Package ‘BoolNet’

January 20, 2025

Type Package

Title Construction, Simulation and Analysis of Boolean Networks

Version 2.1.9

Date 2023-10-02

Imports igraph (>= 0.6), XML

Description Functions to reconstruct, generate, and simulate synchronous, asynchronous, probabilistic, and temporal Boolean networks. Provides also functions to analyze and visualize attractors in Boolean networks <doi:10.1093/bioinformatics/btq124>.

License Artistic-2.0

LazyLoad yes

ByteCompile TRUE

Encoding UTF-8

NeedsCompilation yes

Author Christoph Müssel [aut],
Martin Hopfensitz [aut],
Dao Zhou [aut],
Hans A. Kestler [aut, cre],
Armin Biere [ctb] (contributed PicoSAT code),
Troy D. Hanson [ctb] (contributed uthash macros)

Maintainer Hans A. Kestler <hans.kestler@uni-ulm.de>

Repository CRAN

Date/Publication 2023-10-02 14:30:04 UTC

Contents

attractorsToLaTeX 3
binarizeTimeSeries 4
cellcycle 7
chooseNetwork 8
examplePBN 9
fixGenes 10

generateRandomNKNetwork . . . 11
generateState . . . 14
generateTimeSeries . . . 15
generationFunctions . . . 17
getAttractors . . . 18
getAttractorSequence . . . 23
getBasinOfAttraction . . . 24
getPathToAttractor . . . 25
getStateSummary . . . 27
getTransitionProbabilities . . . 28
getTransitionTable . . . 29
igf . . . 30
loadBioTapestry . . . 30
loadNetwork . . . 32
loadSBML . . . 37
markovSimulation . . . 38
perturbNetwork . . . 40
perturbTrajectories . . . 41
plotAttractors . . . 43
plotNetworkWiring . . . 46
plotPBNTransitions . . . 47
plotSequence . . . 49
plotStateGraph . . . 52
print.AttractorInfo . . . 54
print.BooleanNetwork . . . 54
print.MarkovSimulation . . . 55
print.ProbabilisticBooleanNetwork . . . 56
print.SymbolicSimulation . . . 57
print.TransitionTable . . . 58
reconstructNetwork . . . 59
saveNetwork . . . 61
sequenceToLaTeX . . . 63
simplifyNetwork . . . 65
simulateSymbolicModel . . . 66
stateTransition . . . 69
symbolicToTruthTable . . . 71
testNetworkProperties . . . 72
toPajek . . . 77
toSBML . . . 78
truthTableToSymbolic . . . 79
yeastTimeSeries . . . 80

Index **82**

attractorsToLaTeX *Create LaTeX state table of attractors*

Description

Exports state tables of attractors (corresponding to the plot generated by `plotAttractors` with `mode="table"`) to a LaTeX document.

Usage

```
attractorsToLaTeX(attractorInfo,
                  subset,
                  title = "",
                  grouping = list(),
                  plotFixed = TRUE,
                  onColor = "[gray]{0.9}",
                  offColor = "[gray]{0.6}",
                  reverse = FALSE,
                  file = "attractors.tex")
```

Arguments

`attractorInfo` An object of class `AttractorInfo`, as returned by `getAttractors`, or an object of class `SymbolicSimulation`, as returned by `simulateSymbolicModel`.

`subset` An subset of attractors to be exported. This is a vector of attractor indices in `attractorInfo`.

`grouping` An optional structure to form groups of genes in the plot. This is a list with the following elements:

- **class** A vector of names for the groups. These names will be printed in the region belonging to the group in the table.
- **index** A list with the same length as `class`. Each element is a vector of gene indices belonging to the group.

`title` An optional title for the plot

`plotFixed` If this is true, genes with fixed values are included in the plot. Otherwise, these genes are not shown.

`onColor` An optional color value for the 1/ON values in the table. Defaults to dark grey.

`offColor` An optional color value for the 0/OFF values in the table. Defaults to light grey.

`reverse` Specifies the order of the genes in the plot. By default, the first gene is placed in the first row of the table. If `reverse=TRUE`, the first gene in the network is placed in the bottom row of the table.

`file` The file to which the LaTeX document is written. Defaults to "attractors.tex".

Details

This function creates LaTeX tables that visualize the states of synchronous attractors. Asynchronous attractors are ignored. Attractors in `attractorInfo` are first grouped by length. Then, a LaTeX table environment is created for each attractor length (i.e. one plot with all attractors consisting of 1 state, one plot with all attractors consisting of 2 states, etc.). The output file does not contain a document header and requires the inclusion of the packages `tabularx` and `colortbl`. The tables have the genes in the rows and the states of the attractors in the columns. If not specified otherwise, cells of the table are light grey for 0/OFF values and dark grey for 1/ON values. If `grouping` is set, the genes are rearranged according to the indices in the group, horizontal separation lines are plotted between the groups, and the group names are printed.

Value

A list of matrices corresponding to the plots is returned. Each of these matrices has the genes in the rows and the states of the attractors in the columns.

See Also

`getAttractors`, `plotAttractors`, `sequenceToLaTeX`, `plotSequence`

Examples

```
## Not run:
# load example data
data(cellcycle)

# get attractors
attractors <- getAttractors(cellcycle)

# output LaTeX document
attractorsToLaTeX(attractors, file="attractors.tex")

## End(Not run)
```

---

binarizeTimeSeries *Binarize a set of real-valued time series*

Description

Binarizes a set of real-valued time series using k-means clustering, edge detection, or scan statistics.

Usage

```
binarizeTimeSeries(measurements,
                   method = c("kmeans","edgeDetector","scanStatistic"),
                   nstart = 100,
                   iter.max = 1000,
                   edge = c("firstEdge","maxEdge"),
```

```
                    scaling = 1,
                    windowSize = 0.25,
                    sign.level = 0.1,
                    dropInsignificant = FALSE)
```

Arguments

`measurements` A list of matrices, each corresponding to one time series. Each row of these matrices contains real-valued measurements for one gene on a time line, i. e. column `i+1` contains the successor states of column `i+1`. The genes must be the same for all matrices in the list.

`method` The employed binarization technique. "kmeans" uses k-means clustering for binarization. "edgeDetector" searches for a large gradient in the sorted measurements. "scanStatistic" searches for accumulations in the measurements. See Details for descriptions of the techniques.

`nstart` If `method="kmeans"`, this is the number of restarts for k-means. See `kmeans` for details.

`iter.max` If `method="kmeans"`, the maximum number of iterations for k-means. See `kmeans` for details.

`edge` If `method="edgeDetector"`, this decides which of the edges is used as a threshold for binarization. If set to "firstEdge",the binarization threshold is the first combination of two successive sorted values whose difference exceeds a predefined value (average gradient * `scaling`). The parameter `scaling` can be used to adjust this value.

If set to "maxEdge", the binarization threshold is the position of the edge with the overall highest gradient.

`scaling` If `method="edgeDetector"` and `edge="firstEdge"`, this holds the scaling factor used for adjustment of the average gradient.

`windowSize` If `method="scanStatistic"`, this specifies the size of the scanning window (see Details). The size is given as a fraction of the whole range of input values for a gene. Default is 0.25.

`sign.level` If `method="scanStatistic"`, the significance level used for the scan statistic (see Details).

`dropInsignificant`
If this is set to true, genes whose binarizations are insignificant in the scan statistic (see Details) are removed from the binarized time series. Otherwise, a warning is printed if such genes exist.

Details

This method supports three binarization techniques:

k-means clustering For each gene, k-means clusterings are performed to determine a good separation of groups. The values belonging to the cluster with the smaller centroid are set to 0, and the values belonging to the greater centroid are set to 1.

Edge detector This approach first sorts the measurements for each gene. In the sorted measurements, the algorithm searches for differences of two successive values that satisfy a predefined condition: If the "firstEdge" method was chosen, the pair of values whose difference exceeds the scaled average gradient of all values is chosen and used as maximum and minimum value of the two groups. If the "maxEdge" method was chosen, the largest difference between two successive values is taken. For details, see Shmulevich et al.

Scan statistic The scan statistic assumes that the measurements for each gene are uniformly and independently distributed independently over a certain range. The scan statistic shifts a scanning window across the data and decides for each window position whether there is an unusual accumulation of data points based on an approximated test statistic (see Glaz et al.). The window with the smallest p-value is remembered. The boundaries of this window form two thresholds, from which the value that results in more balanced groups is taken for binarization. Depending on the supplied significance level, gene binarizations are rated according to the p-value of the chosen window.

Value

Returns a list with the following elements:

`binarizedMeasurements`
: A list of matrices with the same structure as `measurements` containing the binarized time series measurements

`reject`
: If `method="scanStatistic"`, a Boolean vector indicating for each gene whether the scan statistic algorithm was able to find a significant binarization window (FALSE) or not (TRUE). Rejected genes should probably be excluded from the data.

`thresholds`
: The thresholds used for binarization

References

I. Shmulevich and W. Zhang (2002), Binary analysis and optimization-based normalization of gene expression data. Bioinformatics 18(4):555–565.

J. Glaz, J. Naus, S. Wallenstein (2001), Scan Statistics. New York: Springer.

See Also

`reconstructNetwork`

Examples

```
# load test data
data(yeastTimeSeries)

# perform binarization with k-means
bin <- binarizeTimeSeries(yeastTimeSeries)
print(bin)

# perform binarization with scan statistic
# - will find and remove 2 insignificant genes!
```

```
bin <- binarizeTimeSeries(yeastTimeSeries, method="scanStatistic",
                          dropInsignificant=TRUE, sign.level=0.2)
print(bin)

# perform binarization with edge detector
bin <- binarizeTimeSeries(yeastTimeSeries, method="edgeDetector")
print(bin)

# reconstruct a network from the data
reconstructed <- reconstructNetwork(bin$binarizedMeasurements,
                                    method="bestfit", maxK=4)
print(reconstructed)
```

`cellcycle` *Mammalian cell cycle network*

Description

The mammalian cell cycle network as described by Faure et al.

Usage

```
data(cellcycle)
```

Details

The data consists of a variable `cellcycle` of class `BooleanNetwork` with 10 genes describing the four phases of the mammalian cell cycle. The network has one steady-state attractor. Furthermore, it has one synchronous attractor with 7 states and one asynchronous complex/loose attractor with 112 states. The class `BooleanNetwork` is described in more detail in `loadNetwork`.

Source

A. Faure, A. Naldi, C. Chaouiya and D. Thieffry (2006), Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle. Bioinformatics 22(14):e124–e131.

Examples

```
data(cellcycle)

# the network is stored in a variable called 'cellcycle'
print(cellcycle)
```

chooseNetwork *Extract a single Boolean network from a probabilistic Boolean network*

Description

Creates a BooleanNetwork object with exactly one function per gene by extracting a specified set of transition functions from a ProbabilisticBooleanNetwork or BooleanNetworkCollection object.

Usage

```
chooseNetwork(probabilisticNetwork,
              functionIndices,
              dontCareValues=NULL,
              readableFunctions=FALSE)
```

Arguments

probabilisticNetwork
: A ProbabilisticBooleanNetwork or BooleanNetworkCollection object as returned by reconstructNetwork or loadNetwork

functionIndices
: A vector of function indices with one entry for each gene

dontCareValues
: If probabilisticNetwork is of class BooleanNetworkCollection, this specifies the values to fill in for "don't care" (*) values in the truth tables of the transition functions. This is a list containing one vector of Boolean values for each gene. The lengths of the vectors must coincide with the numbers of "don't care" values in the functions.

readableFunctions
: If probabilisticNetwork is of class BooleanNetworkCollection, the string representations of the transition functions must be refreshed after filling in values for the "don't care" entries. This parameter specifies if readable DNF representations of the transition function truth tables are generated and displayed when the network is printed. If set to FALSE, the truth table result column is displayed. If set to "canonical", a canonical Disjunctive Normal Form is generated from each truth table. If set to "short", the canonical DNF is minimized by joining terms (which can be time-consuming for functions with many inputs). If set to TRUE, a short DNF is generated for functions with up to 12 inputs, and a canonical DNF is generated for functions with more than 12 inputs.

Value

Returns an object of class BooleanNetwork consisting of the transition functions whose indices were specified in functionIndices. The class BooleanNetwork is described in more detail in loadNetwork.

Constant genes are automatically fixed (e.g. knocked-out or over-expressed). This means that they are always set to the constant value, and states with the complementary value are not considered in transition tables etc. If you would like to change this behaviour, use fixGenes to reset the fixing.

See Also

reconstructNetwork, loadNetwork

Examples

```
  ## Not run:
# load example data
data(examplePBN)

# extract a unique network
# - always use the first function
net <- chooseNetwork(examplePBN, rep(1, length(examplePBN$genes)))

# get attractors from this network
print(getAttractors(net))

## End(Not run)
```

---

examplePBN *An artificial probabilistic Boolean network*

Description

An artificial probabilistic Boolean network example introduced by Shmulevich et al.

Usage

```
data(examplePBN)
```

Details

This artificial network is introduced by Shmulevich et al. for a step-by-step description of their Markov chain algorithm. It is included as a general example for a probabilistic Boolean network. The network consists of 3 genes, where gene 1 and gene 3 have two alternative transition functions, and gene 1 has a unique transition function.

Source

I. Shmulevich, E. R. Dougherty, S. Kim, W. Zhang (2002), Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks. Bioinformatics 18(2):261–274.

Examples

```
data(examplePBN)

# the network is stored in a variable called 'examplePBN'
print(examplePBN)
```

`fixGenes` *Simulate knocked-out or over-expressed genes*

Description

Simulates knocked-out or over-expressed genes by fixing the values of genes to 0 or 1, or turn off knock-out or over-expression of genes.

Usage

```
fixGenes(network, fixIndices, values)
```

Arguments

`network`	The original network of class `BooleanNetwork`, `SymbolicBooleanNetwork` or `ProbabilisticBooleanNetwork` containing the genes to be fixed
`fixIndices`	A vector of names or indices of the genes to be fixed
`values`	Either one single value, or a vector with the same length as `fixIndices`. For each gene, a value of 1 means that the gene is always turned on (over-expressed), a value of 0 means that the gene is always turned off (knocked-out), and a value of -1 means that the gene is not fixed.

Value

Depending on the input, an object of class `BooleanNetwork`, `SymbolicBooleanNetwork` or `ProbabilisticBooleanNetwork` containing the fixed genes is returned. These classes are described in more detail in `loadNetwork`.

See Also

`loadNetwork`

Examples

```
## Not run:
# load example data
data(cellcycle)

# knock out gene CycD (index 1)
net <- fixGenes(cellcycle, 1, 0)
# or
net <- fixGenes(cellcycle, "CycD", 0)
```

```
# get attractors by exhaustive search
attractors <- getAttractors(net)

print(attractors)

## End(Not run)
```

---

generateRandomNKNetwork

Generate a random N-K Boolean network

Description

Generates a random N-K Boolean network (see Kauffman, 1969) using different configurations for the topology, the linkage, and the functions.

Usage

```
generateRandomNKNetwork(n, k,
                        topology = c("fixed", "homogeneous", "scale_free"),
                        linkage = c("uniform", "lattice"),
                        functionGeneration = c("uniform", "biased"),
                        validationFunction, failureIterations=10000,
                        simplify = FALSE, noIrrelevantGenes=TRUE,
                        readableFunctions = FALSE,
                        d_lattice = 1, zeroBias = 0.5,
                        gamma = 2.5, approx_cutoff = 100)
```

Arguments

`n`	The total number of genes in the network
`k`	If this is a single number, this is either the maximum number of genes in the input of a transition function (for `topology="fixed"` and `topology="scale_free"`) or the mean number of genes in the input of a function (for `topology="homogeneous"`). If `topology="fixed"`, this can also be a vector with `n` elements specifying the number of input genes for each gene separately.
`topology`	If set to "fixed", all transition functions of the network depend on exactly `k` input genes (unless there are irrelevant input genes to be removed if `simplify=TRUE` and `noIrrelevantGenes=FALSE`). If set to "homogeneous", the number of input genes is drawn independently at random from a Poisson distribution with lambda = k. If set to "scale_free", the number of input genes of each function is drawn from a Zeta distribution with parameter `gamma`.

`linkage`
: If this parameter is "uniform", the actual input genes are drawn uniformly at random from the total `k` genes.

  If set to "lattice", only genes from the neighbourhood `(i - d_lattice * k_i):(i + d_lattice * k_i)` are taken, which means that all genes are dependent from other genes in the direct neighbourhood.

`functionGeneration`
: This parameter specifies how the truth tables of the transition functions are generated. If set to "uniform", the truth table result column of the function is filled uniformly at random with 0 and 1. If set to "biased", a bias is introduced, where the probability of drawing a 0 is determined by the parameter `zeroBias`.

  As a third option, `functionGeneration` can be set to a user-defined function that generates the truth tables. This function must have a single parameter `input` that is supplied with a vector of input gene indices. It must return a binary vector of size `2^length(input)` corresponding to the result column of the truth table. For the generation of canalyzing and nested canalyzing functions that are often assumed to be biologically plausible, the generation functions `generateCanalyzing` and `generateNestedCanalyzing` are included in **BoolNet**.

