Swiss Institute of Bioinformatics

boolSim

User manual

Amel Bekkar and Julien Dorier
January 2017

1 Introduction

boolSim, also called *genYsis*, is a software written by Abhishek Garg[1]. This version of *boolSim* was modified by Julien Dorier. *boolSim* can be used in the following ways:

- To find attractors of a network, using synchronous or asynchronous network dynamics[1] (section 2 “Attractors”).
- To perform experiments [1]. Experiments consist in a set of consecutive perturbations that should be applied to the network. *boolSim* will evaluate all attractors found for each perturbation, as well as a reachability graph, which specifies transitions between attractors obtained with consecutive perturbations. The dynamics of the network is asynchronous (section 3 “Experiments”).
- To find reachable states from a user defined set of initial states (section 4 “Reachable states”)

It is also possible to perform operations (union, intersection, difference) on sets of states using the tool `boolSim_setutils` (section 5 “Operations on sets of states”).

2 Attractors

2.1 Usage

To evaluate all attractors of a network:

```
$ boolSim -f network_file -p update_mode -o output
```

`-f network_file` is the network in *boolSim* or SBML-qual format (see “2.2 Input file formats”).
`-p update_mode` specify network dynamics:

`-p 1` → synchronous
`-p 2` → asynchronous (slow, do not use)
`-p 3` → asynchronous (fast)

`-o output` is an optional output filename. If not given, the network name will be used.

boolSim will evaluate all attractors of the network, using the specified network dynamics, and output each attractor to a separate file (see “2.3.1 Attractors”).

2.2 Input file formats

2.2.1 SBML-qual network file format

If the network filename has .sbml or .xml extension, it will be interpreted as a SBML-qual file format[2, 3].

2.2.2 BoolSim network file format

A network is defined by a list of interactions. Interactions are specified in a text file, with one interaction per line. Positive interactions (activators) are denoted by ->. To define Node1 as an activator of Node2:

```
Node1 -> Node2
```

Negative interactions (inhibitors) are denoted by -|. To define Node1 as an inhibitor of Node2:

```
Node1 -| Node2
```

Note that a space character must appear on each side of the interaction sign (-> or -|). Except around the interaction sign, space character should not be used.
Multiple activators (e.g. Node1 and Node2) acting on the same node (e.g. Node3) will be combined with OR Boolean operator:

```
Node1 -> Node3
Node2 -> Node3
```

Similarly, multiple inhibitors will be combined with OR operator:

```
Node1 -| Node3
Node2 -| Node3
```

If a node has activators (e.g. Node1 and Node2) and inhibitors (e.g. Node3 and Node4), the strong inhibitor rule will be used, i.e. the target node (Node5) will be activated if and only if at least one activator has value 1 and all inhibitors have value 0. Otherwise it will be inhibited

```
Node1 -> Node5
Node2 -> Node5
Node3 -| Node5
Node4 -| Node5
```

A more precise definition can be found in [1].
In addition to activation and inhibition by a single node, the left hand part of the interaction can contain & (AND) and ^ (NOT) Boolean operators. For example, to specify that Node1 AND NOT Node3 AND Node3 inhibit Node4:

```
Node1&^Node2&Node3 -| Node4
```

As in the case with activation and inhibition by a single node, multiple activators will be combined with OR Boolean operator:

```
Node1&^Node2 -> Node4
^Node3 -> Node4
```

i.e. Node4 will be activated if (Node1 has value 1 AND Node2 has value 0) OR Node3 has value 0.
Multiple inhibitors are also combined with OR, and a combination of inhibitors and activators will again be combined with the strong inhibitor rule. In the following example

```
Node1&^Node2 -> Node6
^Node3 -> Node6
Node4&Node5 -| Node6
```

Node6 will be activated if at least one activator has value 1 (i.e. (Node1 has value 1 AND Node2 has value 0) OR Node3 has value 0) AND all inhibitors have value 0 (i.e. Node4 has value 0 OR Node5 has value 0).
Example: T-helper cell differentiation network[4]

```
TCR -> NFAT
NFAT -> IFN-G
IFN-B -> IFN-BR
IFN-BR -> STAT1
IL-18 -> IL-18R
IL-18R -> IRAK
IRAK -> IFN-G
SOCS1 -| IL-4R
SOCS1 -| JAK1
IL-12 -> IL-12R
IL-12R -> STAT4
STAT4 -> IFN-G
T-bet -> T-bet
T-bet -| GATA3
T-bet -> SOCS1
T-bet -> IFN-G
IFN-G -> IFN-GR
IFN-GR -> JAK1
JAK1 -> STAT1
STAT1 -> T-bet
STAT1 -> SOCS1
STAT1 -| IL-4
IL-4 -> IL-4R
IL-4R -> STAT6
STAT6 -> GATA3
STAT6 -| IL-12R
STAT6 -| IL-18R
GATA3 -> GATA3
GATA3 -| STAT4
GATA3 -| T-bet
GATA3 -> IL-4
GATA3 -> IL-10
IL-10 -> IL-10R
IL-10R -> STAT3
STAT3 -| IFN-G
```

