BoolNetPerturb

Mariana E Martinez-Sanchez

2018-10-09

Introduction

This tutorial explains how to use BoolNetPerturb in conjuction with BoolNet to study Boolean regulatory networks and the biological implications of this analysis.

This tutorial supposes that the reader is familiar with the basic concepts of:

- **Boolean regulatory networks**. There are a lot of basic introductions to the topic like: Kaplan & Glass, 1995, Azpeitia 2011 and Albert 2014. CoLoMoTo also published a more advanced review CoLoMoTo 2015.
- **R programming language**. A good starting point is the R programming course at Coursera.
- **Molecular biology**. Contact your local biologist.
- **Robustness**. The book Wagner 2005 was a great inspiration for this work.

The standar format for logical regulatory networks is SBML-qual.

Installation

The library can be installed from github using `devtools`

```
library(devtools)
install_github("mar-esther23/boolnet-perturb", force = TRUE)
```

Regulatory Networks

Regulatory Networks (RN) are a useful tool for studying the cellular behaivor and robustness of biological systems in response to different kinds of perturbations[Colomoto 2015]. RN integrate the available information of the molecular regulation to predict cellular level phenomena using a mathematical formalism. RN are deterministic dynamic systems. RN consist of nodes -that represent genes, proteins, or other biological processes- and edges -that represent the regulatory interactions among the nodes. Using this information, it is possible to construct functions that describe the state of the nodes depending on the state of its regulators. The value of the node represents wether the gene or protein is active or inactive in the biological system. The effect of the environment can be included in this models as input nodes.

The functions of the network are evaluated to obtain the attractors of the network. Attractors represent stable states in the dynamics of the network and have been related to cell types or biological processes like the cell cycle[Kaufman 1969, Azpeitia 2011, Albert 2014] [Figure 1]. RN let us simulate multiple types of perturbations depending in which part of the RN we alter. We can say that an *attractor* is stable to a perturbation if the RN returns to the same attractor, or plastic if it transitions to a different attractor. The robustness of the *system* is the result of the stability and plasticity of all its attractors. The robustness of the system can differ depending on the perturbation. In this way, robustness is a characteristic of the system that emerges from the interactions between the components of the system.