`validationFunction`
: An optional function that restricts the generated Boolean functions to certain classes. This can be used if no explicit generation function can be specified in `functionGeneration`, but it is nevertheless possible to check whether a generated function belongs to that class or not. The function should have two input parameter `input` and `func` that receive a candidate function. `input` is a matrix of 0/1 integer values specifying the input part of the truth table of the candidate function, with the input genes in the columns. Each of the 2^k rows of `input` (where k is the number of input genes) corresponds to one entry of `func`, which is an integer vector of 0/1 values corresponding to the output of the candidate function. The validation function should return `TRUE` if the candidate function is accepted or `FALSE` if it is rejected.

`failureIterations`
: The maximum number of iterations the generator tries to generate a function that is accepted by `validationFunction` before it gives up and throws an error. Defaults to 10000.

`simplify`
: If this is true, `simplifyNetwork` is called to simplify the gene transition functions after the perturbation. This removes irrelevant input genes. Should not be used together with `noIrrelevantGenes=TRUE`, as this automatically generates a network that cannot be simplified any further. Defaults to FALSE.

`noIrrelevantGenes`
: If set to true, gene transition functions are not allowed to contain irrelevant genes, i.e. the functions have exactly the number of input genes determined by the `topology` method. This means that the network cannot be simplified any further, and `simplify` should be turned off. The default value is TRUE.

`readableFunctions`
: This parameter specifies if readable DNF representations of the transition function truth tables are generated and displayed when the network is printed. If set to FALSE, the truth table result column is displayed. If set to "canonical", a

canonical Disjunctive Normal Form is generated from each truth table. If set to "short", the canonical DNF is minimized by joining terms (which can be time-consuming for functions with many inputs). If set to TRUE, a short DNF is generated for functions with up to 12 inputs, and a canonical DNF is generated for functions with more than 12 inputs.

`d_lattice` The dimension parameter for the lattice if `linkage="lattice"`. Defaults to 1.

`zeroBias` The bias parameter for biased functions for `functionGeneration="biased"`. Defaults to 0.5 (no bias).

`gamma` The Gamma parameter of the Zeta distribution for `topology="scale_free"`. Default is 2.5.

`approx_cutoff` This parameter is only used with `topology="scale_free"`. It sets the number of iterations in the sum used to approximate the Riemann Zeta function. Defaults to 100.

Details

The function supports a high number of different configurations to generate random networks. Several of the parameters are only needed for special configurations. The generated networks have different structural properties. Refer to the literature for more details.

Constant genes are automatically fixed (e.g. knocked-out or over-expressed). This means that they are always set to the constant value, and states with the complementary value are not considered in transition tables etc. If you would like to change this behaviour, use `fixGenes` to reset the fixing.

Value

An object of class `BooleanNetwork` containing the generated random network. The class `BooleanNetwork` is described in more detail in `loadNetwork`.

References

S. A. Kauffman (1969), Metabolic stability and epigenesis in randomly constructed nets. J. Theor. Biol. 22:437–467.

S. A. Kauffman (1993), The Origins of Order. Oxford University Press.

M. Aldana (2003), Boolean dynamics of networks with scale-free topology. Physica D 185: 45–66.

M. Aldana and S. Coppersmith and L. P. Kadanoff (2003), Boolean dynamics with random coupling. In E. Kaplan, J. E. Marsden and K. R. Sreenivasan (editors): Perspectives and Problems in Nonlinear Science, Springer.

See Also

`perturbNetwork`,`loadNetwork`, `simplifyNetwork`, `fixGenes`

Examples

```
## Not run:
# generate different random networks
net1 <- generateRandomNKNetwork(n=10, k=10,
```

```
                                topology="scale_free",
                                linkage="uniform",
                                functionGeneration="uniform",
                                noIrrelevantGenes=FALSE,
                                simplify=TRUE)

net2 <- generateRandomNKNetwork(n=10, k=3,
                                topology="homogeneous",
                                linkage="lattice",
                                functionGeneration="uniform",
                                d_lattice=1.5,
                                simplify=TRUE)

net3 <- generateRandomNKNetwork(n=10, k=2,
                                topology="fixed",
                                linkage="uniform",
                                functionGeneration="biased",
                                noIrrelevantGenes=FALSE,
                                zeroBias=0.6)

# get attractors
print(getAttractors(net1))
print(getAttractors(net2))
print(getAttractors(net3))

## End(Not run)
```

`generateState` *Generate a state vector from single gene values*

Description

This function provides a simple interface to generate full state vectors by specifying only the genes of interest. For example, only those genes that are active can be specified, while the others are set to a default value.

Usage

```
generateState(network,
              specs,
              default = 0)
```

Arguments

`network`	An network of class `BooleanNetwork`, `SymbolicBooleanNetwork` or `ProbabilisticBooleanNetwork` for which a state is generated.
`specs`	A named vector or list specifying the genes to be set. Here, the names of the elements correspond to the gene names, and the elements correspond to the gene values. The function can also generate a matrix of states if the elements of `specs` are vectors of values (of the same length).

`default` The default value used for the unspecified genes (usually 0).

Value

Returns a full state vector with one entry for each gene of the network, or a matrix with one state in each row if `specs` contains vectors of state values.

See Also

`getAttractors`, `simulateSymbolicModel`, `stateTransition`

Examples

```
## Not run:
# load cell cycle network
data(cellcycle)

# generate a state in which only CycD and CycA are active
state <- generateState(cellcycle, c("CycD"=1, "CycA"=1))
print(state)

# use the state as a start state for attractor search
print(getAttractors(cellcycle, startStates=list(state)))

## End(Not run)
```

`generateTimeSeries` *Generate time series from a network*

Description

Generates time series by simulating successive state transitions from random start states. In addition, the resulting matrices can be perturbed by Gaussian noise.

Usage

```
generateTimeSeries(network,
                   numSeries,
                   numMeasurements,
                   type = c("synchronous","asynchronous","probabilistic"),
                   geneProbabilities,
                   perturbations = 0,
                   noiseLevel = 0)
```

Arguments

`network` An object of class `BooleanNetwork` or `SymbolicBooleanNetwork` that contains the network for which time series are generated

`numSeries` The number of random start states used to generate successive series of states, that is, the number of time series matrices to generate

`numMeasurements` The number of states in each of the time series matrices. The first state of each time series is the randomly generated start state. The remaining `numMeasurements - 1` states are obtained by successive state transitions.

`type` The type of state transitions to be performed (see `stateTransition`)

`geneProbabilities` An optional vector of probabilities for the genes if `type="asynchronous"`. By default, each gene has the same probability to be chosen for the next state transition. These probabilities can be modified by supplying a vector of probabilities for the genes which sums up to one.

`perturbations` If this argument has a value greater than 0, artificial perturbation experiments are generated. That is, `perturbations` genes in each time series are knocked out or overexpressed artificially using the `fixGenes` function.

`noiseLevel` If this is non-zero, it specifies the standard deviation of the Gaussian noise which is added to all entries of the time series matrices. By default, no noise is added to the time series.

Value

A list of matrices, each corresponding to one time series. Each row of these matrices contains measurements for one gene on a time line, i. e. column `i+1` contains the successor states of column `i+1`. If `noiseLevel` is non-zero, the matrices contain real values, otherwise they contain only 0 and 1.

If `perturbations>0`, the result list contains an additional matrix `perturbations` specifying the artificial perturbations applied to the different time series. This matrix has `numSeries` columns and one row for each gene in the network. A matrix entry is 0 for a knock-out of the corresponding gene in the corresponding time series, 1 for overexpression, and NA for no perturbation.

The result format is compatible with the input parameters of `binarizeTimeSeries` and `reconstructNetwork`.

See Also

`stateTransition`, `binarizeTimeSeries`, `reconstructNetwork`

Examples

```
## Not run:
# generate noisy time series from the cell cycle network
data(cellcycle)
ts <- generateTimeSeries(cellcycle, numSeries=50, numMeasurements=10, noiseLevel=0.1)

# binarize the noisy time series
```

```
bin <- binarizeTimeSeries(ts, method="kmeans")$binarizedMeasurements

# reconstruct the network
print(reconstructNetwork(bin, method="bestfit"))

## End(Not run)
```

---

generationFunctions *Generation functions for biologically relevant function classes*

Description

These generation functions randomly generate canalyzing or nested canalyzing Boolean functions. These functions are usually not called directly, but are supplied to the `functionGeneration` parameter of `generateRandomNKNetwork`.

Usage

```
generateCanalyzing(input)
generateNestedCanalyzing(input)
```

Arguments

`input`	A vector of input gene indices for the Boolean function

Value

A binary vector corresponding to the result column of the truth table that represents the canalyzing/nested canalyzing function.

References

S. Kauffman and C. Peterson and B. Samuelsson and C. Troein (2004), Genetic networks with canalyzing Boolean rules are always stable. PNAS 101(49):7102–17107.

See Also

`generateRandomNKNetwork`

Examples

```
## Not run:
  # generate a random network with canalyzing functions
  net1 <- generateRandomNKNetwork(n=10, k=5,
                                  functionGeneration="generateCanalyzing")
  print(net1)

  # generate a random network with nested canalyzing functions
  net2 <- generateRandomNKNetwork(n=10, k=5,
```

```
                                    functionGeneration="generateNestedCanalyzing")
  print(net2)

## End(Not run)
```

---

getAttractors *Identify attractors in a Boolean network*

Description

Identifies attractors (cycles) in a supplied Boolean network using synchronous or asynchronous state transitions

Usage

```
getAttractors(network,
              type = c("synchronous","asynchronous"),
              method = c("exhaustive",
                         "sat.exhaustive",
                         "sat.restricted",
                         "random",
                         "chosen"),
              startStates = list(),
              genesON = c(), genesOFF = c(),
              canonical = TRUE,
              randomChainLength = 10000,
              avoidSelfLoops = TRUE,
              geneProbabilities = NULL,
              maxAttractorLength = Inf,
              returnTable = TRUE)
```

Arguments

network — A network structure of class `BooleanNetwork` or `SymbolicBooleanNetwork`. These networks can be read from files by `loadNetwork`, generated by `generateRandomNKNetwork`, or reconstructed by `reconstructNetwork`.

type — If `type="synchronous"`, synchronous state transitions are used, i.e. all genes are updated at the same time. Synchronous attractor search can be performed in an exhaustive manner or using a heuristic that starts from predefined states. For symbolic networks, only synchronous updates are possible.

If `type="asynchronous"`, asynchronous state transitions are performed, i.e. one (randomly chosen) gene is updated in each transition. Steady-state attractors are the same in asynchronous and synchronous networks, but the asynchronous search is also able to identify complex/loose attractors. Asynchronous search relies on a heuristic algorithm that starts from predefined states.

See Details for more information on the algorithms.

`method` The search method to be used. If "exhaustive", attractors are identified by exhaustive state space search, i.e. by calculating the sucessors of all 2^n states (where n is the number of genes that are not set to a fixed value). This kind of search is only available for synchronous attractor search, and the maximum number of genes allowed for exhaustive search is 29. Apart from the attractors, this method generates the full state transition graph.

If `method` is "sat.exhaustive" or "sat.restricted", attractors are identified using algorithms based on the satisfiability problem. This search type is also restricted to synchronous networks. It can be used to identify attractors in much larger networks than with `method="exhaustive"`, but does not return the state transition graph. For `method="sat.exhaustive"`, an exhaustive attractor search is performed, while `method="sat.restricted"` only searches for attractors of a specified maximum length `maxAttractorLength`.

If `method` is "random", `startStates` is interpreted as an integer value specifying the number of states to be generated randomly. The algorithm is then initialized with these random states and identifies the attractors to which these states lead.

If `method` is "chosen", `startStates` is interpreted as a list of binary vectors, each specifying one input state. Each vector must have `length(network$genes)` elements with 0 or 1 values. The algorithm identifies the attractors to which the supplied states lead. If `network` is of class `SymbolicBooleanNetwork` and makes use of more than one predecessor state, this can also be a list of matrices with the genes in the columns and multiple predecessor states in the rows.

If `method` is not supplied, the desired method is inferred from the type of `startStates`. By default, if neither `method` nor `startStates` are provided, an exhaustive search is performed.

`startStates` The value of `startStates` depends on the chosen method. See `method` for more details.

`genesON` A vector of genes whose values are fixed to 1, which reduces the complexity of the search. This is equivalent to a preceding call of `fixGenes`.

`genesOFF` A vector of genes whose values are fixed to 0, which reduces the complexity of the search. This is equivalent to a preceding call of `fixGenes`.

`canonical` If set to true, the states in the attractors are rearranged such that the state whose binary encoding makes up the smallest number is the first element of the vector. This ensures that attractors found by different heuristic runs of `getAttractors` are comparable, as the cycles may have been entered at different states in different runs of the algorithm.

`randomChainLength`
If `type="asynchronous"`, this parameter specifies the number of random transitions performed by the search to enter a potential attractor (see Details). Defaults to 10000.

`avoidSelfLoops` If `type="asynchronous"` and `avoidSelfLoops=TRUE`, the asynchronous attractor search only enters self loops (i.e. transitions that result in the same state) if none of the possible transitions can leave the state. This results in attractors with fewer edges. Otherwise, self loops are included in the attractors. By default, self loops are avoided.

`geneProbabilities`
: If `type="asynchronous"`, this allows to specify probabilities for the genes. By default, each gene has the same probability to be chosen for the next state transition. You can supply a vector of probabilities for each of the genes which sums up to one.

`maxAttractorLength`
: If `method="sat.restricted"`, this required parameter specifies the maximum size of attractors (i.e. the number of states in the loop) to be searched. For `method="sat.exhaustive"`, this parameter is optional and specifies the maximum attractor length for the initial length-restricted search phase that is performed to speed up the subsequent exhaustive search. In this case, changing this value might bring performance benefits, but does not change the results.

`returnTable`
: Specifies whether a transition table is included in the returned `AttractorInfo` structure. If `type="asynchronous"` or `method="sat"`, this parameter is ignored, as the corresponding algorithms never return a transition table.

Details

Depending on the type of network and the chosen parameters, different search algorithms are started.

For `BooleanNetwork` networks, there are three different modes of attractor search:

Exhaustive synchronous state space search In this mode, synchronous state transitions are carried out from each of the possible states until an attractor is reached. This identifies all synchronous attractors.

Heuristic synchronous state space search In contrast to exhaustive synchronous search, only a subset of the possible states is used. From these states, synchronous transitions are carried out until an attractor is reached. This subset is specified in `startStates`.

Exhaustive synchronous SAT-based search Here, the attractor search problem is formulated as a satisfiability problem and solved using Armin Biere's PicoSAT solver. The algorithm is a variant of the method by Dubrova and Teslenko which searches for a satisfying assignment of a chain constructed by unfolding the transition relation. Depending on `maxAttractorLength`, it additionally applies an initial size-restricted SAT-based search (see below) to increase overall search speed. This method is suitable for larger networks of up to several hundreds of genes and exhaustively identifies all attractors in these networks. In contrast to the state space search, it does not construct and return a state transition table.

Size-restricted synchronous SAT-based search Here, the SAT solver directly looks for satisfying assignments for loops of a specific size. This may be more efficient for large networks and is guaranteed to find all attractors that comprise up to `maxAttractorLength` states (e.g. all steady states for `maxAttractorLength=1`) , but does not find any larger attractors. As for the exhaustive SAT-based method, no transition table is returned.

Heuristic asynchronous search This algorithm uses asynchronous state transitions and is able to identify steady-state and complex/loose attractors (see Harvey and Bossomaier, Garg et al.). These attractors are sets of states from which all possible asynchronous transitions lead into a state that is member of the set as well. The heuristic algorithm does the following for each of the input state specified by `startStates`:

1. Perform randomChainLength random asynchronous transitions. After these transitions, the network state is expected to be located in an attractor with a high probability.
2. Calculate the forward reachable set of the current state. Then, compare this set to the forward reachable set of all states in the set. If all sets are equal, a complex attractor is found.

For SymbolicBooleanNetwork networks, getAttractors is simply a wrapper for simulateSymbolicModel with preset parameters.

Printing the return value of getAttractors using print visualizes the identified attractors.