2.3 Output file formats

2.3.1 Attractors file format

Each attractor is written in a separate file. Each file contains a header line, which specifies what appears in each column. The first column contains node names. For an attractor with only one state, a second column contains node states (0 or 1), as in the following example:

```
Gene Name/State No.   S_1
GATA3                   1
STAT6                   1
Tbet                    0
IFNbR                   0
IFNb                    0
IFNg                    0
IRAK                    0
NFAT                    0
STAT3                   1
STAT4                   0
IFNgR                   0
IL10                    1
IL10R                   1
IL12R                   0
IL12                    0
IL18R                   0
IL18                    0
IL4                     1
STAT1                   0
IL4R                    1
SOCS1                   0
```

```
JAK1                     0
TCR                      0
```

An attractor with more than one state may have more than two columns and node state may contain a special value 2, which means that the node can take both values 0 and 1. For each entry with value 2, the column should be duplicated, and the node value replaced by 0 in the first copy, and by 1 in the second copy. A column with n entries with value 2 must be expanded to 2^n states.
Consider the following example:

```
Gene Name/State No.   S_1
GATA3                   1
STAT6                   2
Tbet                    0
IFNbR                   2
IFNb                    0
IFNg                    2
```

Three nodes (STAT6, IFNbR and IFNg) have value 2, and column S_1 must be expanded to 8 states (2^3):

```
GATA3                    1  1  1  1  1  1  1  1
STAT6                    0  0  0  0  1  1  1  1
Tbet                     0  0  0  0  0  0  0  0
IFNbR                    0  0  1  1  0  0  1  1
IFNb                     0  0  0  0  0  0  0  0
IFNg                     0  1  0  1  0  1  0  1
```

A more complex example with multiple columns:

```
Gene Name/State No.   S_1    S_2    S_3
GATA3                   1      0      1
STAT6                   2      1      0
Tbet                    1      2      0
IFNbR                   2      0      0
IFNb                    0      1      1
IFNg                    1      0      1
```

Column S_1 has two entries with value 2, and corresponds to 4 states (2^2). Column S_2 has one entry with value 2 and corresponds to 2 states (2^1). Column S_3 has only values 0 and 1, and should not be expanded (2^0):

```
Gene Name/State No.   S_1 S_1 S_1 S_1 S_2 S_2 S_3
GATA3                   1   1   1   1   0   0   1
STAT6                   0   0   1   1   1   1   0
Tbet                    1   1   1   1   0   1   0
IFNbR                   0   1   0   1   0   0   0
IFNb                    0   0   0   0   1   1   1
IFNg                    1   1   1   1   0   0   1
```

2.4 Example

To illustrate how to use *boolSim*, we will use the T-helper model from Mendoza and Xenarios[4]. This network is given in *boolSim* format in the file `examples/Th_2006.net`. It contains 23 nodes and 35 edges:

```
GATA3 -> GATA3
STAT6 -> GATA3
Tbet -| GATA3
IFNb -> IFNbR
```

```
IRAK -> IFNg
NFAT -> IFNg
STAT3 -| IFNg
STAT4 -> IFNg
Tbet -> IFNg
IFNg -> IFNgR
GATA3 -> IL10
IL10 -> IL10R
IL12 -> IL12R
STAT6 -| IL12R
IL18 -> IL18R
STAT6 -| IL18R
GATA3 -> IL4
STAT1 -| IL4
IL4 -> IL4R
SOCS1 -| IL4R
IL18R -> IRAK
IFNgR -> JAK1
SOCS1 -| JAK1
TCR -> NFAT
STAT1 -> SOCS1
Tbet -> SOCS1
IFNbR -> STAT1
JAK1 -> STAT1
IL10R -> STAT3
GATA3 -| STAT4
IL12R -> STAT4
IL4R -> STAT6
GATA3 -| Tbet
STAT1 -> Tbet
Tbet -> Tbet
```

To find all attractors of the T-helper network using asynchronous dynamics:

```
$ ./boolSim -f examples/Th_2006.net -p 3 -o attractor
```

Here we ask *boolSim* to find attractors of the network written in the file `examples/Th_2006.net` (option `-f`) with asynchronous dynamics (option `-p 3`) and to use the output name `attractor` (option `-o`). *boolSim* will write to the standard output the number of states in each attractor found, as well as where it is saved:

```
geneDbOrg=23 geneDbReduced=13
Attractor 1 ::: number of states = 1
Attractor 2 ::: number of states = 1
Attractor 3 ::: number of states = 1
States of the attractor 1 are written in the file attractor_1
States of the attractor 2 are written in the file attractor_2
States of the attractor 3 are written in the file attractor_3
```

In this example, *boolSim* found three attractors of size 1 (lines 3 to 5), which are saved in the files `attractor_1`, `attractor_2` and `attractor_3` (lines 6 to 8).