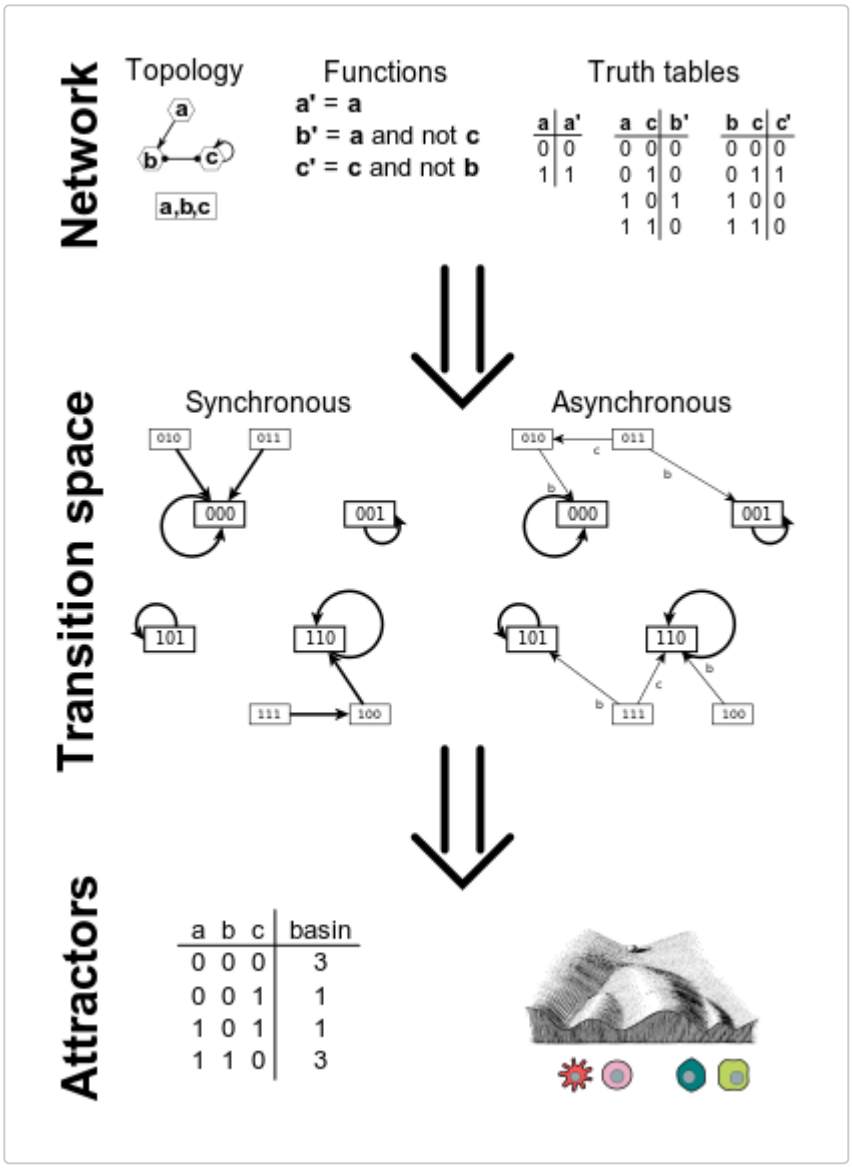
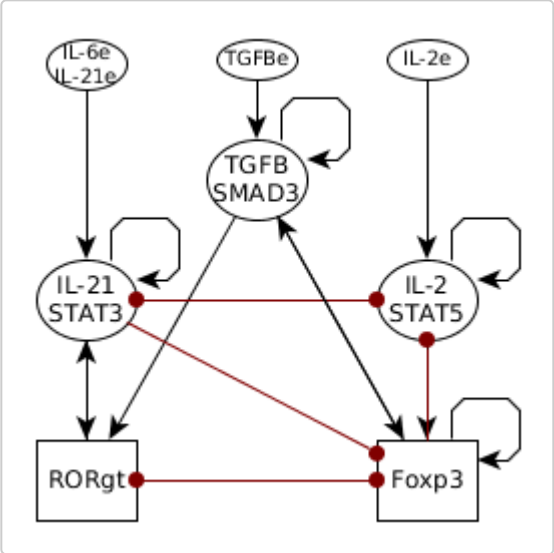


Figure 1: Boolean regulatory networks. A regulatory network consist of nodes, its interactions, and the Boolean functions that regulate the value of each node. The state of each node can be updated using this functions, to obtain the transition space. The attractors of the network correspond to biological cell types.

Biological system: Th17/Treg network

In this work we will use the Th17/Treg regulatory network as an example. This network is a subset of the CD4+ T cell regulatory network that has alredy been published and analysed using this methodology [Martinez-Sanchez 2015]. CD4 + T cells are fundamental for the adaptive immune response. They integrate the signals of the environment and differentiate from naive (Th0) cells into different cell types (Th17, iTreg, Th3, etc), which activate different parts of the immune system. In particular, Th17 cells have been associated with the inflammatory response and iTreg cells with the regulation of the inflammatory response.

CD4+ T cells begin as naïve Th0 cells, which do not express a transcription factor. These cells are activated by antigen presentation and differentiate depending in the cytokines in the environment. In the presence of IL-6 or IL-21 and TGFβ Th0 cells differentiate into Th17 cells and express RORγt, IL-21 and IL-17. In the presence of IL-2 and TGFβ Th0 cells differentiate into iTreg cells and express Foxp3 and TGFβ. There also exist Th3 cells, which are TGFβ+Foxp3-. These cytokines and transcription factors regulate each other and their relationships can be visualized a as graph[Zhu 2010, Carrier 2007].



