Bio-ModelChecker

Bio-ModelChecker: A Biological Network Model Checker

Hooman Sedghamiz

February 22, 2018

Preface and License

Biological Model Checker (Bio-ModelChecker) is a bounded model checker for parameterization of generalized regulatory networks (Thomas Networks) created by Hooman Sedghamiz at Center for Clinical Systems Biology (CCSB) at Rochester General Hospital.

Bio-ModelChecker treats the model parametrization problem as a multi-objective optimization inspired from *Control Theory* and *biology*. It employs the power of state of the art Constraint Programming in order to efficiently parameterize a regulatory network.

Bio-ModelChecker is available for research and evaluation purposes only. It can not be used in a commercial environment, particularly as part of a commercial product, without written permission. Bio-ModelChecker is provided as is, without any warranty. This software relies on third party packages such as **`Google OR-tools`** Perron [2011], **`MiniZinc`** Nethercote u. a. [2007], **`Chuffed`** Chu u. a. [2014] and **`OptimathSat`** Sebastiani und Trentin [2015]. Please visit their corresponding webpage for further information on licensing.

Please write to hooman.sedghamiz@rochesterregional.org for additional questions regarding licensing Bio-ModelChecker or obtaining more up-to-date versions.

Center for Clinical Systems Biology (CCSB)
Hooman Sedghamiz

Contents

Administrative Access!
Bio-ModelChecker needs to be run under administrative access.

Getting Started

Basics

Compatibility and Dependencies

Bio-ModelChecker is currently distributed as a single installation file only for Windows *64bit* operating systems. It does not have any dependencies and is shipped with **`Google OR-tools`** Perron [2011], **`Chuffed`** Chu u. a. [2014], **`OptimathSat`** Sebastiani und Trentin [2015] and corresponding **`FlatZinc`** Nethercote u. a. [2007] interpreter.

In fact, Bio-ModelChecker is independent of the solver. Any solver capable of reading **`FlatZinc`** language is compatible. In the future release of Bio-ModelChecker, we plan to add a solver importer panel in order to make it easier to use other solvers not currently offered in Bio-ModelChecker.

Installation

In order to install Bio-ModelChecker, simply run **`BioRNA-Installer.exe`**. This file installs all the dependencies for Bio-ModelChecker and prompts you with the next steps of the installation. If successful, the installer creates an executable file called **`Bio-ModelChecker.exe`** and a few more directories holding the dependencies of the software.

Note!
Please write to hooman.sedghamiz@rochesterregional.org if you might encounter any error during installation or might need additional information regarding the updated versions of Bio-ModelChecker.

Quick Start

Double click **`Bio-ModelChecker.exe`** and click on *import* to import a model from **`Benchmark`** directory. Select a time-update from the upper left corner of Bio-ModelChecker (see Figure 1) and select a solver (e.g. **`Chuffed`**). Hit **`Run`**. If successful a cmd window will be opened and a solving instance of the model would run on it. Note that if you close the cmd window the solving will be killed. The parametrization results are saved in the **`results`** folder with the same name as the model.

Bio-ModelChecker Overview

Main Panel

Bio-ModelChecker model checker module panel is illustrated in Figure 1. Right panel provides an *excel* type view of the adjacency network under analysis. Left panel provides settings for tuning the time update, model checking bound, type of the solver and maximum solving time out in seconds. In order to get started, one first needs to import a model and get familiar with the input format of Bio-ModelChecker. See the next section for more details.