The first attractor is saved in `attractor_1`:

```
Gene Name/State No. S_1
GATA3                 1
STAT6                 1
Tbet                  0
IFNbR                 0
IFNb                  0
```

The second attractor is saved in `attractor_2`:

```
Gene Name/State No  S_1
GATA3                 0
STAT6                 0
Tbet                  1
IFNbR                 0
IFNb                  0
```

The second attractor is saved in `attractor_3`:

```
Gene Name/State No. S_1
GATA3                 0
STAT6                 0
Tbet                  0
IFNbR                 0
IFNb                  0
```

```
IFNg                    0
IRAK                    0
NFAT                    0
STAT3                   1
STAT4                   0
IFNgR                   0
IL10                    1
IL10R                   1
IL12R                   0
IL12                    0
IL18R                   0
IL18                    0
IL4                     1
STAT1                   0
IL4R                    1
SOCS1                   0
JAK1                    0
TCR                     0
```

```
IFNg                    1
IRAK                    0
NFAT                    0
STAT3                   0
STAT4                   0
IFNgR                   1
IL10                    0
IL10R                   0
IL12R                   0
IL12                    0
IL18R                   0
IL18                    0
IL4                     0
STAT1                   0
IL4R                    0
SOCS1                   1
JAK1                    0
TCR                     0
```

```
IFNg                    0
IRAK                    0
NFAT                    0
STAT3                   0
STAT4                   0
IFNgR                   0
IL10                    0
IL10R                   0
IL12R                   0
IL12                    0
IL18R                   0
IL18                    0
IL4                     0
STAT1                   0
IL4R                    0
SOCS1                   0
JAK1                    0
TCR                     0
```

Attractor 1 corresponds to the biological cell type Th2 (GATA3 state is 1), attractor 2 to Th1 (Tbet state is 1), and attractor 3 to Th0.

3 Experiments

3.1 Usage

To evaluate all attractors of a network under specific perturbations, as well as reachability graph:

```
$ boolSim -f network_file -e experiment_file -o output
```

`-f network_file` is the network in *boolSim* or SBML-qual format (see “2.2 Input file formats”).
`-e experiment_file` is a file describing the experiment, in *boolSim* experiment format (see ”3.2.1 Experiment file format”).
`-o output` is an optional output filename. If not given, the network name is used.

For each stage of the experiment, *boolSim* will evaluate all attractors using asynchronous network dynamics and output each attractor to a separate file (see “2.3.1 Attractors”). In addition, it will evaluate the reachability between attractors in consecutive stages, and output the resulting reachability graph to file (see “3.3.1 Reachability graph”).

3.2 Input file formats

3.2.1 Experiment file format

Experiments are used to simulate the effect of a single or multiple consecutive perturbations on the network. An experiment consists of one or more stages, each defined by a perturbation of the network. A perturbation can be any combination of node over-expression (node always set to 1) and/or node knock-out (node always set to 0). The experiment file has the following format: The first line must contain the number of stages to perform. Stages are then defined consecutively. Each stage starts with a line containing three integer numbers, which correspond respectively to the number of nodes we want to knock-out (fixed at 0), the number of nodes to over-express (fixed at 1) and the number of nodes unconstrained (useful when we want to release nodes we fixed at 0 or 1 in a previous stage). Note that a node perturbed at a given stage will be perturbed until the end of the experiment, unless specifically set to unconstrained. The next lines list in the corresponding order (fixed at 0, fixed at 1 and unconstrained) the nodes of interest. This is repeated for each stage.
As an example, let us consider the following experiment for the T-helper model:

```
3                      ⟵ 3 stages
0  0  0                ⟵ Stage 1: 0 node fixed to 0, 0 node fixed to 1, 0 unconstrained node
```

`0 1 0`	⟵ Stage 2: 0 node fixed to 0, 1 node fixed to 1, 0 unconstrained node
`IFNg`	⟵ Stage 2: IFNg is fixed to state 1
`0 1 1`	⟵ Stage 3: 0 node fixed to 0, 1 nodes fixed to 1, 1 unconstrained node
`IL4`	⟵ Stage 3: IL4 is fixed to state 1
`IFNg`	⟵ Stage 3: IFNg is unconstrained