The Th17/iTreg network can be expressed as a set of boolean functions obtained from the known interactions among the cytokines and transcription factors. Cytokines are intrinsic if produced by the CD4+ T cell, and extrinsic if produced by other cells of the immune system. We will distinguish extrinsic cytokines by adding *e* at the end of the cytokine name.

```
library("BoolNetPerturb")

data(netTh17Treg)
netTh17Treg
#> Boolean network with 8 genes
#>
#> Involved genes:
#> IL2 RORGT STAT3 FOXP3 TGFB IL2e IL21e TGFBe
#>
#> Transition functions:
#> IL2 = (IL2e | (IL2 &  ! FOXP3)) &  ! STAT3
#> RORGT = (STAT3 & TGFB) &  ! FOXP3
#> STAT3 = (IL21e | STAT3 | RORGT) &  ! IL2
#> FOXP3 = (IL2 & (TGFB | FOXP3)) &  ! (STAT3 | RORGT)
#> TGFB = TGFBe | ((TGFB | FOXP3) &  ! STAT3 )
#> IL2e = IL2e
#> IL21e = IL21e
#> TGFBe = TGFBe
```

The function `getNetTopology()` can be used to obtain the topology of the network with interaction signs.

Labels and cell types

As the state of the network is updated using the functions, the network will reach a previously visited state called an attractor. Attractors can be steady states or cycles. The set of states that lead to an attractor is called the basin of the attractor. Attractors represent cell types or biological processes. The function `attractorToDataframe()` allows us to see the attractor as a dataframe

```
attr <- getAttractors(netTh17Treg)
attr.df <- attractorToDataframe(attr, Boolean = TRUE)
head(attr.df)
#>   attractor state IL2 RORGT STAT3 FOXP3 TGFB IL2e IL21e TGFBe
#> 1         1     1   0     0     0     0    0    0     0     0
#> 2         2     1   1     0     0     0    0    0     0     0
#> 3         3     1   0     0     1     0    0    0     0     0
#> 4         4     1   0     0     0     0    1    0     0     0
#> 5         5     1   1     0     0     0    0    1     0     0
#> 6         6     1   0     0     1     0    0    1     0     0
```

It is very important to verify that all the expected cell types appear in our attractors, if they are not present we might be missing interactions. It is also important to see if there are attractors that do not correspond to known cell types, as they may be predictions or show errors in the construction of the network. However, this can be complicated if the network is very large or has a large number of input nodes, as we may get multiple atractors that are biologicaly equivalent. A possible solution is to use known biological markers. If a cell type is charaterized by teh presence of absence of certain molecules we can use that information to create a Boolean function that will allow us to label the attractors.

For example, in the case of the Th17/Treg network, we will consider that an attractor corresponds to a cell type if both the master transcription factor and characteristic cytokine are active.

```
data("labelsTh17Treg")
labelsTh17Treg
#>   labels                    rules
#> 1    Th0  !(RORGT | FOXP3 | TGFB)
#> 2   Th17            RORGT & STAT3
#> 3   Treg             FOXP3 & TGFB
#> 4    Th3 TGFB & ! (RORGT | FOXP3)
#> 5 RORGT+          RORGT & ! STAT3
#> 6 FOXP3+           FOXP3 & ! TGFB
```

Using this rules we can label the attractors of the network using `labelAttractors()`. By default the labels of each state of a cyclic attractor are joined with "/". The function `labelState()` does the same for a single state.

```
labels <- labelAttractors(attr, labelsTh17Treg)
table(labels)
#> labels
#> RORGT+/Th3        Th0    Th0/Th0       Th17        Th3       Treg
#>          2          9          1          4          2          4
```

Robustness and plasticity in biological systems

Organisms develop in a changing world where they are exposed to intrinsic and extrinsic perturbations. Because of this perturbations, they need to be both resilient and adaptable, depending of the situation. This two behaviors coexist in all organisms, which suggests that there are common mechanism that underlie both robustness and plasticity.

Robustness is the capacity of an organism of maintaining its biological function in response to perturbations. A system is stable if it returns to the initial state after a perturbation, and plastic if it transitions to a new state. However, for studying robustness it is necessary to determine *what* function of the system is robust to *which* kind of perturbations[Wagner 2005].