Bio-ModelChecker Main Panel

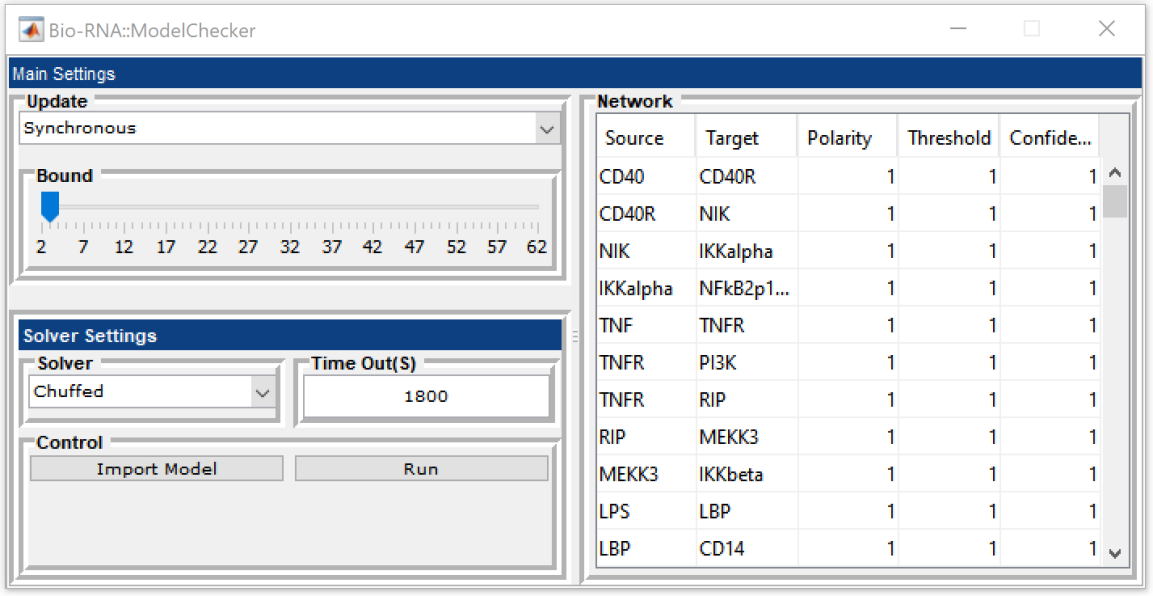

Figure 1: Right panel provides an *excel* type view of the adjacency network under analysis. Left panel provides settings for tuning the time update, model checking bound, type of the solver and maximum solving time out in seconds.

Importing a model

JSON Format

Bio-ModelChecker employs *Java Script Object Notation* (**JSON**) as a standard input format. This is due to the fact that `JSON` follows a standard nesting of objects and there already exists various tools in different languages in order to conveniently convert a matrix or complex structure to a `JSON` file. Figure 2 is an example of Hypothalamic Pituitary Axis (HPA) model in `JSON` format. The main object is the model name (e.g. HPA) and the rest of its attributes are nested under it with `[]`. Note the naming of each attribute. Bio-ModelChecker uses the name of attributes as keys in order to distinguish the name of entities (e.g. `titles`) or the adjacency list of the network (e.g. `interaction`). For a detailed list of attributes and their naming convention see Table 1.

Remarks!

- A model always is represented as an object and has a name (e.g. HPA)
- Interaction has 5 rows, all the unknowns are denoted by -1 (See next page for spec).
- Tr holds the measurements. In this case there are two measurement instances, the first has 2 time samples and the second has 3 samples.
- L stands for maximum levels of each entity. In this case the first two entities assume a binary value while the last two are ternary.

HPA.json

```
{
"HPA": {
"interaction": [
[1,2,2,3,4,4,1,3],
[3,1,4,2,3,4,4,1],
[-1,-1,-1,-1,-1,-1,-1,-1],
[0,1,0,1,1,1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1]
],
"L": [1,1,2,2],
"Tr": [
[
[0,0,0,0],
[1,1,0,0]
],
[
[0,0,2,2],
[0,0,1,2],
[0,0,0,2]
]],
"titles": ["CRH","ACTH","Cort","R"]}
}
```

Figure 2: JSON example.

JSON Spec

A detailed description of attributes can be found in Table 1.

Bio-ModelChecker JSON attribute Spec

Attributes and their interpretation by Bio-ModelChecker

Name of entities in the model.

Attribute key in JSON:
'title','names','entities','TITLES'.

Adjacency list of the model.

Attribute key in JSON:
'connection','interaction','edgelist','Edge','Adjacency','ADJ'.
Attribute in JSON:

- 1st Row : Target node
- 2nd Row : Source node
- 3rd Row : Threshold of action (min:1,max:inf,unknown:-1)
- 4th Row : Polarity (negative:0,positive:1,unknown:-1)
- 5th Row : Confidence (Certain:1,unknown:-1)

Max Expression Level.

Attribute key in JSON:
'Levels','L','levels'.
Note: It should be a vector the same length as the number of entities in the model.

Measurement Trajectory Matrices.

Attribute key in JSON:
'Tr','Trajectory','measurement'.
Note: It should be matrix with minimum 2 rows and columns the same size as the number of entities in the model. Unknown measurements are denoted by -1.

Node Steady States.

Attribute key in JSON:
'attractor','atr','ATR','SS','SteadyState'.
Note: Each attractor is a vector with the same length as the number of entities. For the unknown entities one can pass in -1.

Table 1: Bio-ModelChecker Spec.

Preparing your model as JSON in your favorite Language

Table 2 details a list of import&export packages available under different programing languages in order to convert a data-type into `JSON` format.