This file defines an experiment with 3 stages. In the first stage, the network is not perturbed. In the second stage, IFNg is over-expressed and in the third stage, IL4 is over-expressed. Note that IFNg had to be unconstrained in stage 3, otherwise it would have been stayed over-expressed.
Another example with only one stage:

`1`	⟵ 1 stage
`1 2 0`	⟵ Stage 1: 0 node fixed to 0, 2 nodes fixed to 1, 0 unconstrained node
`IL12`	⟵ Stage 1: IL12 is fixed to state 0
`IL4`	⟵ Stage 1: IL4 is fixed to state 1
`IFNg`	⟵ Stage 1: IFNg is fixed to state 1

This file defines an experiment with one stage where IL12 is knocked-out and both IL4 and IFNg are over-expressed. In this case, since there is only one stage, *boolSim* will only output the attractors resulting from the experiment without any reachability analysis.

3.3 Output file formats

3.3.1 Reachability graph file format

When *boolSim* is used to perform experiments, it outputs a file containing a reachability graph together with names of file where attractors are saved. If an output filename was given to *boolSim*, this file will be saved as `reach_<output>.txt`, otherwise it will be saved as `reach_<network_filename>.txt`.
For each stage of the experiment, the file contains a section that starts with the filenames of each attractor found in the corresponding stage. Except for the first stage, it is followed by the reachability from attractors found in the previous stage to attractors found in the current stage. Reachability from attractor `i` in stage `n-1` to attractor `j` in stage `n` is written as:
`SS_<n-1>_<i>  ---->  SS_<n>_<j>`
A sample reachability file obtained for a two stages experiment on the T-helper model, with a first stage unperturbed, and second stage with IFNg state fixed to 1 (over-expressed):

```
###### unperturbed network #################

States of the attractor 1 are written in the file output_SS_1_1.txt
States of the attractor 2 are written in the file output_SS_1_2.txt
States of the attractor 3 are written in the file output_SS_1_3.txt
##############################################

#######  IFNg over-expressed #######

States of the attractor 1 are written in the file output_SS_2_1.txt
States of the attractor 2 are written in the file output_SS_2_2.txt
***** Reachability Analysis *****
SS_1_1  ---->  SS_2_1
SS_1_2  ---->  SS_2_1
SS_1_3  ---->  SS_2_2
```

It starts with a section for the first stage (unperturbed), specifying that attractors obtained in this stage are saved in files `output_SS_1_*.txt`. The output concerning the second stage starts with

```
#######  IFNg over-expressed #######
```

followed by filenames for attractors obtained in this stage (`output_SS_2_*.txt`). The section concerning the second stage terminates with the reachability analysis. We can see that in this case, both attractors 1 and 2 from stage 1 will flow into attractor 1 after IFNg over-expression, while attractor 3 from stage 1 will flow into attractor 2 from stage 2 after IFNg over-expression.

3.4 Example

We will use the experiment file `examples/Th_2006_experiment.exp`, which contains the experiment described previously in “3.2.1 Experiment file format”:

```
$ cat examples/Th_2006_experiment.exp
3
0 0 0
0 1 0
IL4
0 1 1
IFNg
IL4
```

To evaluate attractors and reachability graph of the T-helper network with the experiment defined in file `examples/Th_2006_experiment.exp`:

```
$ ./boolSim -f examples/Th_2006.net -e examples/Th_2006_experiment.exp
```

Here we use the network file `examples/Th_2006.net` (option `-f)` and experiment file `examples/Th_2006_experiment.exp` (option `-e)`. *boolSim* will write its progress in the standard output, but more importantly, it will specify the location of the output file (last line):

```
processing experiment file level 0
compsteady
extract states
processing experiment file level 1
-- reachability --
SS_1_1  ---->  SS_2_1
SS_1_2  ---->  SS_2_2
SS_1_3  ---->  SS_2_1
processing experiment file level 2
-- reachability --
SS_2_1  ---->  SS_3_1
SS_2_2  ---->  SS_3_2
###### unperturbed network ##################
States of the attractor 1 are written in the file Th_2006_SS_1_1.txt
States of the attractor 2 are written in the file Th_2006_SS_1_2.txt
States of the attractor 3 are written in the file Th_2006_SS_1_3.txt
#######  IL4, over-expressed  #######
States of the attractor 1 are written in the file Th_2006_SS_2_1.txt
States of the attractor 2 are written in the file Th_2006_SS_2_2.txt
#######  IFNg, over-expressed  #######
States of the attractor 1 are written in the file Th_2006_SS_3_1.txt
States of the attractor 2 are written in the file Th_2006_SS_3_2.txt
##############################################
Reachablity results are written in the file reach_Th_2006.txt
```