Value

For BooleanNetwork networks, this returns a list of class AttractorInfo with components

attractors
: A list of attractors. Each element is a 2-element list with the following components:

involvedStates A matrix containing the states that make up the attractor. Each column represents one state. The entries are decimal numbers that internally represent the states of the genes. The number of rows depends on the number of genes in the network: The first 32 genes are encoded in the first row, genes 33-64 are encoded in the second row, etc.

initialStates This element is only available if an asynchronous search was carried out and this is a complex attractor. In this case, it holds the encoded start states of the transitions in the complex attractor

nextStates This element is only available if an asynchronous search was carried out and this is a complex attractor. In this case, it holds the encoded successor states of the transitions in the complex attractor

basinSize The number of states in the basin of attraction. Details on the states in the basin can be retrieved via getBasinOfAttraction.

stateInfo
: A summary structure of class BooleanStateInfo containing information on the transition table. It has the following components:

initialStates This element is only available if type="synchronous", method is "random" or "chosen", and returnTable=TRUE. This is a matrix describing the initial states that lead to the states in table after a state transition. If method is "exhaustive", this component is NULL. In this case, the initial states can be inferred, as all states are used. The format of the matrix is described in involvedStates.

table This element is only available if type="synchronous" and returnTable=TRUE. It holds result vector of the transition table as a matrix with one column for each state. These are encoded bit vectors in decimal numbers as described above.

attractorAssignment This element is only available if type="synchronous" and returnTable=TRUE. It contains a vector that corresponds to the entries in table and describes the attractor index in attractors to which successive transitions from the described state finally lead.

stepsToAttractor This element is only available if type="synchronous" and returnTable=TRUE. Referring to attractorAssignment, this is the number of transitions needed to reach the attractor.

genes A list of names of the genes in `network`.

fixedGenes Specifies the fixed genes as in the `fixed` component of `network`.

The structure supports pretty printing using the `print` method.

For `SymbolicBooleanNetwork` networks, `getAttractors` redirects the call to `simulateSymbolicModel` and returns an object of class `SymbolicSimulation` containing the attractors and (if `returnTable=TRUE`) the transition graph.

References

S. A. Kauffman (1969), Metabolic stability and epigenesis in randomly constructed nets. J. Theor. Biol. 22:437–467.

S. A. Kauffman (1993), The Origins of Order. Oxford University Press.

I. Harvey, T. Bossomaier (1997), Time out of joint: Attractors in asynchronous random Boolean networks. Proc. of the Fourth European Conference on Artificial Life, 67–75.

A. Garg, A. Di Cara, I. Xenarios, L. Mendoza, G. De Micheli (2008), Synchronous versus asynchronous modeling of gene regulatory networks. Bioinformatics 24(17):1917–1925.

E. Dubrova, M. Teslenko (2011), A SAT-based algorithm for finding attractors in synchronous Boolean networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics 8(5):1393–1399.

A. Biere (2008), PicoSAT Essentials. Journal on Satisfiability, Boolean Modeling and Computation 4:75-97.

See Also

`loadNetwork`, `generateRandomNKNetwork`, `simulateSymbolicModel`, `plotAttractors`, `attractorsToLaTeX`, `getTransitionTable`, `getBasinOfAttraction`, `getAttractorSequence`, `getStateSummary`, `getPathToAttractor`, `fixGenes`, `generateState`

Examples

```
## Not run:
# load example data
data(cellcycle)

# get all synchronous attractors by exhaustive search
attractors <- getAttractors(cellcycle)

# plot attractors side by side
par(mfrow=c(2, length(attractors$attractors)))
plotAttractors(attractors)

# finds the synchronous attractor with 7 states
attractors <- getAttractors(cellcycle, method="chosen",
                            startStates=list(rep(1, length(cellcycle$genes))))
plotAttractors(attractors)

# finds the attractor with 1 state
attractors <- getAttractors(cellcycle, method="chosen",
```

```
                              startStates=list(rep(0, length(cellcycle$genes))))
plotAttractors(attractors)

# also finds the attractor with 1 state by restricting the attractor length
attractors <- getAttractors(cellcycle, method="sat.restricted",
                              maxAttractorLength=1)
plotAttractors(attractors)

# identifies asynchronous attractors
attractors <- getAttractors(cellcycle, type="asynchronous", startStates=100)
plotAttractors(attractors, mode="graph")

## End(Not run)
```

`getAttractorSequence` *Decode the state sequence of a synchronous attractor*

Description

Obtains the sequence of states belonging to a single synchronous attractor from the encoded data in an `AttractorInfo` structure or in a `SymbolicSimulation` structure.

Usage

```
getAttractorSequence(attractorInfo, attractorNo)
```

Arguments

`attractorInfo`	An object of class `AttractorInfo`, as returned by `getAttractors`, or of class `SymbolicSimulation`, as returned by `simulateSymbolicModel`. As the transition table information in this structure is required, `getAttractors` must be called in synchronous mode and with `returnTable` set to TRUE. Similarly, `simulateSymbolicModel` must be called with `returnGraph=TRUE`.
`attractorNo`	The index of the attractor in `attractorInfo` whose state sequence should be obtained

Value

Returns a data frame with the genes in the columns. The rows are the successive states of the attractor. The successor state of the last state (i.e. the last row) is the first state (i.e. the first row).

See Also

`getAttractors`, `simulateSymbolicModel`, `getPathToAttractor`, `plotSequence`, `sequenceToLaTeX`

Examples

```
## Not run:
# load example data
data(cellcycle)

# get attractors
attractors <- getAttractors(cellcycle)

# print basin of 7-state attractor
print(getAttractorSequence(attractors, 2))

## End(Not run)
```

---

getBasinOfAttraction *Get states in basin of attraction*

Description

Extracts information on all states in the basin of a supplied attractor

Usage

```
getBasinOfAttraction(attractorInfo, attractorNo)
```

Arguments

attractorInfo	An object of class AttractorInfo, as returned by getAttractors, or of class SymbolicSimulation, as returned by simulateSymbolicModel. As the transition table information in this structure is required, getAttractors must be called in synchronous mode and with returnTable set to TRUE. Similarly, simulateSymbolicModel must be called with returnGraph=TRUE.
attractorNo	The index of the attractor in attractorInfo whose basin should be identified

Details

The function outputs a transition table containing only the states that are contained in the basin of attraction, and displays additional information on these states. If attractorInfo is the result of an exhaustive synchronous attractor search, the complete basin of attraction is returned. If attractorInfo is the result of a heuristic synchronous search, there is no guarantee that the complete basin of attraction is returned, as only the calculated states are included. Asynchronous search results are not supported, as no transition table is calculated.

Value

Returns a generic dataframe of the class TransitionTable. For n genes, the first n columns code for the original state, i.e. each column represents the value of one gene. The next n columns code for the successive state after a transition. The column attractorAssignment indicates the attractor to the state is assigned (in this case, attractorNo). If this information is available, the

column stepsToAttractor indicates how many transitions are needed from the original state to the attractor. The TransitionTable class supports pretty printing using the print method.

See Also

getStateSummary, getTransitionTable, getAttractors, simulateSymbolicModel

Examples

```
## Not run:
# load example data
data(cellcycle)

# get attractors
attractors <- getAttractors(cellcycle)

# print basin of first attractor
print(getBasinOfAttraction(attractors, 1))

## End(Not run)
```

---

getPathToAttractor *Get state transitions between a state and its attractor*

Description

Lists the states in the path from a specified state to the corresponding synchronous attractor.

Usage

```
getPathToAttractor(network,
                   state,
                   includeAttractorStates = c("all","first","none"))
```

Arguments

network	Either a network structure of class BooleanNetwork or SymbolicBooleanNetwork , or an attractor search result of class AttractorInfo. In the former case, a synchronous attractor search starting from state is conducted. In the latter case, network must be the result of a call to getAttractors with returnTable=TRUE, and its transition table must include state.
state	A binary vector with exactly one entry per gene in the network. If network is of class SymbolicBooleanNetwork and makes use of more than one predecessor state, this can also be a matrix with the genes in the columns and multiple predecessor states in the rows.

`includeAttractorStates`
: Specifies whether the actual attractor states are included in the resulting table or not. If `includeAttractorStates = "all"` (which is the default behaviour), the sequence ends when the attractor was traversed once. If `includeAttractorStates = "first"`, only the first state of attractor is added to the sequence. This corresponds to the behaviour prior to **BoolNet** version 1.5. If includeAttractorStates = "none", the sequence ends with the last non-attractor state. In this case, the sequence can be empty if the start state is an attractor state.

Value

Returns a data frame with the genes in the columns. The rows are the successive states from `state` to the the corresponding attractor. Depending on `includeAttractorStates`, attractor states are included or not. The data frame has an attribute `attractor` specifying the indices of the states that belong to the attractor. If `includeAttractorStates` is `"first"` or `"none"`, these indices may correspond to states that are not included in the sequence itself. This attribute is used by `plotSequence` to highlight the attractor states.

See Also

`getAttractors`, `simulateSymbolicModel`, `getTransitionTable`, `getBasinOfAttraction`, `plotSequence`, `attributes`

Examples

```
## Not run:
# load example network
data(cellcycle)

# get path from a state to its attractor
# include all attractor states
path <- getPathToAttractor(cellcycle, rep(1,10),
                           includeAttractorStates="all")
print(path)

# include only the first attractor state
path <- getPathToAttractor(cellcycle, rep(1,10),
                           includeAttractorStates="first")
print(path)

# exclude attractor states
path <- getPathToAttractor(cellcycle, rep(1,10),
                           includeAttractorStates="none")
print(path)

## End(Not run)
```

getStateSummary — *Retrieve summary information on a state*

Description

Returns information on the supplied state, i.e. the successive state after a transition, the (synchronous) attractor to which the state leads, and the distance to this attractor.

Usage

```
getStateSummary(attractorInfo, state)
```

Arguments

attractorInfo	An object of class AttractorInfo, as returned by getAttractors, or of class SymbolicSimulation, as returned by simulateSymbolicModel. As the transition table information in this structure is required, getAttractors must be called in synchronous mode and with returnTable set to TRUE. Similarly, simulateSymbolicModel must be called with returnGraph=TRUE.
state	A 0-1 vector with n elements (where n is the number of genes in the underlying networks) describing the state.

Value

Returns a generic dataframe of the class TransitionTable. For n genes, the first n columns code for the original state (in this case, the state parameter), i.e. each column represents the value of one gene. The next n columns code for the successive state after a transition. The column attractorAssignment indicates the attractor to the state is assigned. If this information is available, the column stepsToAttractor indicates how many transitions are needed from the original state to the attractor. In this case, the table has only one row describing the supplied state. The TransitionTable class supports pretty printing using the print method.

See Also

getBasinOfAttraction, getTransitionTable, getAttractors, simulateSymbolicModel

Examples

```
## Not run:
# load example data
data(cellcycle)

# get attractors
attractors <- getAttractors(cellcycle)

# print information for an arbitrary state
print(getStateSummary(attractors, c(1,1,1,1,1,1,1,1,1,1)))

## End(Not run)
```

getTransitionProbabilities

Get a matrix of transitions and their probabilities in probabilistic Boolean networks

Description

Retrieves the state transitions and their probabilities in a probabilistic Boolean network. This takes the transition table information calculated by the `markovSimulation` method.

Usage

```
getTransitionProbabilities(markovSimulation)
```

Arguments

`markovSimulation`
: An object of class `MarkovSimulation`, as returned by `markovSimulation`. As the transition table information in this structure is required, `markovSimulation` must be called with `returnTable` set to TRUE.

Value

Returns a data frame with the first `n` columns describing the values of the genes before the transition, the next `n` columns describing the values of the genes after the transition, and the last column containing the probability of the transition. Here, `n` is the number of genes in the underlying network. Only transitions with non-zero probability are included.

See Also

`markovSimulation`

Examples

```
## Not run:
# load example network
data(examplePBN)

# perform a Markov chain simulation
sim <- markovSimulation(examplePBN)

# print out the probability table
print(getTransitionProbabilities(sim))

## End(Not run)
```

getTransitionTable *Retrieve the transition table of a network*

Description

Retrieves the transition table and additional attractor information of a network.

Usage

```
getTransitionTable(attractorInfo)
```

Arguments

attractorInfo	An object of class AttractorInfo, as returned by getAttractors, or of class SymbolicSimulation, as returned by simulateSymbolicModel. As the transition table information in this structure is required, getAttractors must be called in synchronous mode and with returnTable set to TRUE. Similarly, simulateSymbolicModel must be called with returnGraph=TRUE.

Details

Depending on the configuration of the call to getAttractors or simulateSymbolicModel that returned attractorInfo, this function either returns the complete transition table (for exhaustive synchronous search) or the part of the transition table calculated in a heuristic synchronous search. Asynchronous search is not supported, as no transition table is calculated.

Value

Returns a generic dataframe of the class TransitionTable. For n genes, the first n columns code for the original state (in this case, the state parameter), i.e. each column represents the value of one gene. The next n columns code for the successive state after a transition. The column attractorAssignment indicates the attractor to the state is assigned. If this information is available, the column stepsToAttractor indicates how many transitions are needed from the original state to the attractor. The table has a row for each possible input state. The TransitionTable class supports pretty printing using the print method.

See Also

getStateSummary, getBasinOfAttraction, getAttractors, simulateSymbolicModel

Examples

```
## Not run:
# load example data
data(cellcycle)

# get attractors
attractors <- getAttractors(cellcycle)
```

```
# print the transition table
print(getTransitionTable(attractors))

## End(Not run)
```

---

igf *Boolean model of the IGF pathway*

Description

A small Boolean model of major components of the IGF (Insuline-like growth receptor) pathway. Through IRS, IGF activates the well-known PI3K-Akt-mTOR signalling cascade. This cascade is finally inactivated by a feedback inhibion of IRS.

The model simplifies several complex formations and cascades by representing them as single nodes and specifying time delays instead. It therefore demonstrates the usage of temporal Boolean networks in **BoolNet**.

Usage

```
data(igf)
```

Format

This data set consists of a variable `igf` of class `SymbolicBooleanNetwork` with 5 genes. The class `SymbolicBooleanNetwork` is described in more detail in `loadNetwork`.

Examples

```
data(igf)

sim <- simulateSymbolicModel(igf)
plotAttractors(sim)
```

---

loadBioTapestry *Import a network from BioTapestry*

Description

Imports a Boolean network from a BioTapestry file (*.btp). BioTapestry is an interactive tool for building, visualizing, and simulating gene-regulatory networks, and can be accessed at https://biotapestry.systemsbiology.net.

Usage

```
loadBioTapestry(file,
                symbolic = FALSE)
```

Arguments

`file`	The name of the file to import. This must be a BioTapestry XML file (*.btp).
`symbolic`	If set to `TRUE`, the function returns an object of class `SymbolicBooleanNetwork` with an expression tree representation. Otherwise, it returns an object of class `BooleanNetwork` with a truth table representation.

Details

The function builds up a Boolean network by importing the nodes, the links between these nodes, and the simulation parameters of the top-level plot of a BioTapestry file. The BioTapestry network should have the following properties:

- All links should be either enhancers or repressors. Unspecified ("neutral") links are ignored.
- In the simulation parameters, each node should specify the correct logical function (AND, OR, XOR) for its inputs.
- Constant genes can be generated by modeling a gene without any input link and setting the simulation parameter `initVal` to 0 or 1.

Value

A network of class `BooleanNetwork` or `SymbolicBooleanNetwork`, as described in `loadNetwork`.

References

W. J. R. Longabaugh, E. H. Davidson, H. Bolour (2005), Computational representation of developmental genetic regulatory networks. Developmental Biology 283(1):1–16.

See Also

`loadNetwork`, `loadSBML`

Examples

```
# import the example BioTapestry file
# included in the package vignette
exampleFile <- system.file("doc/example.btp",
                           package="BoolNet")
net <- loadBioTapestry(exampleFile)

# print the imported network
print(net)
```

`loadNetwork` *Load a Boolean network from a file*

Description

Loads a Boolean network or probabilistic Boolean network from a file and converts it to an internal transition table representation.

Usage

```
loadNetwork(file,
            bodySeparator = ",",
            lowercaseGenes = FALSE,
            symbolic = FALSE)
```

Arguments

`file`	The name of the file to be read
`bodySeparator`	An optional separation character to divide the target factors and the formulas. Default is ",".
`lowercaseGenes`	If set to `TRUE`, all gene names are converted to lower case, i.e. the gene names are case-insensitive. This corresponds to the behaviour of **BoolNet** versions prior to 1.5. Defaults to `FALSE`.
`symbolic`	If set to `TRUE`, a symbolic representation of class `SymbolicBooleanNetwork` is returned. This is not available for asynchronous or probabilistic Boolean networks, but is required for the simulation of networks with extended temporal predicates and time delays (see `simulateSymbolicModel`). If such predicates are detected, the switch is activated by default.

Details

Depending on whether the network is loaded in truth table representation or not, the supported network file formats differ slightly.

For the truth table representation (`symbolic=FALSE`), the language basically consists of expressions based on the Boolean operators AND (&), or (|), and NOT (!). In addition, some convenience operators are included (see EBNF and operator description below). The first line contains a header. In case of a Boolean network with only one function per gene, the header is "targets, functions"; in a probabilistic network, there is an optional third column "probabilities". All subsequent lines contain Boolean rules or comment lines that are omitted by the parser. A rule consists of a target gene, a separator, a Boolean expression to calculate a transition step for the target gene, and an optional probability for the rule (for probabilistic Boolean networks only – see below).