Here we provide some tools for some of the most common perturbations that can be done to a Boolean network.

Target	Perturbation	Biological equivalent
Function	Fixed functions	Knock-out and over expression mutants, permanent changes in environment
Function	Transition table	Misconstruction of the network, small changes in regulation, evolvability
Dynamic	Updating	Time and hierarchy of biological processes.
Dynamic	State transitions	Transient biological behaivor.
State	Directed transient	Temporal changes in expression, transient environmental signals.
State	Stochastic	Biological stochastic processes.

Functions

Functions recapitulate the regulatory interactions and determine the dynamic of the Boolean regulatory network. They are limited by the topology, as nodes can only be directly influeced by their regulators. The changes in the functions of a network can be associated with multiple biological phenomena. Knock out and over expresion experiments, environmental factors, evolution, epigenetics and the intrinsic flexibility of the regulatory mechanisms all alter the iteractions of the regulatory network of a biological system.

Knock-out and over-expression

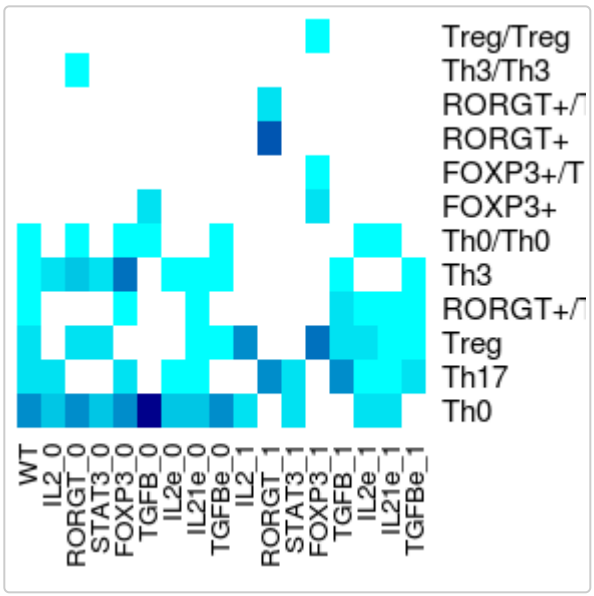
One possible kind of perturbation it to fix the value of the regulatory function to 0 or 1. The perturbation is equivalent to a knock-out or over-expression experiment. In this way it is possible to validate the model against known mutants. It can also be used to simulate the effect of conditional mutants of proteins that are hard to study in the wet lab, as the functions of the model only represent the effect of the mutation in the specific system. This can be useful if mutating the target protein is letal, as it can predict the effect of technically complicated conditional mutants.

We can use the BoolNetPerturb function `perturbNetworkFixedNodes()` to obtain the attractors and basins of the different fixed networks. By default, this method obtains all the single node knockouts and overexpressions. The function returns a dataframe where the rownames are the ocurrences of the attractor and each column corresponds to each mutant network. If the attractor cannot be found in the network it returns 0. If this function receives a set of labeling rules it will automaticaly label and group the attractors.

```
mutants <- perturbNetworkFixedNodes(netTh17Treg, label.rules = labelsTh17Treg)

mutants <- mutants[order(-mutants$WT),]  # order by WT in descending order
mutants[mutants==0] <- NA  # replace 0 for NA for plotting
colfunc <- colorRampPalette(c("cyan", "darkblue"))  # color scale
```

```
heatmap(data.matrix(mutants),
        Rowv=NA, Colv=NA,
        col= colfunc(10),scale="none")
```



Fixed environments

The state of a network usually depends in external factors that can be modeled as inputs. Fixing the value of the functions can also be used to study the effect of the environment in the differentiation and robustness of different cell types. Most cell-types are higly dependent on the signals in the environment. At the same time, not all combinations of inputs might be biological relevant, so limiting the environments can reduce the complexity of the model. This can be simulated by fixing the values of the inputs of the network according to the different environments [Monteiro 2015].

