable `inputs[][]`), for example for the gene with inputs '6' and '3' (in that order) and rule `xyzw`, 'x' corresponds to gene-'6'=false and gene-'3'=false, y corresponds to gene-'6'=false and gene-'3'=true, z corresponds to gene-'6'=true and gene-'3'=false and w corresponds to gene-'6'=true and gene-'3'=true.
- **int mode**
The behavioral model of the gene. One of the following.
- **final static int KAUFFMAN**
- **final static int PROBABILISTIC**
- **final static int MEMORIFUL**
The three gene models described above (pages 4-5).

- **double r**
A variable indicating a state transition rate that is used by the MEMORIFUL gene model.
- **History stateHistory**
The “history ”of the gene’s state (series of 0 and 1).

The most important methods implemented by the BooleanGene class are the following :

- **void setRandomRule()**
Random initialization of the gene’s rule.
- **void setRule(String)**
Initialization of the gene’s rule on the String given as an argument.
- **String ruleString(int)**
Translation of the gene’s rule to a String (binary : 01001...). The connectivity K is given as an argument.
- **void setState(boolean)**
- **void setRandomState()**
Initialization of the gene’s state on the value that is given as an argument or on a random value.
- **void execute()**
It implements the gene’s behavioral models.

References

- R.J.Bagley, L.Glass (1996). Counting and classifying attractors in high dimensional dynamical systems, *Journal of Theoretical Biology*, 183(1996):269-284.
- L.G.Volkert, M.Conrad (1998). The role of weak interactions in biological systems: the dual dynamics model, *Journal of Theoretical Biology*, 193(1998):287-306.
- W.R.Stark, W.H.Hughes (1996). Asynchronous, irregular automata nets: the path not taken, *Biosystems*, 55(2000):107-117.

See also ...

Discrete Dynamics Lab, by Andy Wünsche
<http://www.ddlab.com/>