The EBNF description of the network file format is as follows:

```
Network          = Header Newline {Rule Newline | Comment Newline};
Header           = "targets" Separator "factors";
```

```
Rule             = GeneName Separator BooleanExpression [Separator Probability];
Comment            = "#" String;
BooleanExpression = GeneName
                     | "!" BooleanExpression
                     | "(" BooleanExpression ")"
                     | BooleanExpression " & " BooleanExpression
                     | BooleanExpression " | " BooleanExpression;
                     | "all(" BooleanExpression {"," BooleanExpression} ")"
                     | "any(" BooleanExpression {"," BooleanExpression} ")"
                     | "maj(" BooleanExpression {"," BooleanExpression} ")"
              | "sumgt(" BooleanExpression {"," BooleanExpression} "," Integer ")"
              | "sumlt(" BooleanExpression {"," BooleanExpression} "," Integer ")";
GeneName           = ? A gene name from the list of involved genes ?;
Separator          = ",";
Integer            = ? An integer value?;
Probability        = ? A floating-point number ?;
String             = ? Any sequence of characters (except a line break) ?;
Newline            = ? A line break character ?;
```

The extended format for Boolean networks with temporal elements that can be loaded if `symbolic=TRUE` additionally allows for a specification of time steps. Furthermore, the operators can be extended with iterators that evaluate their arguments over multiple time steps.

```
Network                = Header Newline
                         {Function Newline | Comment Newline};
Header                 = "targets" Separator "factors";
Function               = GeneName Separator BooleanExpression;
Comment                = "#" String;
BooleanExpression    = GeneName | GeneName TemporalSpecification | BooleanOperator | TemporalOperator
BooleanOperator        =   BooleanExpression
                         | "!" BooleanExpression
                         | "(" BooleanExpression ")"
                         | BooleanExpression " & " BooleanExpression
                         | BooleanExpression " | " BooleanExpression;
TemporalOperator       =   "all" [TemporalIteratorDef]
                                "(" BooleanExpression {"," BooleanExpression} ")"
                         | "any" [TemporalIteratorDef]
                                "(" BooleanExpression {"," BooleanExpression} ")"
                         | "maj" [TemporalIteratorDef]
                                "(" BooleanExpression {"," BooleanExpression} ")"
                         | "sumgt" [TemporalIteratorDef]
                        "(" BooleanExpression {"," BooleanExpression} "," Integer ")"
                         | "sumlt" [TemporalIteratorDef]
                        "(" BooleanExpression {"," BooleanExpression} "," Integer ")"
                         | "timeis" "(" Integer ")"
                         | "timegt" "(" Integer ")"
                         | "timelt" "(" Integer ")";
TemporalIteratorDef    = "[" TemporalIterator "=" Integer ".." Integer "]";
TemporalSpecification = "[" TemporalOperand {"+" TemporalOperand | "-" TemporalOperand} "]";
```

```
TemporalOperand       = TemporalIterator | Integer
TemporalIterator      = ? An alphanumeric string ?;
GeneName              = ? A gene name from the list of involved genes ?;
Separator             = ",";
Integer               = ? An integer value?;
String                = ? Any sequence of characters (except a line break) ?;
Newline               = ? A line break character ?;
```

The meaning of the operators is as follows:

`all` Equivalent to a conjunction of all arguments. For symbolic networks, the operator can have a time range, in which case the arguments are evaluated for each time point specified in the iterator.

`any` Equivalent to a disjunction of all arguments. For symbolic networks, the operator can have a time range, in which case the arguments are evaluated for each time point specified in the iterator.

`maj` Evaluates to true if the majority of the arguments evaluate to true. For symbolic networks, the operator can have a time range, in which case the arguments are evaluated for each time point specified in the iterator.

`sumgt` Evaluates to true if the number of arguments (except the last) that evaluate to true is greater than the number specified in the last argument. For symbolic networks, the operator can have a time range, in which case the arguments are evaluated for each time point specified in the iterator.

`sumlt` Evaluates to true if the number of arguments (except the last) that evaluate to true is less than the number specified in the last argument. For symbolic networks, the operator can have a time range, in which case the arguments are evaluated for each time point specified in the iterator.

`timeis` Evaluates to true if the current absolute time step (i.e. number of state transitions performed from the current start state) is the same as the argument.

`timelt` Evaluates to true if the current absolute time step (i.e. number of state transitions performed from the current start state) is the less than the argument.

`timegt` Evaluates to true if the current absolute time step (i.e. number of state transitions performed from the current start state) is greater than the argument.

If `symbolic=FALSE` and there is exactly one rule for each gene, a Boolean network of class `BooleanNetwork` is created. In these networks, constant genes are automatically fixed (e.g. knocked-out or over-expressed). This means that they are always set to the constant value, and states with the complementary value are not considered in transition tables etc. If you would like to change this behaviour, use `fixGenes` to reset the fixing.

If `symbolic=FALSE` and two or more rules exist for the same gene, the function returns a probabilistic network of class `ProbabilisticBooleanNetwork`. In this case, alternative rules may be annotated with probabilities, which must sum up to 1 for all rules that belong to the same gene. If no probabilities are supplied, uniform distribution is assumed.

If `symbolic=TRUE`, a symbolic representation of a (possibly temporal) Boolean network of class `SymbolicBooleanNetwork` is created.

Value

If `symbolic=FALSE` and only one function per gene is specified, a structure of class `BooleanNetwork` representing the network is returned. It has the following components:

`genes` A vector of gene names involved in the network. This list determines the indices of genes in inputs of functions or in state bit vectors.

`interactions` A list with `length(genes)` elements, where the i-th element describes the transition function for the i-th gene. Each element has the following sub-components:

input A vector of indices of the genes that serve as the input of the Boolean transition function. If the function has no input (i.e. the gene is constant), the vector consists of a zero element.

func The transition function in truth table representation. This vector has `2^length(input)` entries, one for each combination of input variables. If the gene is constant, the function is 1 or 0.

expression A string representation of the Boolean expression from which the truth table was generated

`fixed` A vector specifying which genes are knocked-out or over-expressed. For each gene, there is one element which is set to 0 if the gene is knocked-out, to 1 if the gene is over-expressed, and to -1 if the gene is not fixed at all, i. e. can change its value according to the supplied transition function. Constant genes are automatically set to fixed values.

If `symbolic=FALSE` and there is at least one gene with two or more alternative transition functions, a structure of class `ProbabilisticBooleanNetwork` is returned. This structure is similar to `BooleanNetwork`, but allows for storing more than one function in an interaction. It consists of the following components:

`genes` A vector of gene names involved in the network. This list determines the indices of genes in inputs of functions or in state bit vectors.

`interactions` A list with `length(genes)` elements, where the i-th element describes the alternative transition functions for the i-th gene. Each element is a list of transition functions. In this second-level list, each element has the the following sub-components:

input A vector of indices of the genes that serve as the input of the Boolean transition function. If the function has no input (i.e. the gene is constant), the vector consists of a zero element.

func The transition function in truth table representation. This vector has `2^length(input)` entries, one for each combination of input variables. If the gene is constant, the function is -1.

expression A string representation of the underlying Boolean expression

probability The probability that the corresponding transition function is chosen

`fixed` A vector specifying which genes are knocked-out or over-expressed. For each gene, there is one element which is set to 0 if the gene is knocked-out, to 1 if the gene is over-expressed, and to -1 if the gene is not fixed at all, i. e. can change its value according to the supplied transition function. You can knock-out and over-express genes using `fixGenes`.

If `symbolic=TRUE`, a structure of class `SymbolicBooleanNetwork` that represents the network as expression trees is returned. It has the following components:

`genes` A vector of gene names involved in the network. This list determines the indices of genes in inputs of functions or in state bit vectors.

`interactions` A list with `length(genes)` elements, where the i-th element describes the transition function for the i-th gene in a symbolic representation. Each such element is a list that represents a recursive expression tree, possibly consisting of sub-elements (operands) that are expression trees themselves. Each element in an expression tree can be a Boolean/temporal operator, a literal ("atom") or a numeric constant.

`internalStructs` A pointer referencing an internal representation of the expression trees as raw C objects. This is used for simulations and must be set to NULL if `interactions` are changed to force a refreshment.

`timeDelays` An integer vector storing the temporal memory sizes required for each of the genes in the network. That is, the vector stores the minimum number of predecessor states of each gene that need to be saved to determine the successor state of the network.

`fixed` A vector specifying which genes are knocked-out or over-expressed. For each gene, there is one element which is set to 0 if the gene is knocked-out, to 1 if the gene is over-expressed, and to -1 if the gene is not fixed at all, i. e. can change its value according to the supplied transition function. Constant genes are automatically set to fixed values.

See Also

`getAttractors`, `simulateSymbolicModel`, `markovSimulation`, `stateTransition`, `fixGenes`, `loadSBML`, `loadBioTapestry`

Examples

```
## Not run:
# write example network to file
fil <- tempfile(pattern = "testNet")
sink(fil)
cat("targets, factors\n")
cat("Gene1, !Gene2 | !Gene3\n")
cat("Gene2, Gene3 & Gene4\n")
cat("Gene3, Gene2 & !Gene1\n")
cat("Gene4, 1\n")
sink()

# read file
net <- loadNetwork(fil)
print(net)

## End(Not run)
```

loadSBML — *Load an SBML document*

Description

Loads an SBML document that specifies a qualitative model using the `sbml-qual` extension package.

Usage

```
loadSBML(file, symbolic=FALSE)
```

Arguments

`file`	The SBML document to be imported
`symbolic`	If set to `TRUE`, the function returns an object of class `SymbolicBooleanNetwork` with an expression tree representation. Otherwise, it returns an object of class `BooleanNetwork` with a truth table representation.

Details

The import assumes an SBML level 3 version 1 document with the `sbml-qual` extension package version 1.0. **BoolNet** only supports a subset of the `sbml-qual` standard. The function tries to import those documents that describe a logical model with two possible values per species. It does not support general logical models with more than two values per species or Petri nets.

Further details on the import:

- The import supports multiple function terms with the same output for a transition and interprets them as a disjunction, as proposed in the specification.
- Comparison operators are converted to the corresponding Boolean expressions.
- Compartments are ignored.

For the import, the **XML** package is required.

Value

Returns a structure of class `BooleanNetwork` or `SymbolicBooleanNetwork`, as described in `loadNetwork`.

References

http://sbml.org/Documents/Specifications/SBML_Level_3/Packages/Qualitative_Models_(qual)

See Also

`toSBML`, `loadNetwork`

Examples

```
## Not run:
# load the cell cycle network
data(cellcycle)

fil <- tempfile()
# export the network to SBML
toSBML(cellcycle, fil)

# reimport the model
print(loadSBML(fil))

## End(Not run)
```

`markovSimulation` *Identify important states in probabilistic Boolean networks*

Description

Identifies important states in probabilistic Boolean networks (PBN) using a Markov chain simulation

Usage

```
markovSimulation(network,
                 numIterations = 1000,
                 startStates = list(),
                 cutoff = 0.001,
                 returnTable = TRUE)
```

Arguments

`network`	An object of class `ProbabilisticBooleanNetwork` or `BooleanNetwork` whose transitions are simulated
`numIterations`	The number of iterations for the matrix multiplication, which corresponds to the number of state transitions to simulate
`startStates`	An optional list of start states. Each entry of the list must be a vector with a 0/1 value for each gene. If specified, the simulation is restricted to the states reachable from the supplied start states. Otherwise, all states are considered.
`cutoff`	The cutoff value used to determine if a probability is 0. All output probabilities less than or equal to this value are set to 0.
`returnTable`	If set to true, a transition table annotated with the probabilities for the transitions is included in the results. This is required by `plotPBNTransitions` and `getTransitionProbabilities`.

Details

The algorithm identifies important states by performing the following steps: First, a Markov matrix is calculated from the set of transition functions, where each entry of the matrix specifies the probability of a state transition from the state belonging to the corresponding row to the state belonging to the corresponding column. A vector is initialized with uniform probability for all states (or – if specified – uniform probability for all start states) and repeatedly multiplied with the Markov matrix. The method returns all states with non-zero probability in this vector. See the references for more details.

Value

An object of class `MarkovSimulation` with the following components:

`reachedStates` A data frame with one state in each row. The first columns specify the gene values of the state, and the last column holds the probability that the corresponding state is reached after `numIterations` transitions. Only states with a probability greater than `cutoff` are included in this table.

`genes` A vector of gene names of the input network

`table` If `returnTable=TRUE`, this structure holds a table of transitions with the corresponding probabilities that transitions are chosen. This is a list with the following components:

initialStates A matrix of encoded start states of the transitions

nextStates The encoded successor states of the transitions

probabilities The probabilities that the transitions are chosen in a single step

References

I. Shmulevich, E. R. Dougherty, S. Kim, W. Zhang (2002), Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks. Bioinformatics 18(2):261–274.

See Also

`reconstructNetwork`, `plotPBNTransitions`, `getTransitionProbabilities`

Examples

```
  ## Not run:
# load example network
data(examplePBN)

# perform a Markov chain simulation
sim <- markovSimulation(examplePBN)

# print the relevant states and transition probabilities
print(sim)

# plot the transitions and their probabilities
plotPBNTransitions(sim)

## End(Not run)
```

perturbNetwork *Perturb a Boolean network randomly*

Description

Modifies a synchronous, asynchronous, or probabilistic Boolean network by randomly perturbing either the functions for single genes or the state transitions. Random perturbations can be employed to assess the stability of the network.

Usage

```
perturbNetwork(network,
               perturb = c("functions","transitions"),
               method = c("bitflip","shuffle"),
               simplify = (perturb[1]!="functions"),
               readableFunctions = FALSE,
               excludeFixed = TRUE,
               maxNumBits = 1,
               numStates = max(1,2^length(network$genes)/100))
```

Arguments

`network` A network structure of class `BooleanNetwork` or `ProbabilisticBooleanNetwork`. These networks can be read from files by `loadNetwork`, generated by `generateRandomNKNetwork`, or reconstructed by `reconstructNetwork`.

`perturb` If set to "functions", a transition function of a single gene is chosen at random and perturbed directly. This is the default mode. If set to "transitions", the transition table is generated, one or several state transitions are perturbed randomly, and the gene transition functions are rebuilt from the modified transition table. `perturb="transitions"` is only allowed for non-probabilistic networks of class `BooleanNetworks`.

`method` The perturbation method to be applied to the functions or transitions. "bitflip" randomly inverts one or several bits (depending on the value of `maxNumBits`). "shuffle" generates a random permutation of the positions in the function or state and rearranges the bits according to this permutation.

`simplify` If this is true, `simplifyNetwork` is called to simplify the gene transition functions after the perturbation. This removes irrelevant input genes. Defaults to TRUE if `perturb` is "transitions", and to FALSE otherwise.

`readableFunctions` If this is true, readable DNF representations of the truth tables of the functions are generated. These DNF are displayed when the network is printed. The DNF representations are not minimized and can thus be very long. If set to FALSE, the truth table result column is displayed.

`excludeFixed` Determines whether fixed variables can also be perturbed (if set to FALSE) or if they are excluded from the perturbation (if set to TRUE). Default is TRUE.

maxNumBits	The maximum number of bits to be perturbed in one function or state. Defaults to 1.
numStates	The number of state transitions to be perturbed if perturb is "transitions". Defaults to 1

Value

Depending on the input, an object of class BooleanNetwork or ProbabilisticBooleanNetwork containing the perturbed copy of the original network is returned. The classes BooleanNetwork and ProbabilisticBooleanNetwork are described in more detail in loadNetwork.

References

Y. Xiao and E. R. Dougherty (2007), The impact of function perturbations in Boolean networks. Bioinformatics 23(10):1265–1273.

I. Shmulevich, E. R. Dougherty, W. Zhang (2002), Control of stationary behavior in probabilistic Boolean networks by means of structural intervention. Journal of Biological Systems 10(4):431–445.

See Also

loadNetwork, generateRandomNKNetwork, reconstructNetwork, simplifyNetwork

Examples

```
## Not run:
# load example data
data(cellcycle)

# perturb the network
perturbedNet1 <- perturbNetwork(cellcycle, perturb="functions", method="shuffle")
perturbedNet2 <- perturbNetwork(cellcycle, perturb="transitions", method="bitflip")

# get attractors
print(getAttractors(perturbedNet1))
print(getAttractors(perturbedNet2))

## End(Not run)
```

---

perturbTrajectories *Perturb the state trajectories and calculate robustness measures*

Description

Perturbs the state trajectories of a network and assesses the robustness by comparing the successor states or the attractors of a set of initial states and a set of perturbed copies of these initial states.

Usage

```
perturbTrajectories(network,
                    measure = c("hamming", "sensitivity", "attractor"),
                    numSamples = 1000,
                    flipBits = 1,
                updateType = c("synchronous", "asynchronous", "probabilistic"),
                    gene,
                    ...)
```

Arguments

network	A network structure of class BooleanNetwork, SymbolicBooleanNetwork or ProbabilisticBooleanNetwork whose robustness is measured.
measure	Defines the way the robustness is measured (see Details).
numSamples	The number of randomly generated pairs of initial states and perturbed copies. Defaults to 1000.
flipBits	The number of bits that are flipped to generate a perturbed copy of an initial state. Defaults to 1.
updateType	If measure="hamming", the type of update that is performed to calculate successor states.
gene	If measure="sensitivity", the name or index of the gene for whose transition function the average sensitivity is calculated.
...	Further parameters to stateTransition and getAttractors.