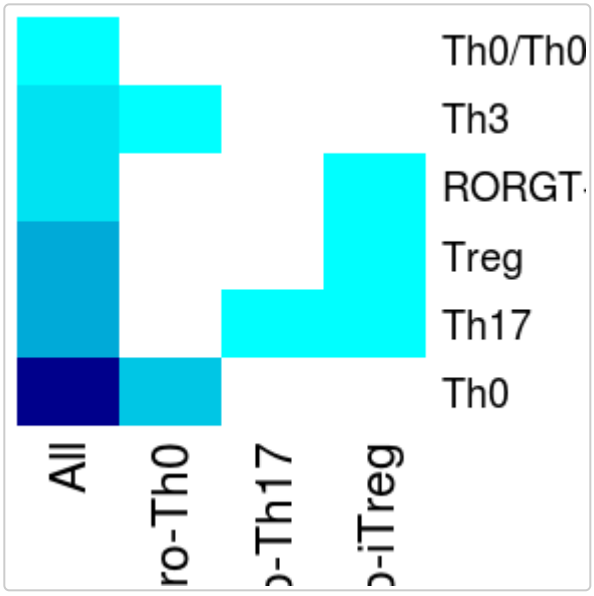
```
data("envTh17Treg")
envTh17Treg
#> $label
#> [1] "All"       "pro-Th0"   "pro-Th17"  "pro-iTreg"
#>
#> $nodes
#> $nodes[[1]]
#> [1] NA
#>
#> $nodes[[2]]
#> [1] "IL2e"  "IL21e" "TGFBe"
#>
#> $nodes[[3]]
#> [1] "IL2e"  "IL21e" "TGFBe"
#>
#> $nodes[[4]]
#> [1] "IL2e"  "IL21e" "TGFBe"
#>
#>
#> $values
#> $values[[1]]
```

```
#> [1] NA NA NA
#>
#> $values[[2]]
#> [1] 0 0 0
#>
#> $values[[3]]
#> [1] 0 1 1
#>
#> $values[[4]]
#> [1] 1 0 1
```

The BoolNetPerturb function `perturbNetworkFixedNodes()` can simulate multiple fixed genes if we pass them as a vector inside a list. In this case we will compare different micro-environments to determine which cell types we can expect in each environment. This fuction can also be used to simulate multiple mutants.

```
env.attr <- perturbNetworkFixedNodes(netTh17Treg, label.rules = labelsTh17Treg,
                                     genes  = envTh17Treg$nodes,
                                     values = envTh17Treg$value,
                                     names  = envTh17Treg$label)

env.attr <- env.attr[order(-env.attr$All),]  # order by WT in descending order
env.attr[env.attr==0] <- NA  # replace 0 for NA for plotting
colfunc <- colorRampPalette(c("cyan", "darkblue"))  # color scale
heatmap(data.matrix(env.attr),
        Rowv=NA, Colv=NA,
        scale="none", col= colfunc(10))
```



Transition table

We can simulate perturbations in the truth table using the `BoolNet` function `perturbNetwork()` and compare this results with the attractors of our WT network. This is equivalent to small changes in the regulation of a node.

Here we show the difference between the two sets of attractors. As `perturbNetwork()` randomly changes a value of the table it is a good idea to repeat the experiment multiple times (see BoolNet package vignette for more information).

```
# Get WT attractors
attr.df <- getAttractors(netTh17Treg)
attr.df <- attractorToDataframe(attr.df)

# Perturb net and get attractors
perturbed.net <- perturbNetwork(netTh17Treg, perturb="functions")
pertubed.attr <- getAttractors(perturbed.net)
pertubed.attr.df <- attractorToDataframe(pertubed.attr)

# Calculate number of attractors not in the original network
diff <- setdiff(attr.df$involvedStates, pertubed.attr.df$involvedStates)
length(diff)
#> [1] 6
```

Dynamic

The effect of the order in which the functions are evaluated can also be studied. Studying the robustness of the updating schema is useful for predicting the effect of temporal differences and developmental noise in the dynamic regulation of different proteins.