Here, the reachability graph was saved in `reach_Th_2006.txt`. By default, *boolSim* use the prefix of the network file for output file name. The output file `reach_Th_2006.txt` specifies the number of attractors found in each stage, as well as where they are saved on disk, and reachability between them:

```
$ cat reach_Th_2006.txt
##############################################

###### unperturbed network ##################

States of the attractor 1 are written in the file Th_2006_SS_1_1.txt
States of the attractor 2 are written in the file Th_2006_SS_1_2.txt
States of the attractor 3 are written in the file Th_2006_SS_1_3.txt
##############################################

#######  IL4, over-expressed  #######

States of the attractor 1 are written in the file Th_2006_SS_2_1.txt
States of the attractor 2 are written in the file Th_2006_SS_2_2.txt
***** Reachability Analysis *****
SS_1_1  ---->  SS_2_1
SS_1_2  ---->  SS_2_2
SS_1_3  ---->  SS_2_1
##############################################

#######  IFNg, over-expressed  #######

States of the attractor 1 are written in the file Th_2006_SS_3_1.txt
States of the attractor 2 are written in the file Th_2006_SS_3_2.txt
***** Reachability Analysis *****
SS_2_1  ---->  SS_3_1
SS_2_2  ---->  SS_3_2
```

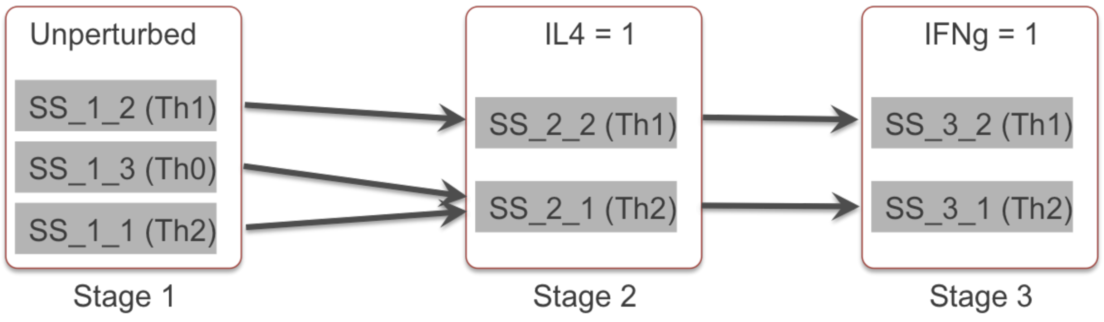
Here, attractors obtained in first stage (unperturbed network) can be found in `Th_2006_SS_1_*.txt`, attractors obtained in second stage (IL4 over-expressed) in `Th_2006_SS_2_*.txt`, and attractors obtained in third stage (IFNG over-expressed) in `Th_2006_SS_3_*.txt`. Reachability between attractors is given the lines

```
SS_1_1  ---->  SS_2_1
SS_1_2  ---->  SS_2_2
SS_1_3  ---->  SS_2_1
```

and

```
SS_2_1  ---->  SS_3_1
SS_2_2  ---->  SS_3_2
```

This information can be used to reconstruct the following attractors' reachability graph:



4 Reachable states

4.1 Usage

To evaluate all reachable states from a set of initial states:

```
$ boolSim [-n nb_iterations] -f network_file -p update_mode -i initial_states_file -o
output
```

`-f network_file` is the network in *boolSim* or SBML-qual format (see "2.2 Input file formats").
`-p update_mode` specify network dynamics:

- `-p 1` → synchronous
- `-p 2` → asynchronous (slow, do not use)

`-p 3` → asynchronous (fast)

`-i initial_states_file` is a file with the set of initial states (see “4.2.1 Set of states”).
`-o output` is an optional output filename. If not given, the network name is used.
`-n nb_iterations` is an optional maximum number of iterations (i.e. time steps).
boolSim will evaluate all states reached by the network starting from all initial states, using the specified network dynamics, and output the resulting set of reached states (see “4.3.1 Set of states”). If a maximum number of iterations is specified (`-n`), only the states reachable within `nb_iterations` time steps will be evaluated.
Note: the set of reached states generated by *boolSim* always contain the set of initial states.