Details

The function generates a set of numSamples initial states and then applies flipBits random bit flips to each initial state to generate a perturbed copy of each initial state. For each pair of initial state and perturbed state, a robustness statistic is calculated depending measure:

If measure="hamming", the normalized Hamming distances between the successor states of each initial state and the corresponding perturbed state are calculated.

If measure="sensitivity", the average sensitivity of a specific transition function (specified in the gene parameter) is approximated: The statistic is a logical vector that is TRUE if gene differs in the successor states of each initial state and the corresponding perturbed state.

If measure="attractor", the attractors of all initial states and all perturbed states are identified. The statistic is a logical vector specifying whether the attractors are identical in each pair of initial state and perturbed initial state.

Value

A list with the following items:

stat	A vector of size numSamples containing the robustness statistic for each pair of initial state and perturbed copy.
value	The summarized statistic (i.e. the mean value) over all state pairs.

References

I. Shmulevich and S. A. Kauffman (2004), Activities and Sensitivities in Boolean Network Models. Physical Review Letters 93(4):048701.

See Also

testNetworkProperties, perturbNetwork

Examples

```
## Not run:
data(cellcycle)

# calculate average normalized Hamming distance of successor states
hamming <- perturbTrajectories(cellcycle, measure="hamming", numSamples=100)
print(hamming$value)

# calculate average sensitivity of transition function for gene "Cdh1"
sensitivity <- perturbTrajectories(cellcycle, measure="sensitivity", numSamples=100, gene="Cdh1")
print(sensitivity$value)

# calculate percentage of equal attractors for state pairs
attrEqual <- perturbTrajectories(cellcycle, measure="attractor", numSamples=100)
print(attrEqual$value)

## End(Not run)
```

---

plotAttractors *Plot state tables or transition graphs of attractors*

Description

Visualizes attractors, either by drawing a table of the involved states in two colors, or by drawing a graph of transitions between the states of the attractor.

Usage

```
plotAttractors(attractorInfo,
               subset,
               title = "",
               mode = c("table","graph"),
               grouping = list(),
               plotFixed = TRUE,
               onColor = "#4daf4a",
               offColor = "#e41a1c",
               layout = layout.circle,
               drawLabels = TRUE,
               drawLegend = TRUE,
```

```
                ask = TRUE,
                reverse = FALSE,
                borderColor = "black",
                eps = 0.1,
                allInOnePlot = FALSE,
                ...)
```

Arguments

`attractorInfo` An object of class `AttractorInfo`, as returned by `getAttractors`, or an object of class `SymbolicSimulation`, as returned by `simulateSymbolicModel`.

`subset` An subset of attractors to be plotted. This is a vector of attractor indices in `attractorInfo`.

`title` An optional title for the plot

`mode` Switches between two kinds of attractor plots. See Details for more information. Default is "table".

`grouping` This optional parameter is only used if `mode="table"` and specifies a structure to form groups of genes in the plot. This is a list with the following elements:

class A vector of names for the groups. These names will be printed in the region belonging to the group in the plot.

index A list with the same length as `class`. Each element is a vector of gene names or gene indices belonging to the group.

`plotFixed` This optional parameter is only used if `mode="table"`. If this is true, genes with fixed values are included in the plot. Otherwise, these genes are not drawn.

`onColor` This optional parameter is only used if `mode="table"` and specifies the color value for the 1/ON values in the table. Defaults to green.

`offColor` This optional parameter is only used if `mode="table"` and specifies the color value for the 0/OFF values in the table. Defaults to red.

`layout` If `mode="graph"`, this parameter specifies a layouting function that determines the placement of the nodes in the graph. Please refer to the `layout` manual entry in the **igraph** package for further details. By default, the circle layout is used.

`drawLabels` This parameter is only relevant if `mode="graph"`. It determines whether the nodes of the graph are annotated with the corresponding values of the genes in the attractor states.

`drawLegend` Specifies whether a color key for the ON/OFF states is drawn if `mode="table"`. Defaults to `TRUE`.

`ask` If set to true, the plot function will prompt for a user input for each new plot that is shown on an interactive device (see `link{par("ask")}`).

`reverse` Specifies the order of the genes in the plot. By default, the first gene is placed in the first row of the plot. If `reverse=TRUE` (which was the default until **BoolNet** version 2.0.2), the first gene in the network is placed in the bottom row of the plot.

`borderColor` Specifies the border or seprating color of states in an attractor. Defaults to `"black"`.

eps	Specifies plotting margin for the sequence of states. Defaults to 0.1.
allInOnePlot	If this is TRUE then all attractors, with mode = "table", are plotted in one plot as specified internally by par$mfrow parameter. Previous value of the par$mfrow parameter is preserved. Defaults to FALSE, meaning the plots for more than one attractor will be switched interactively or all plotted in an non-interactive graphical device.
...	Further graphical parameters to be passed to plot.igraph if mode="graph".

Details

This function comprises two different types of plots:

The "table" mode visualizes the gene values of the states in the attractor and is only suited for synchronous or steady-state attractors. Complex asynchronous attractors are omitted in this mode. Attractors in attractorInfo are first grouped by length. Then, a figure is plotted to the currently selected device for each attractor length (i.e. one plot with all attractors consisting of 1 state, one plot with all attractors consisting of 2 states, etc.). If ask=TRUE and the standard X11 output device is used, the user must confirm that the next plot for the next attractor size should be shown. The figure is a table with the genes in the rows and the states of the attractors in the columns. Cells of the table are (by default) red for 0/OFF values and green for 1/ON values. If grouping is set, the genes are rearranged according to the indices in the group, horizontal separation lines are plotted between the groups, and the group names are printed.

The "graph" mode visualizes the transitions between different states. It creates a graph in which the vertices are the states in the attractor and the edges are state transitions among these states. This mode can visualize all kinds of attractors, including complex/loose attractors. One plot is drawn for each attractor. As before, this means that on the standard output device, only the last plot is displayed unless you set par(mfrow=c(...)) accordingly.

Value

If mode="table", a list of matrices corresponding to the tables is returned. Each of these matrices has the genes in the rows and the states of the attractors in the columns.

If mode="graph", a list of objects of class igraph is returned. Each of these objects describes the graph for one attractor.

See Also

getAttractors, simulateSymbolicModel, attractorsToLaTeX, plotSequence, sequenceToLaTeX

Examples

```
## Not run:
# load example data
data(cellcycle)

# get attractors
attractors <- getAttractors(cellcycle)

# calculate number of different attractor lengths,
# and plot attractors side by side in "table" mode
```

```
par(mfrow=c(1, length(table(sapply(attractors$attractors,
                          function(attractor)
                          {
                            length(attractor$involvedStates)
                          })))))
plotAttractors(attractors)

# plot attractors in "graph" mode
par(mfrow=c(1, length(attractors$attractors)))
plotAttractors(attractors, mode="graph")

# identify asynchronous attractors
attractors <- getAttractors(cellcycle, type="asynchronous")

# plot attractors in "graph" mode
par(mfrow=c(1, length(attractors$attractors)))
plotAttractors(attractors, mode="graph")

## End(Not run)
```

---

plotNetworkWiring *Plot the wiring of a Boolean network*

Description

Plots the wiring of genes (i.e. the gene dependencies) of a synchronous or probabilistic Boolean network. The nodes of the graph are the genes, and the directed edges show the dependencies of the genes. This requires the **igraph** package.

Usage

```
plotNetworkWiring(network,
                  layout = layout.fruchterman.reingold,
                  plotIt = TRUE, ...)
```

Arguments

network	A network structure of class BooleanNetwork, SymbolicBooleanNetwork or ProbabilisticBooleanNetwork. These networks can be read from files by loadNetwork, generated by generateRandomNKNetwork, or reconstructed by reconstructNetwork.
layout	A layouting function that determines the placement of the nodes in the graph. Please refer to the layout manual entry in the **igraph** package for further details. By default, the Fruchterman-Reingold algorithm is used.
plotIt	If this is true, a plot is generated. Otherwise, only an object of class igraph is returned, but no plot is drawn.
...	Further graphical parameters to be passed to plot.igraph.

Details

This function uses the `plot.igraph` function from the **igraph** package. The plots are customizeable using the `...` argument. For details on possible parameters, please refer to `igraph.plotting`.

Value

Returns an invisible object of class `igraph` containing the wiring graph.

See Also

`loadNetwork`, `generateRandomNKNetwork`, `reconstructNetwork`, `plotStateGraph`, `igraph.plotting`

Examples

```
## Not run:
# load example data
data(cellcycle)

# plot wiring graph
plotNetworkWiring(cellcycle)

## End(Not run)
```

---

plotPBNTransitions *Visualize the transitions in a probabilistic Boolean network*

Description

Visualizes the state transitions and their probabilities in a probabilistic Boolean network. This takes the transition table information calculated by the `markovSimulation` method. Only transitions with non-zero probability are included in the plot. The function requires the **igraph** package.

Usage

```
plotPBNTransitions(markovSimulation,
                   stateSubset,
                   drawProbabilities = TRUE,
                   drawStateLabels = TRUE,
                   layout = layout.fruchterman.reingold,
                   plotIt = TRUE, ...)
```

Arguments

`markovSimulation`
: An object of class `MarkovSimulation`, as returned by `markovSimulation`. As the transition table information in this structure is required, `markovSimulation` must be called with `returnTable` set to TRUE.

`stateSubset` An optional list of states, where each element of the list must be a vector with a 0/1 entry for each gene. If this argument is supplied, the graph only contains the specified states and transitions between these states.

`drawProbabilities` If set to true, the edges of the graph are annotated with the probabilities of the corresponding transitions. Default is TRUE.

`drawStateLabels` If set to true, the vertices of the graph are annotated with the gene values of the corresponding states. Defaults to TRUE.

`layout` A layouting function that determines the placement of the nodes in the graph. Please refer to the `layout` manual entry in the **igraph** package for further details. By default, the Fruchterman-Reingold algorithm is used.

`plotIt` If this is true, a plot is generated. Otherwise, only an object of class `igraph` is returned, but no plot is drawn.

`...` Further graphical parameters to be passed to `plot.igraph`.

Details

This function uses the `plot.igraph` function from the **igraph** package. The plots are customizeable using the `...` argument. For details on possible parameters, please refer to `igraph.plotting`.

Value

Returns an invisible object of class `igraph` containing the wiring graph.

See Also

`markovSimulation`

Examples

```
## Not run:
# load example network
data(examplePBN)

# perform a Markov chain simulation
sim <- markovSimulation(examplePBN)

# plot the transitions and their probabilities
plotPBNTransitions(sim)

## End(Not run)
```

plotSequence *Plot a sequence of states*

Description

Visualizes sequences of states in synchronous Boolean networks, either by drawing a table of the involved states in two colors, or by drawing a graph of transitions between the successive states.

Usage

```
plotSequence(network,
             startState,
             includeAttractorStates = c("all","first","none"),
             sequence,
             title = "",
             mode=c("table","graph"),
             plotFixed = TRUE, grouping = list(),
             onColor="#4daf4a",
             offColor = "#e41a1c",
             layout,
             drawLabels=TRUE,
             drawLegend=TRUE,
             highlightAttractor=TRUE,
             reverse = FALSE,
             borderColor = "black",
             eps=0.1,
             attractor.sep.lwd = 2,
             attractor.sep.col = "blue",
             ...)
```

Arguments

`network` An object of class `BooleanNetwork` or `SymbolicBooleanNetwork` for which a sequence of state transitions is calculated

`startState` The start state of the sequence

`includeAttractorStates`
Specifies whether the actual attractor states are included in the plot or not (see also `getPathToAttractor`). If `includeAttractorStates = "all"` (which is the default behaviour), the sequence ends when the attractor was traversed once. If `includeAttractorStates = "first"`, only the first state of attractor is added to the sequence. If includeAttractorStates = "none", the sequence ends with the last non-attractor state.

`sequence` The alternative call to `plotSequence` requires the specification of the sequence itself instead of the network and the start state. The sequence must be provided as a data frame with the genes in the columns and the successive states in the rows. For example, sequences can be obtained using `getPathToAttractor` or

getAttractorSequence (however, the specialized plot plotAttractors exists for attractors).

`title` An optional title for the plot

`mode` Switches between two kinds of attractor plots. See Details for more information. Default is "table".

`plotFixed` This optional parameter is only used if `mode="table"`. If this is true, genes with fixed values are included in the plot. Otherwise, these genes are not drawn.

`grouping` This optional parameter is only used if `mode="table"` and specifies a structure to form groups of genes in the plot. This is a list with the following elements:

class A vector of names for the groups. These names will be printed in the region belonging to the group in the plot.

index A list with the same length as `class`. Each element is a vector of gene names or gene indices belonging to the group.

`onColor` This optional parameter is only used if `mode="table"` and specifies the color value for the 1/ON values in the table. Defaults to green.

`offColor` This optional parameter is only used if `mode="table"` and specifies the color value for the 0/OFF values in the table. Defaults to red.

`layout` If `mode="graph"`, this parameter specifies a layouting function that determines the placement of the nodes in the graph. Please refer to the layout manual entry in the **igraph** package for further details. By default, the nodes are placed in a horizontal line.

`drawLabels` This parameter is only relevant if `mode="graph"`. It determines whether the nodes of the graph are annotated with the corresponding values of the genes in the attractor states.

`drawLegend` Specifies whether a color key for the ON/OFF states is drawn if `mode="table"`. Defaults to `TRUE`.

`highlightAttractor` If set to true, the attractor states are highlighted in the plot. If `mode="table"`, a line is drawn at the begin of the attractor, and the states are labeled correspondingly. If `mode="graph"`, the attractor transitions are drawn as bold lines. Information on the attractor must be supplied in the attribute `attractor` of the sequence, which is a vector of indices of the states that belong to the attractor. This attribute is usually present if the sequence was obtained using getPathToAttractor.

`reverse` Specifies the order of the genes in the plot. By default, the first gene is placed in the first row of the plot. If `reverse=TRUE` (which was the default until **BoolNet** version 2.0.2), the first gene in the network is placed in the bottom row of the plot.

`borderColor` Specifies the border or seprating color of states in an attractor. Defaults to `"black"`.

`eps` Specifies plotting margin for the sequence of states. Defaults to `0.1`.

`attractor.sep.lwd` Specifies the line width of the attractor separator. Defaults to `2`.

`attractor.sep.col` Specifies the line color of the attractor separator. Defaults to `"blue"`.

`...` Further graphical parameters to be passed to plot.igraph if `mode="graph"`.

Details

This function comprises two different types of plots:

The "table" mode visualizes the gene values of the states in the sequence. The figure is a table with the genes in the rows and the successive states of the sequence in the columns. Cells of the table are (by default) red for 0/OFF values and green for 1/ON values. If `grouping` is set, the genes are rearranged according to the indices in the group, horizontal separation lines are plotted between the groups, and the group names are printed.

The "graph" mode visualizes the transitions between different states. It creates a graph in which the vertices are the states in the sequence and the edges are state transitions among these states.

The function can be called with different types of inputs: The user can specify the parameters `network`, `startState` and `includeAttractorStates`), in which case `getPathToAttractor` is called to obtain the sequence. Alternatively, the sequence can be supplied directly as a data frame in the `sequence` parameter.

Value

If `mode="table"`, a matrix corresponding to the table is returned. The matrix has the genes in the rows and the states of the attractors in the columns. If `sequence` was supplied, this corresponds to the transposed input whose rows may be rearranged if `grouping` was set.

If `mode="graph"`, an object of class `igraph` describing the graph for the sequence is returned.

See Also

`sequenceToLaTeX`, `plotAttractors`, `attractorsToLaTeX`, `getPathToAttractor`, `getAttractorSequence`, `simulateSymbolicModel`

Examples

```
## Not run:
# load example data
data(cellcycle)

# alternative 1: supply network and start state
# and plot sequence as a table
plotSequence(network=cellcycle,
             startState=rep(1,10),
             includeAttractorStates="all")

# alternative 2: calculate sequence in advance
sequence <- getPathToAttractor(cellcycle,
                               state=rep(1,10),
                               includeAttractorStates="all")

# plot sequence as a graph
plotSequence(sequence=sequence,
             mode="graph")

## End(Not run)
```

plotStateGraph *Visualize state transitions and attractor basins*

Description

Plots a graph containing all states visited in stateGraph, and optionally highlights attractors and basins of attraction. This requires the **igraph** package.