An other possible perturbation of the dynamic is to alter the sucessor states. Boolean regulatory networks are deterministic, the state in this time step determines the state in the next time step. However, it is possible to alter the successor, changing the transition graph. This can alter -or not- the trajectory of the simulation and change the final attractor. This is equivalent to developmental noise in the differentiation process of a cell. However, this kind of perturbations can also be considered a modification of the truth table.

Synchronous vs asynchronous

In a discrete regulatory network the value of a node n in $t + 1$ is a function of the values of its regulators in the time $t. However, we can update the value of the nodes in different ways. The two main schemas for updating are synchronous -where all functions are evaluated at the same time- or asynchronous -where each function is evaluated independently. If synchronous updates are used each state the transition space has only one successor. However, if asynchronous updates are used each state the transition space can have more than one successor. Complex attractors depend on the updating policy and are harder and more expensive to compute in the asynchronous case.

Biologicaly speaking, synchronous updates supose that all the processes happen at the same time, while asynchronous updates supose that the processes do not. If a process is faster than an other, or if there is a lag in its regulation, it may affect the dynamic of the regulatory logic.

The synchronous update may recover some cyclic attractors that are not recovered in the asynchronous update. To verify wether the attractors can be recovered in both update schema we can use the function `verifySyncronousVsAsyncronous()`

```
attr <- getAttractors(netTh17Treg)
verifySyncronousVsAsyncronous(netTh17Treg, attr, label.rules = labelsTh17Treg)
#>            sync.Freq async.Freq
```

```
#> RORGT+/Th3          2          NA
#> Th0                 9           9
#> Th0/Th0             1          NA
#> Th17                4           4
#> Th3                 2           2
#> Treg                4           4
#> Total              22          19
```

State

Boolean networks can be subjected to noice during the trayectory or once an attractor has been reached. This perturbations do not alter the wiring or updating of the network, but its state. This perturbations are transient, as the functions remain the same, it is only the state that is perturbed for a certain time, affecting its trajectory. This pertrubations can be transient changes in the value of a node caused by intrinsic noise or fluctuations in the microenvironment.

Biologicaly, this is equivalent to signals of the environment or intrinsic process of the cell that change the expression of an element for a certain time. For example, most drugs change the micro-environment of the organism while they are ingested, but once the treatment ends the perturbation ends. An other example of transient perturbations in biological systems is stochastic noise.

For example, suppose the state Th0, we can expose this cell to a temporary pro-Th17 environment and determine its trajectory using the `perturbState()` function.

```
perturbState(netTh17Treg, state = 0,
             genes=c("IL21e","TGFBe"),
             result = "trajectory")
#>   IL2 RORGT STAT3 FOXP3 TGFB IL2e IL21e TGFBe
#> 1   0     0     0     0    0    0     0     0
#> 2   0     0     0     0    0    0     1     1
#> 3   0     0     1     0    1    0     0     0
#> 4   0     1     1     0    0    0     0     0
#> 5   0     0     1     0    0    0     0     0
```

This function can also be used to simulate stochastic noise, asby default it perturbs a random state and gene. This function is meant to complement the `BoolNet` function `perturbTrajectories()` that measures the robustness of a network to noise. On the other hand `perturbState()` is much more directed, as it focuses in directed perturbations.

One way to measure the stability is to determine the one-state neighbors [Huang? 2018]. Thu function `cellFateMap()` we transiently perturb all the states in all the attractors one by one, and determine if they stay in the same attractor or transition to a new one, creating a cell fate map of the system.