4.2 Input file formats

4.2.1 Set of states

Sets of initial states are defined using a space-separated or tab-separated values file format. Separators can be either spaces or tabs, and consecutive separators are treated as one separatore. The first column contains node names. Subsequent columns contain node states (0 or 1), possibly compressed using the special value 2, which means that the node can take both values 0 and 1.
For example:

```
GATA3 1 0 1
STAT6 2 1 0
Tbet  1 2 0
IFNbR 2 0 0
IFNb  0 1 1
IFNg  1 0 1
```

Column S_1 has two entries with value 2, and corresponds to 4 states (2^2). Column S_2 has one entry with value 2 and corresponds to 2 states (2^1). Column S_3 has only values 0 and 1, and should not be expanded (2^0):

```
GATA3 1 1 1 1 0 0 1
STAT6 0 0 1 1 1 1 0
Tbet  1 1 1 1 0 1 0
IFNbR 0 1 0 1 0 0 0
IFNb  0 0 0 0 1 1 1
IFNg  1 1 1 1 0 0 1
```

Notes:

- The file can optionally start with a header “Gene Name/State No. S_1 S_2 ...”
- Attractor file format (see “2.3.1 Attractors”) is compatible with this format, therefore attractors obtained with *boolSim* can be directly used.
- Network nodes that do not appear in this file are considered as not fixed (value 2).
- Empty files are interpreted as empty set (no state).

4.3 Output file formats

4.3.1 Set of states

The file starts with a header line, which specifies what appears in each column. The first column contains node names. Subsequent column contains node states (0 or 1), possibly compressed using the special value 2, which means that the node can take both values 0 and 1. For each entry with value 2, the column should be duplicated, and the node value replaced by 0 in the first copy, and by 1 in the second copy. A column with n entries with value 2 should therefore be expanded to 2^n states. For example:

```
Gene Name/State No.   S_1   S_2   S_3
GATA3                   1     0     1
STAT6                   2     1     0
Tbet                    1     2     0
```

```
IFNbR                     2      0      0
IFNb                      0      1      1
IFNg                      1      0      1
```

Column S_1 has two entries with value 2, and corresponds to 4 states (2^2). Column S_2 has one entry with value 2 and corresponds to 2 states (2^1). Column S_3 has only values 0 and 1, and should not be expanded (2^0):

```
Gene Name/State No.   S_1 S_1 S_1 S_1 S_2 S_2 S_3
GATA3                   1   1   1   1   0   0   1
STAT6                   0   0   1   1   1   1   0
Tbet                    1   1   1   1   0   1   0
IFNbR                   0   1   0   1   0   0   0
IFNb                    0   0   0   0   1   1   1
IFNg                    1   1   1   1   0   0   1
```

Notes:

- An empty file corresponds to the empty set (no states).
- This format is the same as the format used to output attractors (see "2.3.1 Attractors").

4.4 Example

As initial states, we will use the state defined in `examples/Th_2006_initial_states.txt`:

```
$ cat examples/Th_2006_initial_states.txt
GATA3 1
STAT6 1
Tbet 0
IFNbR 0
IFNb 0
IFNg 0
IRAK 0
NFAT 0
STAT3 1
STAT4 0
IFNgR 1
IL10 1
IL10R 1
IL12R 0
IL12 0
IL18R 0
IL18 0
IL4 1
STAT1 0
IL4R 1
SOCS1 0
JAK1 0
TCR 0
```

To evaluate the set of states reached by T-helper network with asynchronous dynamics:

```
$ ./boolSim -f examples/Th_2006.net -p 3 -i examples/Th_2006_initial_states.txt
```

Here we use the network file `examples/Th_2006.net` (option `-f`) with asynchronous dynamics (option `–p 3`) and the initial state defined in `examples/Th_2006_initial_states.txt` (option `-i`). *boolSim* will write to standard output the number of states reached, as well as where the set of reached states is saved:

```
geneDbOrg=23 geneDbReduced=22
constructing BDD
BDD construction done
```

```
Initial state creading bdd
 Initial state size: 17 states
 Evaluating reachable states
 Number of reached states: 104
Reached states are written in the file Th_2006_reached.txt
```

In this example, the 104 reached states are saved in the file `Th_2006_reached.txt`:

```
$ cat Th_2006_reached.txt
Gene Name/State No.   S_1   S_2   S_3   S_4   S_5   S_6   S_7
GATA3                   1     1     1     1     1     1     1
STAT6                   2     2     2     2     2     2     2
Tbet                    0     0     0     0     0     0     0
IFNbR                   0     0     0     0     0     0     0
IFNb                    0     0     0     0     0     0     0
IFNg                    0     0     0     0     0     0     0
IRAK                    0     0     0     0     0     0     0
NFAT                    0     0     0     0     0     0     0
STAT3                   1     1     1     1     1     1     1
STAT4                   0     0     0     0     0     0     0
IFNgR                   0     0     0     1     1     1     1
IL10                    1     1     1     1     1     1     1
IL10R                   1     1     1     1     1     1     1
IL12R                   0     0     0     0     0     0     0
IL12                    0     0     0     0     0     0     0
IL18R                   0     0     0     0     0     0     0
IL18                    0     0     0     0     0     0     0
IL4                     2     2     2     2     2     2     2
STAT1                   0     0     1     0     0     1     1
IL4R                    2     2     2     2     2     2     2
SOCS1                   0     1     2     0     1     0     1
JAK1                    2     0     2     2     0     1     2
TCR                     0     0     0     0     0     0     0
```