Usage

```
plotStateGraph(stateGraph, highlightAttractors = TRUE,
               colorBasins = TRUE, colorSet,
               drawLegend = TRUE, drawLabels = FALSE,
               layout = layout.kamada.kawai,
               piecewise = FALSE,
               basin.lty = 2, attractor.lty = 1,
               plotIt = TRUE,
               colorsAlpha = c(colorBasinsNodeAlpha    = .3,
                               colorBasinsEdgeAlpha    = .3,
                               colorAttractorNodeAlpha = 1,
                               colorAttractorEdgeAlpha = 1),
               ...)
```

Arguments

stateGraph
: An object of class AttractorInfo or SymbolicSimulation, as returned by getAttractors and simulateSymbolicModel respectively. As the transition table information in this structure is required, getAttractors must be called in synchronous mode and with returnTable set to TRUE. Similarly, simulateSymbolicModel must be called with returnGraph=TRUE. Alternatively, stateGraph can be an object of class TransitionTable, which can be extracted using the functions getTransitionTable, getBasinOfAttraction, or getStateSummary

highlightAttractors
: If this parameter is true, edges in attractors are drawn bold and with a different line type (which can be specified in attractor.lty). Defaults to TRUE.

colorBasins
: If set to true, each basin of attraction is drawn in a different color. Colors can be specified in colorSet. Defaults to TRUE.

colorSet
: An optional vector specifying the colors to be used for the different attractor basins. If not supplied, a default color set is used.

drawLegend
: If set to true and colorBasins is true, a legend for the colors of the basins of attraction is drawn. Defaults to TRUE.

drawLabels
: If set to true, the binary encodings of the states are drawn beside the vertices of the graph. As this can be confusing for large graphs, the default value is FALSE.

layout
: A layouting function that determines the placement of the nodes in the graph. Please refer to the layout manual entry in the **igraph** package for further details. By default, the Fruchterman-Reingold algorithm is used.

`piecewise`	If set to true, a piecewise layout is used, i.e. the subgraphs corresponding to different basins of attraction are separated and layouted separately.
`basin.lty`	The line type used for edges in a basin of attraction. Defaults to 2 (dashed).
`attractor.lty`	If `highlightAttractors` is true, this specifies the line type for edges in an attractor. Defaults to 1 (straight).
`plotIt`	If this is true, a plot is generated. Otherwise, only an object of class `igraph` is returned, but no plot is drawn.
`colorsAlpha`	These parameters apply alpha correction to the colors of basins and attractors in the following order: basin node, basin edge, attractor node, attractor edge. Defaults to a vector of length 4 with settings `alpha = 0.3` for basins and `alpha = 1` for attractors.
`...`	Further graphical parameters to be passed to `plot.igraph`.

Details

This function uses the `plot.igraph` function from the **igraph** package. The plots are customizeable using the `...` argument. For details on possible parameters, please refer to `igraph.plotting`.

Value

Returns an invisible object of class `igraph` containing the state graph, including color and line attributes.

See Also

`getAttractors`, `simulateSymbolicModel`, `getTransitionTable`, `getBasinOfAttraction`, `getStateSummary`, `plotNetworkWiring`, `igraph.plotting`

Examples

```
# load example data
data(cellcycle)

# get attractors
attractors <- getAttractors(cellcycle)

# plot state graph
## Not run:
plotStateGraph(attractors, main = "Cell cycle network", layout = layout.fruchterman.reingold)

## End(Not run)
```

print.AttractorInfo *Print attractor cycles*

Description

Specialized print method to print the attractor cycles stored in an `AttractorInfo` object. For simple or steady-state attractors, the states of the attractors are printed in binary encoding in the order they are reached. For asynchronous complex/loose attractors, the possible transitions of the states in the attractor are printed. The method can print either the full states, or only the active genes of the states.

Usage

```
## S3 method for class 'AttractorInfo'
print(x,
      activeOnly = FALSE,
      ...)
```

Arguments

`x`	An object of class `AttractorInfo` to be printed
`activeOnly`	If set to true, a state is represented by a list of active genes (i.e., genes which are set to 1). If set to false, a state is represented by a binary vector with one entry for each gene, specifying whether the gene is active or not. Defaults to `FALSE`.
`...`	Further parameters for the `print` method. Currently not used.

Value

Invisibly returns the printed object

See Also

`print`, `getAttractors`

print.BooleanNetwork *Print a Boolean network*

Description

A specialized method to print an object of class `BooleanNetwork`. This prints the transition functions of all genes. If genes are knocked-out or over-expressed, these genes are listed below the functions.

Usage

```
## S3 method for class 'BooleanNetwork'
print(x, ...)
```

Arguments

x	An object of class BooleanNetwork to be printed
...	Further parameters for the print method. Currently not used.

Value

Invisibly returns the printed object

See Also

print, loadNetwork

---

print.MarkovSimulation

Print the results of a Markov chain simulation

Description

A specialized method to print an object of class MarkovSimulation. This prints all states that have a non-zero probability to be reached after the number of iterations in the Markov simulation. If the simulation was run with returnTable=TRUE, it also prints a table of state transitions and their probabilities to be chosen in a single step.

Usage

```
## S3 method for class 'MarkovSimulation'
print(x,
      activeOnly = FALSE,
      ...)
```

Arguments

x	An object of class MarkovSimulation to be printed
activeOnly	If set to true, a state is represented by a list of active genes (i.e., genes which are set to 1). If set to false, a state is represented by a binary vector with one entry for each gene, specifying whether the gene is active or not. Defaults to FALSE.
...	Further parameters for the print method. Currently not used.

Value

Invisibly returns the printed object

See Also

print, markovSimulation

---

print.ProbabilisticBooleanNetwork

Print a probabilistic Boolean network

Description

A specialized method to print an object of class ProbabilisticBooleanNetwork. For backward compatibility, this method also prints objects of class BooleanNetworkCollection, which have been replaced by ProbabilisticBooleanNetwork. This prints all alternative transition functions and their probabilities. If the network is the result of a reconstruction from time series measurements, it also outputs the error the functions make on the time series. If genes are knocked-out or over-expressed, these genes are listed below the functions.

Usage

```
## S3 method for class 'ProbabilisticBooleanNetwork'
print(x, ...)

## S3 method for class 'BooleanNetworkCollection'
print(x, ...)
```

Arguments

x	An object of class ProbabilisticBooleanNetwork or BooleanNetworkCollection to be printed
...	Further parameters for the print method. Currently not used.

Value

Invisibly returns the printed object

See Also

print, reconstructNetwork, loadNetwork

print.SymbolicSimulation

Print simulation results

Description

Specialized print method to print the information stored in an `AttractorInfo` object. By default, the states of the identified attractors are printed in a binary encoding. Furthermore, the state transition graph and the sequences from the start states to the attractors can be printed. The method can print either the full states, or only the active genes of the states.

Usage

```
## S3 method for class 'SymbolicSimulation'
print(x,
                                    activeOnly = FALSE,
                                    sequences = FALSE,
                                    graph = FALSE,
                                    attractors = TRUE,
                                    ...)
```

Arguments

`x`	An object of class `SymbolicSimulation` to be printed.
`activeOnly`	If set to true, a state is represented by a list of active genes (i.e., genes which are set to 1). If set to false, a state is represented by a binary vector with one entry for each gene, specifying whether the gene is active or not. Defaults to `FALSE`.
`sequences`	If set to true and if `simulateSymbolicModel` has been started with `returnSequences=TRUE`, the sequences from the start states to the attractors are printed. Defaults to `FALSE`.
`graph`	If set to true if `simulateSymbolicModel` has been started with `returnGraph=TRUE`, the state transition table is printed. Defaults to `FALSE`.
`attractors`	If set to true if `simulateSymbolicModel` has been started with `returnAttractor=TRUE`, the state transition table is printed. Defaults to `TRUE`.
`...`	Further parameters for the `print` method. Currently not used.

Value

Invisibly returns the printed object

See Also

`simulateSymbolicModel`

print.TransitionTable *Print a transition table*

Description

Specialized print method to print a transition table with the initial state in the first column, the successor state in the second column, the basin of attraction to which the state leads in the third column, and the number of transitions to the attractor in the fourth column.

Usage

```
## S3 method for class 'TransitionTable'
print(x,
      activeOnly = FALSE,
      ...)

## S3 method for class 'BooleanStateInfo'
print(x,
      activeOnly=FALSE,
      ...)
```

Arguments

x	An object of class TransitionTable or BooleanStateInfo to be printed
activeOnly	If set to true, a state is represented by a list of active genes (i.e., genes which are set to 1). If set to false, a state is represented by a binary vector with one entry for each gene, specifying whether the gene is active or not. Defaults to FALSE.
...	Further parameters for the print method. Currently not used.

Value

Invisibly returns the printed object

See Also

print, getTransitionTable, getBasinOfAttraction, getStateSummary

reconstructNetwork *Reconstruct a Boolean network from time series of measurements*

Description

Reconstructs a Boolean network from a set of time series or from a transition table using the best-fit extension algorithm or the REVEAL algorithm.

Usage

```
reconstructNetwork(measurements,
                   method = c("bestfit", "reveal"),
                   maxK = 5,
                   requiredDependencies = NULL,
                   excludedDependencies = NULL,
                   perturbations=NULL,
                   readableFunctions=FALSE,
                   allSolutions=FALSE,
                   returnPBN=FALSE)
```

Arguments

`measurements` This can either be an object of class `TransitionTable` as returned by `getTransitionTable`, or a set of time series of measurements. In this case, `measurements` must be a list of matrices, each corresponding to one time series. Each row of these matrices contains measurements for one gene on a time line, i. e. column `i+1` contains the successor states of column `i`. The genes must be the same for all matrices in the list. Real-valued time series can be binarized using `binarizeTimeSeries`.

`method` This specifies the reconstruction algorithm to be used. If set to "bestfit", Laehdesmaeki's Best-Fit Extension algorithm is employed. This algorithm is an improvement of the algorithm by Akutsu et al. with a lower runtime complexity. It determines the functions with a minimal error for each gene. If set to "reveal", Liang's REVEAL algorithm is used. This algorithm searches for relevant input genes using the mutual information between the input genes and the output gene.

`maxK` The maximum number of input genes for one gene to be tested. Defaults to 5.

`requiredDependencies` An optional specification of known dependencies that must be included in reconstructed networks. This is a named list containing a vector of gene names (regulators) for each target.

`excludedDependencies` Analogous to `requiredDependencies`, this is an optional specification of dependencies that must not be included in reconstructed networks. This is a named list containing a vector of gene names (prohibited regulators) for each target.

`perturbations` If `measurements` contains data obtained from perturbation experiments (i.e. different targeted knock-outs and overexpressions), this optional parameter is a matrix with one column for each entry in `measurements` and a row for each gene. A matrix entry is 0 for a knock-out of the corresponding gene in the corresponding time series, 1 for overexpression, and NA or -1 for no perturbation. If `measurements` has an element `perturbations` and this argument is not specified, the element of `measurements` is taken.

`readableFunctions` If this is true, readable DNF representations of the truth tables of the functions are generated. These DNF are displayed when the network is printed. The DNF representations are not minimized and can thus be very long. If set to FALSE, the truth table result column is displayed.

`allSolutions` If this is true, all solutions with the minimum error and up to `maxK` inputs are returned. By default, `allSolutions=FALSE`, which means that only the solutions with both minimum error and minimum `k` are returned.

`returnPBN` Specifies the way unknown values in the truth tables of the transition functions ("don't care" values) are processed. If `returnPBN=TRUE`, all possible functions are enumerated recursively, and an object of class `ProbabilisticBooleanNetwork` is returned. This can consume a high amount of memory and computation time. If `returnPBN=FALSE`, the transition functions may contain "don't care" (*) values, and an object of class `BooleanNetworkCollection` is returned. `returnPBN=TRUE` corresponds to the behaviour prior to version 2.0. The default value is `returnPBN=FALSE`.

Details

Both algorithms iterate over all possible input combinations. While Best-Fit Extension is capable of returning functions that do not perfectly explain the measurements (for example, if there are inconsistent measurements or if `maxK` was specified too small), REVEAL only finds functions that explain all measurements. For more information, please refer to the cited publications.

Value

If `returnPBN=TRUE`, the function returns an object of class `ProbabilisticBooleanNetwork`, with each alternative function of a gene having the same probability. The structure is described in detail in `loadNetwork`. In addition to the standard components, each alternative transition function has a component `error` which stores the error of the function on the input time series data. If `returnPBN=FALSE`, the function returns an object of class `BooleanNetworkCollection` that has essentially the same structure as `ProbabilisticBooleanNetwork`, but does not store probabilities and keeps "don't care" values in the functions. Due to the "don't care" (*) values, this collection cannot be simulated directly. However, a specific Boolean network of class `BooleanNetwork` can be extracted from both `BooleanNetworkCollection` and `ProbabilisticBooleanNetwork` structures using `chooseNetwork`.

References

H. Laehdesmaeki, I. Shmulevich and O. Yli-Harja (2003), On Learning Gene-Regulatory Networks Under the Boolean Network Model. Machine Learning 52:147–167.

T. Akutsu, S. Miyano and S. Kuhara (2000). Inferring qualitative relations in genetic networks and metabolic pathways. Bioinformatics 16(8):727–734.

S. Liang, S. Fuhrman and R. Somogyi (1998), REVEAL, a general reverse engineering algorithm for inference of genetic network architectures. Pacific Symposium on Biocomputing 3:18–29.

See Also

generateTimeSeries, binarizeTimeSeries, chooseNetwork

Examples

```
## Not run:
# load example data
data(yeastTimeSeries)

# perform binarization with k-means
bin <- binarizeTimeSeries(yeastTimeSeries)

# reconstruct networks from binarized measurements
net <- reconstructNetwork(bin$binarizedMeasurements, method="bestfit", maxK=3, returnPBN=TRUE)

# print reconstructed net
print(net)

# plot reconstructed net
plotNetworkWiring(net)

## End(Not run)
```

---

saveNetwork *Save a network*

Description

Saves synchronous, asynchronous, probabilistic and temporal networks in the **BoolNet** network file format .

Usage

```
saveNetwork(network,
            file,
            generateDNFs = FALSE,
            saveFixed = TRUE)
```

Arguments

network	An object of class BooleanNetwork or SymbolicBooleanNetwork to be exported
file	The name of the network file to be created
generateDNFs	If network is a BooleanNetwork object, this parameter specifies whether formulae in Disjunctive Normal Form are exported instead of the expressions that describe the transition functions. If set to FALSE, the original expressions are exported. If set to "canonical", a canonical Disjunctive Normal Form is generated from each truth table. If set to "short", the canonical DNF is minimized by joining terms (which can be time-consuming for functions with many inputs). If set to TRUE, a short DNF is generated for functions with up to 12 inputs, and a canonical DNF is generated for functions with more than 12 inputs. For objects of class SymbolicBooleanNetwork, this parameter is ignored.
saveFixed	If set to TRUE, knock-outs and overexpression of genes override their transition functions. That is, if a gene in the network is fixed to 0 or 1, this value is saved, regardless of the transition function. If set to FALSE, the transition function is saved. Defaults to TRUE.

Details

The network is saved in the **BoolNet** file format (see loadNetwork for details).

If the expressions in the transition functions cannot be parsed or generateDNFs is true, a DNF representation of the transition functions is generated.

See Also

loadNetwork

Examples

```
## Not run:
# load the cell cycle network
data(cellcycle)

# save it to a file
saveNetwork(cellcycle, file="cellcycle.txt")

# reload the model
print(loadNetwork("cellcycle.txt"))

## End(Not run)
```

sequenceToLaTeX *Create LaTeX table of state sequences*

Description

Exports tables of state sequences (corresponding to the plot generated by plotSequence with mode="table") to a LaTeX document.

Usage

```
sequenceToLaTeX(network,
                startState,
                includeAttractorStates = c("all","first","none"),
                sequence,
                title = "",
                grouping = list(),
                plotFixed = TRUE,
                onColor="[gray]{0.9}",
                offColor="[gray]{0.6}",
                highlightAttractor=TRUE,
                reverse = FALSE,
                file="sequence.tex")
```

Arguments

network — An object of class BooleanNetwork or SymbolicBooleanNetwork for which a sequence of state transitions is calculated

startState — The start state of the sequence

includeAttractorStates — Specifies whether the actual attractor states are included in the table or not (see also getPathToAttractor). If includeAttractorStates = "all" (which is the default behaviour), the sequence ends when the attractor was traversed once. If includeAttractorStates = "first", only the first state of attractor is added to the sequence. If includeAttractorStates = "none", the sequence ends with the last non-attractor state.

sequence — The alternative call to sequenceToLaTeX requires the specification of the sequence itself instead of the network and the start state. The sequence must be provided as a data frame with the genes in the columns and the successive states in the rows. For example, sequences can be obtained using getPathToAttractor or getAttractorSequence (however, the specialized function attractorsToLaTeX exists for attractors).

title — An optional title for the table

plotFixed — If this is true, genes with fixed values are included in the plot. Otherwise, these genes are not shown.

grouping — This optional parameter specifies a structure to form groups of genes in the table. This is a list with the following elements:

class A vector of names for the groups. These names will be printed in the region belonging to the group in the tabke.

index A list with the same length as `class`. Each element is a vector of gene names or gene indices belonging to the group.