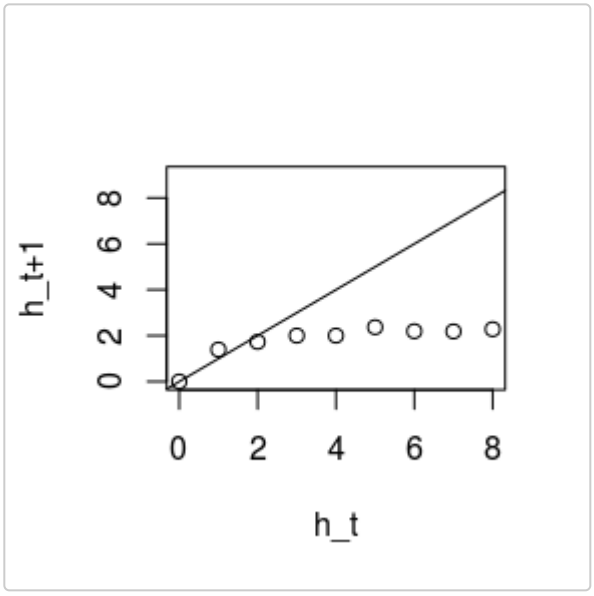
```
cellfate <- cellFateMap(netTh17Treg, label.rules = labelsTh17Treg)
head(cellfate)
#>   initial      final genes values
#> 1    Treg       Th17   IL2      0
#> 2    Treg       Treg RORGT      1
#> 3    Treg       Th17 STAT3      1
#> 4    Treg       Treg FOXP3      0
```

```
#> 5    Treg       Treg  TGFB      0
#> 6    Treg RORGT+/Th3  IL2e      0
```

Derrida curves

Derrida curves are used to determine the dynamic behaviour of Boolean networks. We take two states with Hamming distance h in the time t, and determine the hamming distance in time $t+1$. This process is repeated. If the network is chaotic, the curve tends to be over the diagonal. If the network is ordered it will be below the diagonal.

```
res.der <- derrida(netTh17Treg)
plot(x=names(res.der), y=res.der,
     ylim = c(0,length(res.der)),
     xlab = "h_t", ylab = "h_t+1")
abline(0,1)
```



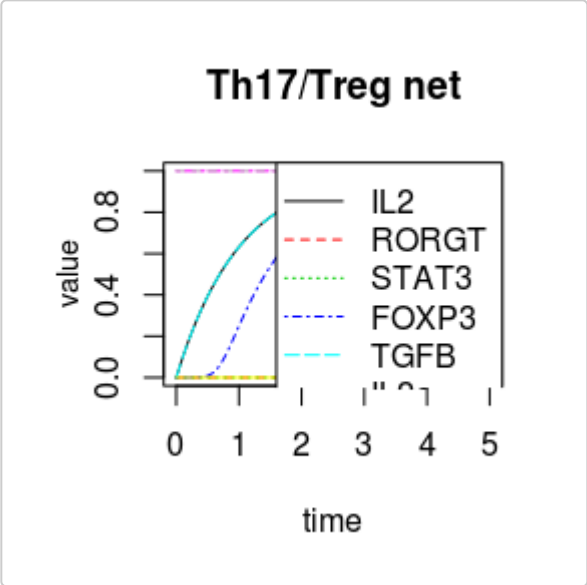
Connect to other packages

The library `BoolNetPerturb` was conceived as an extension of `Boolnet`. It is possible to plot the network topology and other dataframes with the R package `igraph`. This dataframe can also be used to import and export to other resources like the python library `networkx` or to the software `Cytoscape`. It is possible to export the network functions as an SBML file using the `BoolNet` function `toSBML()`.

Boolean to ODE networks

It is possible to translate a Boolean network to a ODE network using logistic functions (see refs). The function `booleanToODE()` translate a BoolNet network to an ODE model using Zadeh or probabilistic logic and the equation proposed by Sanchez-Corrales et al., 2010 or Villarreal et al., 2012. This process can also be used to verify that the model is robust to a translation to a continuous system.

```
library(deSolve)
#>
#> Attaching package: 'deSolve'
#> The following object is masked from 'package:graphics':
#>
#>     matplot
net.ode <- booleanToODE(netTh17Treg, keep.input = TRUE)
state <- validateState(c(0,0,0,0,0,1,0,1), netTh17Treg$genes)
out <- ode(func = net.ode$func,
     parms = net.ode$parameters,
     y = state,
     times = seq(0, 5, 0.1))
matplot(out, type="l", ylab="value", main="Th17/Treg net")
```



References