To evaluate the set of states reached after only one iteration:

```
$ ./boolSim -n 1 -f examples/Th_2006.net -p 3 -i examples/Th_2006_initial_states.txt
```

boolSim will write to standard output the number of states reached, as well as where the set of reached states is saved:

```
geneDbOrg=23 geneDbReduced=13
constructing BDD
BDD construction done
Initial state creading bdd
 Initial state size: 1 states
 Evaluating reachable states
 Number of reached states: 3
Reached states are written in the file Th_2006_reached.txt
```

In this example, the 3 reached states are saved in the file `Th_2006_reached.txt`:

```
$ cat Th_2006_reached.txt
Gene Name/State No.   S_1   S_2
GATA3                   1     1
STAT6                   1     1
Tbet                    0     0
IFNbR                   0     0
IFNb                    0     0
IFNg                    0     0
```

```
IRAK                    0     0
NFAT                    0     0
STAT3                   1     1
STAT4                   0     0
IFNgR                   0     1
IL10                    1     1
IL10R                   1     1
IL12R                   0     0
IL12                    0     0
IL18R                   0     0
IL18                    0     0
IL4                     1     1
STAT1                   0     0
IL4R                    1     1
SOCS1                   0     0
JAK1                    0     2
TCR                     0     0
```

5 Operations on sets of states

5.1 Usage

To evaluate all attractors of a network:

```
$ boolSim_setutils -o output operation file1 file2 file3 ...
```

`file1, file2, file 3, ...` are files describing sets of states (see “4.2.1 Set of states”).
`operation` specify the operation to apply on the sets of states. It can take the following values:

`union`	→ output all states that are in `file1` OR in `file2` OR in `file3` ...
`intersection`	→ output all states that are in `file1` AND in `file2` AND in `file3` ...
`difference`	→ output all states that are in `file1` AND NOT in `file2` AND NOT in `file3` ...
`partition`	→ partition the set of all possible states into disjoints sets formed by all intersections of input sets of states. Output the number of states in each set.

`-o output` is an optional output filename. If not given, output to standard output.

`boolSim_setutils` will apply the operation on the input sets and output the resulting set of states (see “4.3.1 Set of states”).

Notes:

- Empty files are allowed and correspond to empty sets.
- Nodes that do not appear in one of the files (but appears in at least another file) are considered as not fixed (value 2).

5.2 Example

We will work with the 3 sets of states defined in `examples/Th_2006_set1.txt`, `examples/Th_2006_set1.txt` and `examples/Th_2006_set1.txt`:

```
$ cat examples/Th_2006_set1.txt
GATA3  1  2
STAT6  0  1
Tbet  0  1
IFNbR  0  1
IFNb  0  1
IFNg  0  1
IRAK  1  2
NFAT  0  1
STAT3  0  0
STAT4  0  0
```

```
IFNgR 1 0
IL10 1 0
IL10R 1 0
IL12R 0 0
IL12 0 1
IL18R 0 1
IL18 0 1
IL4 1 0
STAT1 0 2
IL4R 1 1
SOCS1 0 0
JAK1 0 1
TCR 0 1
```

```
$ cat examples/Th_2006_set2.txt
GATA3 1
STAT6 1
Tbet 0
IFNbR 0
IFNb 0
IFNg 0
IRAK 0
NFAT 0
STAT3 1
STAT4 0
IFNgR 1
IL10 1
IL10R 1
IL12R 0
IL12 0
IL18R 0
IL18 0
IL4 1
STAT1 0
IL4R 1
SOCS1 0
JAK1 0
TCR 0
```

```
$ cat examples/Th_2006_set3.txt
GATA3 1
STAT6 1
Tbet 0
IFNbR 0
IFNb 0
IFNg 0
IRAK 0
NFAT 0
STAT3 1
STAT4 0
IFNgR 1
IL10 1
IL10R 1
IL12R 0
IL12 0
IL18R 0
IL18 0
IL4 1
STAT1 0
IL4R 1
```

```
SOCS1 0
JAK1 0
```

Note that `examples/Th_2006_set3.txt` does not contain node TCR.
To evaluate the union of all states and save the result to file `union_sets123.txt` (option `-o`):

```
$ boolSim_setutils -o union_sets123.txt union examples/Th_2006_set1.txt \
  examples/Th_2006_set2.txt  examples/Th_2006_set3.txt
```

The number of states in each file and in the resulting set of states is written to standard output:

```
file1:  9 states
file2:  1 states
file3:  2 states
result:  11 states (compressed in 4 patterns)
```