`onColor` An optional color value for the 1/ON values in the table. Defaults to dark grey.

`offColor` An optional color value for the 0/OFF values in the table. Defaults to light grey.

`highlightAttractor`
If set to true, the attractor states are highlighted in the plot by drawing a line at the begin of the attractor and labeling the states correspondingly. Information on the attractor must be supplied in the attribute `attractor` of the sequence, which is a vector of indices of the states that belong to the attractor. This attribute is usually present if the sequence was obtained using `getPathToAttractor`.

`reverse` Specifies the order of the genes in the plot. By default, the first gene is placed in the first row of the table. If `reverse=TRUE`, the first gene in the network is placed in the bottom row of the table.

`file` The file to which the LaTeX document is written. Defaults to "sequence.tex".

Details

This function creates a LaTeX table that visualizes a sequence of states in a synchronous network. The output file does not contain a document header and requires the inclusion of the packages `tabularx` and `colortbl`. The tables have the genes in the rows and the successive states of the sequence in the columns. If not specified otherwise, cells of the table are light grey for 0/OFF values and dark grey for 1/ON values. If `grouping` is set, the genes are rearranged according to the indices in the group, horizontal separation lines are plotted between the groups, and the group names are printed.

The function can be called with different types of inputs: The user can specify the parameters `network`, `startState` and `includeAttractorStates`), in which case `getPathToAttractor` is called to obtain the sequence. Alternatively, the sequence can be supplied directly as a data frame in the `sequence` parameter.

Value

Returns a matrix corresponding to the table. The matrix has the genes in the rows and the states of the attractors in the columns. If `sequence` was supplied, this corresponds to the transposed input whose rows may be rearranged if `grouping` was set.

See Also

`attractorsToLaTeX`, `plotSequence`, `plotAttractors`, `getPathToAttractor`, `getAttractorSequence`.

Examples

```
## Not run:
# load example data
data(cellcycle)

# alternative 1: supply network and start state
```

```
# and export sequence to LaTeX
sequenceToLaTeX(network=cellcycle,
                startState=rep(1,10),
                includeAttractorStates="all",
                file="sequence.txt")

# alternative 2: calculate sequence in advance
sequence <- getPathToAttractor(cellcycle,
                               state=rep(1,10),
                               includeAttractorStates="all")

sequenceToLaTeX(sequence=sequence,
                file="sequence.txt")

## End(Not run)
```

`simplifyNetwork` *Simplify the functions of a synchronous, asynchronous, or probabilistic Boolean network*

Description

Eliminates irrelevant variables from the inputs of the gene transition functions. This can be useful if the network was generated randomly via `generateRandomNKNetwork` or if it was perturbed via `perturbNetwork`.

Usage

```
simplifyNetwork(network, readableFunctions = FALSE)
```

Arguments

`network` A network structure of class `BooleanNetwork`, `ProbabilisticBooleanNetwork` or `BooleanNetworkCollection`. These networks can be read from files by `loadNetwork`, generated by `generateRandomNKNetwork`, or reconstructed by `reconstructNetwork`.

`readableFunctions` This parameter specifies if readable DNF representations of the transition function truth tables are generated and displayed when the network is printed. If set to FALSE, the truth table result column is displayed. If set to "canonical", a canonical Disjunctive Normal Form is generated from each truth table. If set to "short", the canonical DNF is minimized by joining terms (which can be time-consuming for functions with many inputs). If set to TRUE, a short DNF is generated for functions with up to 12 inputs, and a canonical DNF is generated for functions with more than 12 inputs.

Details

The function checks whether the output of a gene transition function is independent from the states of any of the input variables. If this is the case, these input variables are dropped, and the transition function is shortened accordingly.

In non-probabilistic Boolean networks (class BooleanNetwork), constant genes are automatically fixed (e.g. knocked-out or over-expressed). This means that they are always set to the constant value, and states with the complementary value are not considered in transition tables etc. If you would like to change this behaviour, use fixGenes to reset the fixing.

Value

The simplified network of class BooleanNetwork, ProbabilisticBooleanNetwork or BooleanNetworkCollection. These classes are described in more detail in loadNetwork and reconstructNetwork.

See Also

loadNetwork,generateRandomNKNetwork, perturbNetwork, reconstructNetwork, fixGenes

Examples

```
## Not run:
# load example data
data(cellcycle)

# perturb the network
perturbedNet <- perturbNetwork(cellcycle, perturb="functions", method="shuffle")
print(perturbedNet$interactions)

# simplify the network
perturbedNet <- simplifyNetwork(perturbedNet)
print(perturbedNet$interactions)

## End(Not run)
```

---

simulateSymbolicModel *Simulate a symbolic Boolean network*

Description

This function simulates Boolean networks in a symbolic representation, possibly with additional temporal qualifiers. The function can identify attractors, determine the state transition graph, and generate sequences of successive states.

Usage

```
simulateSymbolicModel(network,
              method = c("exhaustive",
                         "random",
                         "chosen",
                         "sat.exhaustive",
                         "sat.restricted"),
              startStates = NULL,
              returnSequences =
                (!(match.arg(method) %in%
                 c("sat.exhaustive", "sat.restricted"))),
              returnGraph =
                  (!(match.arg(method) %in%
                   c("sat.exhaustive", "sat.restricted"))),
              returnAttractors = TRUE,
              maxTransitions = Inf,
              maxAttractorLength = Inf,
              canonical = TRUE)
```

Arguments

`network` A network structure of class `SymbolicBooleanNetwork`. These networks can be read from files by `loadNetwork`, `loadBioTapestry` or `loadSBML` with the `symbolic=TRUE` flag.

`startStates` An optional parameter specifying the start states. If this is an integer value, it denotes the number of random start states to generate. Otherwise, it has to be a list of states. The list elements must either be vectors with one value for each gene in the network, or matrices with the genes in the columns and multiple predecessor states in the rows. These predecessor states may be evaluated if temporal predicates in the network have a time delay of more than one. If the number of supplied predecessor states is smaller than the maximum time delay in the network, genes are assumed to have had the same value as in the first supplied state prior to this state. In particular, if only a single state is supplied, it is assumed that the network always resided in this state prior to starting the simulation.

`method` The simulation method to be used (see details). If `method` is not specified, the desired method is inferred from the type of `startStates`.

`returnSequences` If set to true (and no SAT-based method is chosen), the return value has an element `sequences` specifying the sequences of states to the attractor.

`returnGraph` If set to true (and no SAT-based method is chosen), the return value has an element `graph` specifying the state transition graph of the network.

`returnAttractors` If set to true, the return value has an element `attractors` containing a list of identified attractors.

`maxTransitions` The maximum number of state transitions to be performed for each start state (defaults to `Inf`).

`maxAttractorLength`
: If `method="sat.restricted"`, this required parameter specifies the maximum size of attractors (i.e. the number of states in the loop) to be searched. For `method="sat.exhaustive"`, this parameter is optional and specifies the maximum attractor length for the initial length-restricted search phase that is performed to speed up the subsequent exhaustive search. In this case, changing this value might bring performance benefits, but does not change the results.

`canonical`
: If set to true and `returnAttractors=TRUE`, the states in the attractors are rearranged such that the state whose binary encoding makes up the smallest number is the first element of the vector. This ensures that attractors determined in runs with different start states are comparable, as the cycles may have been entered at different states.

Details

Similarly to `getAttractors`, the symbolic simulator supports different simulation modes which can be specified in the `method` parameter:

- Exhaustive search If `method="exhaustive"`, all possible states in the network are used as start states. If the network has time delays greater than one (temporal network), this means that exhaustive search does not only cover all 2^n possible states for a network with n genes, but also all possible state histories of those genes for which longer delays are required.
- Heuristic search For `method="random"` or `method="chosen"`, a subset of states is used as start states for the simulation.

  If `method="random"`, `startStates` is interpreted as an integer value specifying the number of states to be generated randomly. The algorithm is then initialized with these random start states.

  If `method="chosen"`, `startStates` is interpreted as a list of binary vectors, each specifying one start state (see also parameter description above for details).
- SAT-based attractor search If `method` is "sat.exhaustive" or "sat.restricted", the simulator transforms the network into a satisfiability problem and solves it using Armin Biere's PicoSAT solver (see also `getAttractors` for more details). If `method="sat.restricted"`, only attractors comprising up to `maxAttractorLength` states are identified. Otherwise, the algorithm by Dubrova and Teslenko is applied to identify all attractors. As the SAT-based approaches identify attractors directly, no state sequences and no transition graph are returned.

Value

Returns a list of class `SymbolicSimulation` containing the simulation results:

If `returnSequences` is true and no SAT-based method was chosen, the list contains an element `sequences` consisting of a list of data frames, each representing the state transitions performed from one start state (denoted as time step 0) to the attractor. Here, the columns correspond to the genes in the network, and the rows correspond to the states. Apart from the immediate start state, the sequences may also contain the supplied or assumed predecessor states of the start state (marked by a negative time step t) if the network contains time delays greater than one.

If `returnGraph` is true and no SAT-based method was chosen, the list contains an element `graph` of class `TransitionTable`. Each row of the table corresponds to one state transition from an initial state to a successor state, i.e. an edge in the state transition graph.

If `returnAttractors` is true, the list contains an element `attractors`, which itself is a list of data frames. Each data frame represents one unique attractor, where each column corresponds to a gene, and each row corresponds to one state in the attractor.

If both `returnSequences` and `returnAttractors` are true, there is an additional element `attractorAssignment`. This integer vector specifies the indices of the attractors to which the sequences lead.

The structure supports pretty printing using the `print` method.

References

E. Dubrova, M. Teslenko (2011), A SAT-based algorithm for finding attractors in synchronous Boolean networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics 8(5):1393–1399.

A. Biere (2008), PicoSAT Essentials. Journal on Satisfiability, Boolean Modeling and Computation 4:75-97.

See Also

`loadNetwork`, `loadBioTapestry`, `loadSBML`, `getAttractors`, `plotAttractors`, `attractorsToLaTeX`, `getTransitionTable`, `getBasinOfAttraction`, `getAttractorSequence`, `getStateSummary`, `getPathToAttractor`, `fixGenes`

Examples

```
## Not run:
data(igf)

# exhaustive state space simulation
sim <- simulateSymbolicModel(igf)
plotAttractors(sim)

# exhaustive attractor search using SAT solver
sim <- simulateSymbolicModel(igf, method="sat.exhaustive")
plotAttractors(sim)

## End(Not run)
```

`stateTransition` *Perform a transition to the next state*

Description

Calculates the next state in a supplied network for a given current state

Usage

```
stateTransition(network,
                state,
                type = c("synchronous","asynchronous","probabilistic"),
                geneProbabilities,
                chosenGene,
                chosenFunctions,
                timeStep = 0)
```

Arguments

`network` A network structure of class `BooleanNetwork`, `SymbolicBooleanNetwork` or `ProbabilisticBooleanNetwork`. These networks can be read from files by `loadNetwork`, generated by `generateRandomNKNetwork`, or reconstructed by `reconstructNetwork`.

`state` The current state of the network, encoded as a vector with one 0-1 element for each gene. If `network` is of class `SymbolicBooleanNetwork` and makes use of more than one predecessor state, this can also be a matrix with the genes in the columns and multiple predecessor states in the rows.

`type` The type of transition to be performed.

If set to "synchronous", all genes are updated using the corresponding transition functions.

If set to "asynchronous", only one gene is updated. This gene is either chosen randomly or supplied in parameter `chosenGene`.

If set to "probabilistic", one transition function is chosen for each gene, and the genes are updated synchronously. The functions are either chosen randomly depending on their probabilities, or they are supplied in parameter `chosenFunctions`.

Default is "synchronous" for objects of class `BooleanNetwork` and `SymbolicBooleanNetwork`, and "probabilistic" for objects of class `ProbabilisticBooleanNetwork`.

`geneProbabilities` An optional vector of probabilities for the genes if `type="asynchronous"`. By default, each gene has the same probability to be chosen for the next state transition. These probabilities can be modified by supplying a vector of probabilities for the genes which sums up to one.

`chosenGene` If `type="asynchronous"` and this parameter is supplied, no random update is performed. Instead, the gene with the supplied name or index is chosen for the next transition.

`chosenFunctions` If `type="probabilistic"`, this parameter can contain a set of function indices for each gene. In this case, transition functions are not chosen randomly, but the provided functions are used in the state transition.

`timeStep` An optional parameter that specifies the current time step associated with `state`. This is only relevant for networks of class `SymbolicBooleanNetwork` that make use of time-dependent predicates (`timelt`, `timeis`, `timegt`). Otherwise, this parameter is ignored.

Value

The subsequent state of the network, encoded as a vector with one 0-1 element for each gene.

See Also

loadNetwork, generateRandomNKNetwork, generateState

Examples

```
## Not run:
# load example network
data(cellcycle)

# calculate a synchronous state transition
print(stateTransition(cellcycle, c(1,1,1,1,1,1,1,1,1,1)))

# calculate an asynchronous state transition of gene CycA
print(stateTransition(cellcycle, c(1,1,1,1,1,1,1,1,1,1),
                      type="asynchronous", chosenGene="CycA"))

# load probabilistic network
data(examplePBN)

# perform a probabilistic state transition
print(stateTransition(examplePBN, c(0,1,1),
                      type="probabilistic"))

## End(Not run)
```

---

symbolicToTruthTable *Convert a symbolic network into a truth table representation*

Description

Converts an object of class SymbolicBooleanNetwork into an object of class BooleanNetwork by generating truth tables from the symbolic expression trees.

Usage

```
symbolicToTruthTable(network)
```

Arguments

network	An object of class SymbolicBooleanNetwork to be converted.

Details

The symbolic network network must not contain temporal operators, as these are not compatible with the truth table representation in BooleanNetwork objects.

Value

Returns an object of class BooleanNetwork, as described in loadNetwork.

See Also

truthTableToSymbolic, loadNetwork

Examples

```
## Not run:
# Convert a truth table representation into a
# symbolic representation and back
data(cellcycle)

symbolicNet <- truthTableToSymbolic(cellcycle)
print(symbolicNet)

ttNet <- symbolicToTruthTable(symbolicNet)
print(cellcycle)

## End(Not run)
```

---

testNetworkProperties *Test properties of networks by comparing them to random networks*

Description

This is a general function designed to determine unique properties of biological networks by comparing them to a set of randomly generated networks with similar structure.

Usage

```
testNetworkProperties(network,
                      numRandomNets = 100,
                      testFunction = "testIndegree",
                      testFunctionParams = list(),
                      accumulation = c("characteristic", "kullback_leibler"),
                      alternative=c("greater","less"),
                      sign.level = 0.05,
                      drawSignificanceLevel = TRUE,
                      klBins,
                      klMinVal = 1e-05,
                      linkage = c("uniform", "lattice"),
                      functionGeneration = c("uniform", "biased"),
                      validationFunction, failureIterations=10000,
                      simplify = FALSE,
                      noIrrelevantGenes = TRUE,
                      d_lattice = 1,
```

```
                    zeroBias = 0.5,
                    title = "",
                    xlab,
                    xlim,
                    breaks = 30,
                    ...)
```

Arguments

`network` A network structure of class `BooleanNetwork` or `SymbolicBooleanNetwork`

`numRandomNets` The number of random networks to generate for comparison

`testFunction` The name of a function that calculates characteristic values that describe properties of the network. There are two built-in functions: "testIndegree" calculates the in-degrees of states in the network, and "testAttractorRobustness" counts the occurrences of attractors in perturbed copies. It is possible to supply user-defined functions here. See Details.

`testFunctionParams`
A list of parameters to `testFunction`. The elements of the list depend on the chosen function.

`accumulation` If "characteristic" is chosen, the test function is required to return a single value that describes the network. In this case, a histogram of these values in random networks is plotted, and the value of the original network is inserted as a vertical line.

If "kullback_leibler" is chosen, the test function can return a vector of values which is regarded as a sample from a probability distribution. In this case, the Kullback-Leibler distances of the distributions from the original network and each of the random networks are calculated and plotted in a histogram. The Kullback-Leibler distance measures the difference between two probability distributions. In this case, the resulting histogram shows the distribution of differences between the original network and randomly generated networks.

`alternative` If `accumulation="characteristic"`, this specifies whether the characteristic value is expected to be greater or less than the random results under the alternative hypothesis.

`sign.level` If `accumulation="characteristic"`, this specifies a significance level for a computer-intensive test.

If `alternative="greater"`, the test is significant if the characteristic value is greater than at least `(1-sign.level)*100%` of the characteristic values of the random networks.

If `alternative="less"`, the test is significant if the characteristic value is less than at most `sign.level*100%` of the characteristic values of the random networks.

`drawSignificanceLevel`
If `accumulation="characteristic"` and this is true, a vertical line is plotted for the significance level in the histogram.