In this example file 1 (`examples/Th_2006_set1.txt`) contains 9 states, file 2 (`examples/Th_2006_set2.txt`) contains 1 state and file 3 (`examples/Th_2006_set3.txt`) contains 2 states (because TCR is missing, and therefore can take any value 0 or 1). The union of these three sets of states contains 11 states, and is saved in the file `union_sets123.txt`:

```
$ cat union_sets123.txt
Gene Name/State No.   S_1   S_2   S_3   S_4
GATA3                   0     1     1     1
IFNb                    1     0     0     1
IFNbR                   1     0     0     1
IFNg                    1     0     0     1
IFNgR                   0     1     1     0
IL10                    0     1     1     0
IL10R                   0     1     1     0
IL12                    1     0     0     1
IL12R                   0     0     0     0
IL18                    1     0     0     1
IL18R                   1     0     0     1
IL4                     0     1     1     0
IL4R                    1     1     1     1
IRAK                    2     0     1     2
JAK1                    1     0     0     1
NFAT                    1     0     0     1
SOCS1                   0     0     0     0
STAT1                   2     0     0     2
STAT3                   0     1     0     0
STAT4                   0     0     0     0
STAT6                   1     1     0     1
TCR                     1     2     0     1
Tbet                    1     0     0     1
```

To evaluate the intersection of the three sets of states and save the result to file `intersect_sets123.txt` (option `-o`):

```
$ boolSim_setutils -o intersect_sets123.txt intersection examples/Th_2006_set1.txt \
  examples/Th_2006_set2.txt  examples/Th_2006_set3.txt
```

The number of states in each file and in the resulting set of states is written to standard output:

```
file1:  9 states
file2:  1 states
file3:  2 states
```

```
result:  0 states (compressed in 0 patterns)
Empty set
```

As before, file 1 (`examples/Th_2006_set1.txt`) contains 9 states, file 2 (`examples/Th_2006_set2.txt`) contains 1 state and file 3 (`examples/Th_2006_set3.txt`) contains 2 states. None of these states is shared among all three sets, and the result of the intersection is empty. The output file `intersect_sets123.txt` is therefore empty:

```
$ cat intersect_sets123.txt
```

To better understand the overlap between the three sets (`examples/Th_2006_set*.txt`), one can partition the set of all possible states into disjoint sets formed by intersections of the input sets:

```
$ boolSim_setutils partition examples/Th_2006_set1.txt examples/Th_2006_set2.txt \
  examples/Th_2006_set3.txt
```

The number of states in each disjoint set is written to standard ouput:

```
file1:  9 states
file2:  1 states
file3:  2 states
none:  8388597 states (NOT in file1) AND (NOT in file2) AND (NOT in file3) (~100%)
1:     9       states (in file1)     AND (NOT in file2) AND (NOT in file3) (~0.000107%)
2:     0       states (NOT in file1) AND (in file2)     AND (NOT in file3) (~0%)
3:     1       states (NOT in file1) AND (NOT in file2) AND (in file3)     (~1.19e-05%)
1&2:   0       states (in file1)     AND (in file2)     AND (NOT in file3) (~0%)
1&3:   0       states (in file1)     AND (NOT in file2) AND (in file3)     (~0%)
2&3:   1       states (NOT in file1) AND (in file2)     AND (in file3)     (~1.19e-05%)
1&2&3: 0       states (in file1)     AND (in file2)     AND (in file3)     (~0%)
```

In this example, we see that among all 2^{23}=8388608 possible states (23 nodes, each with 2 states), 8388597 states are not in file 1, file 2, nor file 3. 9 states are only in file 1, one state is only in file 3, and 1 state is shared by files 2 and file 3.

6 References

1. Garg A, Di Cara A, Xenarios I, Mendoza L, De Micheli G: **Synchronous versus asynchronous modeling of gene regulatory networks.** *Bioinformatics (Oxford, England)* 2008, **24**:1917-1925.
2. Chaouiya C, Keating SM, Berenguier D, Naldi A, Thieffry D, van Iersel MP, Le Novère N, Helikar T: **The Systems Biology Markup Language (SBML) Level 3 Package: Qualitative Models, Version 1, Release 1.** *Journal of integrative bioinformatics* 2015, **12**:270.
3. Chaouiya C, Berenguier D, Keating SM, Naldi A, van Iersel MP, Rodriguez N, Dräger A, Büchel F, Cokelaer T, Kowal B *et al*: **SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools**. *BMC Systems Biology* 2013, **7**:135.
4. Mendoza L, Xenarios I: **A method for the generation of standardized qualitative dynamical systems of regulatory networks.** *Theoretical biology & medical modelling* 2006, **3**:13.