`linkage`, `functionGeneration`, `validationFunction`, `failureIterations`, `simplify`, `noIrrelevantGenes`, `d_lattice`, `zeroBias`
The corresponding parameters of `generateRandomNKNetwork` used to generate the random networks. This allows for customization of the network generation

process. The three remaining parameters of `generateRandomNKNetwork` are set to values that ensure structural similarity to the original network: The parameters `n` and `k` are set to the corresponding values of the original network, and `topology="fixed"`.

`klBins` If `accumulation="kullback_leibler"`, the number of bins used to discretize the samples for the Kullback-Leibler distance calculations. By default, each unique value in the samples has its own bin, i.e. no further discretization is performed. The influence of discretization on the resulting histogram may be high.

`klMinVal` If `accumulation="kullback_leibler"`, this defines the minimum probability for the calculation of the Kullback-Leibler distance to ensure stability of the results.

`title` The title of the plots. This is empty by default.

`xlab` Customizes label of the x axis of the histogram. For the built-in test functions, the x axis label is set automatically.

`xlim` Customizes the limits of the x axis of the histogram. For the built-in test functions, suitable values are chosen automatically.

`breaks` Customizes the number of breaks in the

`...` Further graphical parameters for `hist`

Details

This function generically compares properties of biological networks to a set of random networks. It can be extended by supplying custom functions to the parameter `testFunction`. Such a function must have the signature

```
function(network, accumulate=TRUE, params)
```

network This is the network to test. In the process of the comparison, both the original network and the random networks are passed to the function

accumulate If `accumulate=TRUE`, the function must return a single value quantifying the examined property of the network. If `accumulate=FALSE`, the function can return a vector of values (e.g., one value for each gene/state etc.)

params A list of further parameters for the function supplied by the user in `testFunctionParams` (see above). This can contain any type of information your test function needs.

Three built-in functions for synchronous Boolean networks already exist:

testIndegree This function is based on the observation that, often, in biological networks, many state transitions lead to the same states. In other words, there is a small number of "hub" states. In the state graph, this means that the in-degree of some states (i.e., the number of transitions leading to it) is high, while the in-degree of many other states is 0. We observed that random networks do not show this behaviour, thus it may be a distinct property of biological networks. For this function, the parameter `alternative` of `testNetworkProperties` should be set to "greater".

The function does not require any parameter entries in `params`. If `accumulate=FALSE`, it returns the in-degrees of all synchronous states in the network. If `accumulate=TRUE`, the Gini

index of the in-degrees is returned as a characteristic value of the network. The Gini index is a measure of inequality. If all states have an in-degree of 1, the Gini index is 0. If all state transitions lead to one single state, the Gini index is 1.

This function requires the **igraph** package for the analysis of the in-degrees.

testAttractorRobustness This function tests the robustness of attractors in a network to noise. We expect attractors in a real network to be less susceptible to noise than attractors in randomly generated networks, as biological processes can be assumed to be comparatively stable. There are modes of generating noise: Either the functions of the network can be perturbed, or the state trajectories can be perturbed in a simulation of the network. If `perturb="functions"` or `perturb="transitions"`, the function generates a number of perturbed copies of the network using `perturbNetwork` and checks whether the original attractors can still be found in the network. If `perturb="trajectories"`, the network itself is not perturbed. Instead, a set of random initial states is generated, and a set of perturbed states is generated from these initial states by flipping one or more bits. Then, the function tests whether the attractors are the same for the initial states and the corresponding perturbed states. This corresponds to calling `perturbTrajectories` with `measure="attractor"`.

`params` can hold a number of parameters:

numSamples If `perturb="trajectories"`, the number of randomly generated state pairs to generate. Otherwise the number of perturbed networks that are generated.

perturb Specifies the type of perturbation to be applied (possible values: `"functions"`, `"transitions"` and `"trajectories"` – see above).

method, simplify, readableFunctions, excludeFixed, maxNumBits, numStates If `perturb="functions"` or `perturb="transitions"`, these are the corresponding parameters of `perturbNetwork` that influence the way the network is perturbed.

flipBits If `perturb="trajectories"`, the are the corresponding parameters of `perturbTrajectories` that defines how many bits are flipped.

If perturb="functions" or `perturb="transitions"` and `accumulate=FALSE`, the function returns a vector of percentages of original attractors found in each of the perturbed copies of the original network. If `accumulate=TRUE`, the function returns the overall percentage of original attractors found in all perturbed copies.

If `perturb="trajectories"` and `accumulate=FALSE`, the function returns a logical vector of length `numSamples` specifying whether the attractor was the same for each initial state and the corresponding perturbed state. If `accumulate=TRUE`, the function returns the percentage of pairs of initial states and perturbed states for which the attractors were the same.

For this function, the parameter `alternative` of `testNetworkProperties` should be set to "greater".

testTransitionRobustness This function calls `perturbTrajectories` with `measure="hamming"` to measure the average Hamming distance between successor states of randomly generated initial states and perturbed copies of these states.

codeparams can hold parameters `numSamples, flipBits` corresponding to the parameters of `perturbTrajectories` that define how many initial states are drawn and how many bits are flipped.

If `accumulate=FALSE`, the function returns a numeric vector of length `numSamples` with the normalized Hamming distances of all pairs of initial states and perturbed copies. If `accumulate=TRUE`, the mean normalized Hamming distance over all pairs is returned.

For this function, the parameter `alternative` of `testNetworkProperties` should be set to "less".

Value

The function returns a list with the following elements

hist	The histogram that was plotted. The type of histogram depends on the parameter accumulation.
pval	If accumulation="characteristic", a p-value for the alternative hypothesis that the test statistic value of the supplied network is greater than the value of a randomly generated network is supplied.
significant	If accumulation="characteristic", this is true for pval < sign.level.

See Also

generateRandomNKNetwork, perturbNetwork, perturbTrajectories, plotStateGraph, getAttractors

Examples

```
## Not run:
# load mammalian cell cycle network
data(cellcycle)

if (interactive())
# do not run these examples in the package check, as they take some time
{
  # compare the in-degrees of the states in the
  # cell cycle network to random networks
  testNetworkProperties(cellcycle, testFunction="testIndegree", alternative="greater")

  # compare the in-degrees of the states in the
  # cell cycle network to random networks,
  # and plot the Kullback-Leibler distances of the 100 experiments
  testNetworkProperties(cellcycle, testFunction="testIndegree",
                        accumulation = "kullback_leibler")

  # compare the robustness of attractors in the cell cycle network
  # to random networks by perturbing the networks
  testNetworkProperties(cellcycle, testFunction="testAttractorRobustness",
                        testFunctionParams=list(perturb="functions", numSamples=10),
                        alternative="greater")

  # compare the robustness of attractors in the cell cycle network
  # to random networks by perturbing the state trajectories
  testNetworkProperties(cellcycle, testFunction="testAttractorRobustness",
                        testFunctionParams=list(perturb="trajectories", numSamples=10),
                        alternative="greater")

  # compare the robustness of single state transitions in the cell cycle network
  testNetworkProperties(cellcycle, testFunction="testTransitionRobustness",
                        testFunctionParams=list(numSamples=10),
                        alternative="less")
}
```

```
## End(Not run)
```

---

toPajek *Export a network to the Pajek file format*

Description

Exports a network to the Pajek file format to visualize transition trajectories. For more information on Pajek, please refer to http://mrvar.fdv.uni-lj.si/pajek/

Usage

```
toPajek(stateGraph, file = "boolean.net", includeLabels=FALSE, ...)
```

Arguments

stateGraph	An object of class AttractorInfo or SymbolicSimulation, as returned by getAttractors and simulateSymbolicModel respectively. As the transition table information in this structure is required, getAttractors must be called in synchronous mode and with returnTable set to TRUE. Similarly, simulateSymbolicModel must be called with returnGraph=TRUE. Alternatively, stateGraph can be an object of class TransitionTable, which can be extracted using the functions getTransitionTable, getBasinOfAttraction, or getStateSummary .
file	The name of the output file for Pajek. Defaults to "boolean.net".
includeLabels	If set to true, the vertices of the graph in the output file are labeled with the binary encodings of the states. Defaults to FALSE.
...	This is only for compatibility with previous versions and should not be used.

Value

This function has no return value.

See Also

getAttractors, simulateSymbolicModel, getTransitionTable, getBasinOfAttraction, getStateSummary, toSBML

Examples

```
## Not run:
# load example data
data(cellcycle)

# get attractors
attractors <- getAttractors(cellcycle)

# export to Pajek
toPajek(attractors, file="pajek_export.net")
```

```
## End(Not run)
```

---

toSBML *Export a network to SBML*

Description

Exports a synchronous or asynchronous Boolean network to SBML with the `sbml-qual` extension package.

Usage

```
toSBML(network,
       file,
       generateDNFs = FALSE,
       saveFixed = TRUE)
```

Arguments

`network`	An object of class `BooleanNetwork` or `SymbolicBooleanNetwork` to be exported
`file`	The name of the SBML file to be created
`generateDNFs`	If `network` is a `BooleanNetwork` object, this parameter specifies whether formulae in Disjunctive Normal Form are exported instead of the expressions that describe the transition functions. If set to FALSE, the original expressions are exported. If set to "canonical", a canonical Disjunctive Normal Form is generated from each truth table. If set to "short", the canonical DNF is minimized by joining terms (which can be time-consuming for functions with many inputs). If set to TRUE, a short DNF is generated for functions with up to 12 inputs, and a canonical DNF is generated for functions with more than 12 inputs. For objects of class `SymbolicBooleanNetwork`, this parameter is ignored.
`saveFixed`	If set to TRUE, knock-outs and overexpression of genes override their transition functions. That is, if a gene in the network is fixed to 0 or 1, this value is exported, regardless of the transition function. If set to FALSE, the transition function is exported. Defaults to TRUE.

Details

The export creates an SBML file describing a general logical model that corresponds to the Boolean network. Importing tools must support the `sbml-qual` extension package version 1.0.

The export translates the expressions that describe the network transition functions to a MathML description. If these expressions cannot be parsed or `generateDNFs` is true, a DNF representation of the transition functions is generated and exported.

For symbolic networks, temporal operators and delays of more than one time step are not allowed, as they are not compatible with SBML.

References

http://sbml.org/Documents/Specifications/SBML_Level_3/Packages/Qualitative_Models_(qual)

See Also

loadSBML, loadNetwork, saveNetwork, toPajek

Examples

```
## Not run:
# load the cell cycle network
data(cellcycle)

# export the network to SBML
toSBML(cellcycle, file="cellcycle.sbml")

# reimport the model
print(loadSBML("cellcycle.sbml"))

## End(Not run)
```

---

truthTableToSymbolic — *Convert a network in truth table representation into a symbolic representation*

Description

Converts an object of class BooleanNetwork into an object of class SymbolicBooleanNetwork by generating symbolic expression trees.

Usage

```
truthTableToSymbolic(network, generateDNFs = FALSE)
```

Arguments

network	An object of class BooleanNetwork to be converted.
generateDNFs	This parameter specifies whether formulae in Disjunctive Normal Form are generated instead of the parsing the string expressions that describe the transition functions. If set to FALSE, the original expressions are parsed. If set to "canonical", a canonical Disjunctive Normal Form is generated from each truth table. If set to "short", the canonical DNF is minimized by joining terms (which can be time-consuming for functions with many inputs). If set to TRUE, a short DNF is generated for functions with up to 12 inputs, and a canonical DNF is generated for functions with more than 12 inputs.

Value

Returns an object of class SymbolicBooleanNetwork, as described in loadNetwork.

See Also

truthTableToSymbolic, loadNetwork

Examples

```
## Not run:
# Convert a truth table representation into a
# symbolic representation and back
data(cellcycle)

symbolicNet <- truthTableToSymbolic(cellcycle)
print(symbolicNet)

ttNet <- symbolicToTruthTable(symbolicNet)
print(cellcycle)

## End(Not run)
```

---

yeastTimeSeries *Yeast cell cycle time series data*

Description

Preprocessed time series measurements of four genes from the yeast cell cycle data by Spellman et al.

Usage

```
data(yeastTimeSeries)
```

Format

A matrix with 14 measurements for the genes Fhk2, Swi5, Sic1, and Clb1. Each gene is a row of the matrix, and each column is a measurement.

Details

The data were obtained from the web site of the yeast cell cycle analysis project. The time series synchronized with the elutriation method were extracted for the genes Fhk2, Swi5, SIC1, and Clb1. In a preprocessing step, missing values were imputed by taking the means of the measurements of the same genes at neighbouring time points.

Source

P. T. Spellman, G. Sherlock, M. Q. Zhang, V. R. Iyer, K. Anders, M. B. Eisen, P. O. Brown, D. Botstein, B. Futcher (1998), Comprehensive Identification of Cell Cycle-regulated Genes of the Yeast Saccharomyces cerevisiae by Microarray Hybridization. Molecular Biology of the Cell 9(12):3273–3297.

Examples

```
data(yeastTimeSeries)

# the data set is stored in a variable called 'yeastTimeSeries'
print(yeastTimeSeries)
```

Index

attractorsToLaTeX, 3, *22*, *45*, *51*, *63*, *64*, *69*
attributes, *26*

binarizeTimeSeries, 4, *16*, *59*, *61*

cellcycle, 7
chooseNetwork, 8, *60*, *61*

examplePBN, 9

fixGenes, *9*, 10, *13*, *16*, *19*, *22*, *34–36*, *66*, *69*

generateCanalyzing, *12*
generateCanalyzing (generationFunctions), 17
generateNestedCanalyzing, *12*
generateNestedCanalyzing (generationFunctions), 17
generateRandomNKNetwork, 11, *17*, *18*, *22*, *40*, *41*, *46*, *47*, *65*, *66*, *70*, *71*, *73*, *74*, *76*
generateState, 14, *22*, *71*
generateTimeSeries, 15, *61*
generationFunctions, 17
getAttractors, *3*, *4*, *15*, 18, *23–27*, *29*, *36*, *42*, *44*, *45*, *52–54*, *68*, *69*, *76*, *77*
getAttractorSequence, *22*, 23, *50*, *51*, *63*, *64*, *69*
getBasinOfAttraction, *21*, *22*, 24, *26*, *27*, *29*, *52*, *53*, *58*, *69*, *77*
getPathToAttractor, *22*, *23*, 25, *49–51*, *63*, *64*, *69*
getStateSummary, *22*, *25*, 27, *29*, *52*, *53*, *58*, *69*, *77*
getTransitionProbabilities, 28, *38*, *39*
getTransitionTable, *22*, *25–27*, 29, *52*, *53*, *58*, *59*, *69*, *77*

hist, *74*

igf, 30
igraph.plotting, *47*, *48*, *53*

kmeans, *5*

layout, *44*, *46*, *48*, *50*, *52*
loadBioTapestry, 30, *36*, *67*, *69*
loadNetwork, *7–10*, *13*, *18*, *22*, *30*, *31*, 32, *37*, *40*, *41*, *46*, *47*, *55*, *56*, *60*, *62*, *65–67*, *69–72*, *79*, *80*
loadSBML, *31*, *36*, 37, *67*, *69*, *79*

markovSimulation, *28*, *36*, 38, *47*, *48*, *56*

perturbNetwork, *13*, 40, *43*, *65*, *66*, *75*, *76*
perturbTrajectories, 41, *75*, *76*
plot.igraph, *45–48*, *50*, *53*
plotAttractors, *3*, *4*, *22*, 43, *50*, *51*, *64*, *69*
plotNetworkWiring, 46, *53*
plotPBNTransitions, *38*, *39*, 47
plotSequence, *4*, *23*, *26*, *45*, 49, *63*, *64*
plotStateGraph, *47*, 52, *76*
print, *21*, *22*, *25*, *27*, *29*, *54–58*, *69*
print.AttractorInfo, 54
print.BooleanNetwork, 54
print.BooleanNetworkCollection (print.ProbabilisticBooleanNetwork), 56
print.BooleanStateInfo (print.TransitionTable), 58
print.MarkovSimulation, 55
print.ProbabilisticBooleanNetwork, 56
print.SymbolicSimulation, 57
print.TransitionTable, 58

reconstructNetwork, *6*, *8*, *9*, *16*, *18*, *39–41*, *46*, *47*, *56*, 59, *65*, *66*, *70*

saveNetwork, 61, *79*
sequenceToLaTeX, *4*, *23*, *45*, *51*, 63
simplifyNetwork, *12*, *13*, *40*, *41*, 65

simulateSymbolicModel, *3*, *15*, *21–27*, *29*, *32*, *36*, *44*, *45*, *51–53*, *57*, 66, *77*
stateTransition, *15*, *16*, *36*, *42*, 69
symbolicToTruthTable, 71

testAttractorRobustness (testNetworkProperties), 72
testIndegree (testNetworkProperties), 72
testNetworkProperties, *43*, 72
testTransitionRobustness (testNetworkProperties), 72
toPajek, 77, *79*
toSBML, *37*, *77*, 78
truthTableToSymbolic, *72*, 79, *80*

yeastTimeSeries, 80