Language	Toolbox&Link
R	`jsonlite`
MatLab	`JSONlab`
Python	`JSON`
C++	`JSON++`
Java	`JSON Oracle`
Scala	`CIRCE`

Table 2: Packages for working with JSON format.

Benchmarks

Bio-ModelChecker currently is shipped with 5 benchmarks that are detailed in Table 3. Each example provides a different type of model in the sense that some have missing interactions or require to reproduce a certain node steady state or measurement. The models have chosen specifically to show different functionalities of Bio-ModelChecker. Number of measurements here stands for a measurement under a certain condition not number of samples in each measurement. For instance, a measurement might have been done under knockout of a certain input and measured for 10 time samples. The bench marks can be found in the **`Benchmark`** directory.

Model	Author	Nr. Entities	Nr. Measurements	Nr. Attractors
`HPA Axis`	Sedghamiz u. a. [2018]	4	2	0
`HPG Axis`	Sedghamiz u. a. [2017]	5	1	0
`Dendritic Cell`	Garg u. a. [2008]	111	1	0
`Irma`	Cantone u. a. [2009]	6	1	0
`Th-helper`	Garg u. a. [2008]	23	0	3

Table 3: Benchmarks provided with Bio-ModelChecker.

Parameterizing a model

In order to parameterize a model, first *import* the model (e.g. from **Benchmark** directory). Then, choose a bound for model checking. Select an updating scheme (Synchronous or Asynchronous Chaouiya u. a. [2003]) and a particular solver. After hitting the *Run*, Bio-ModelChecker translates all the constraints along with measurement trajectories into a **FlatZinc** that is solved with a SAT, CP, SMT or LCG solver.

> Note that increasing the bound dramatically increases the solving time. Specially for larger models, one might start with a lower bound, since Bio-ModelChecker relies on CP technique which is usually really fast to prove *Unsatisfiability*.

Interpreting Results

The output of the parameterization is saved in **results** folder as a `.txt` file. The output if the parametrization was satisfiable looks like Figure 3;

HPA.TXT

```
A = array1d(1..3, [8, 0, 0]);
EM = array1d(1..8, [true, true, true, true, true, true, false, false]);
K = array2d(1..4, 1..4, [0, 1, 0, 0, 0, 1, 1, 1, 0, 2, 0, 0, 0, 1, 2, 2]);
U = array1d(1..8, [false, true, false, true, true, true, false, false]);
W = array1d(1..8, [1, 1, 1, 1, 2, 2, 1, 1]);
```

Figure 3: Sample of parametrization results.

Table 4 details each vector in the results.

Name	Interpretation
A	Objective vector: Number Edges × Threshold of action , Min path length, Max Robustness respectively.
EM	Confidence of interactions, 1 implies that the edge is necessary, 0 edge is redundant
K	Logical parameters
U	Polarity of each edge
W	Threshold of action for each interaction.

Table 4: Result output Spec.

Bibliography

[Cantone u. a. 2009] Cantone, Irene ; Marucci, Lucia ; Iorio, Francesco ; Ricci, Maria A. ; Belcastro, Vincenzo ; Bansal, Mukesh ; Santini, Stefania ; Bernardo, Mario di ; Bernardo, Diego di ; Cosma, Maria P.: A Yeast Synthetic Network for In Vivo Assessment of Reverse-Engineering and Modeling Approaches. In: *Cell* 137 (2009), Nr. 1, S. 172–181. – URL http://dx.doi.org/10.1016/j.cell.2009.01.055. – ISBN 0092-8674

[Chaouiya u. a. 2003] Chaouiya, Claudine ; Remy, Elisabeth ; Moss, Brigitte ; Thieffry, Denis: Qualitative Analysis of Regulatory Graphs : A Computational Tool Based on a Discrete Formal Framework. In: *Positive Systems* (2003), S. 119–126. ISBN 978-3-540-40342-5

[Chu u. a. 2014] Chu, Geoffrey ; Garcia De La Banda, Maria ; Mears, Christopher ; Stuckey, Peter J.: Symmetries, almost symmetries, and lazy clause generation. In: *Constraints* 19 (2014), Nr. 4, S. 434–462. – ISBN 9781577355120

[Garg u. a. 2008] Garg, Abhishek ; Di Cara, Alessandro ; Xenarios, Ioannis ; Mendoza, Luis ; De Micheli, Giovanni: Synchronous versus asynchronous modeling of gene regulatory networks. In: *Bioinformatics* 24 (2008), Nr. 17, S. 1917–1925. – ISBN 1367-4811 (Linking)

[Nethercote u. a. 2007] Nethercote, Nicholas ; Stuckey, Peter J. ; Becket, Ralph ; Brand, Sebastian ; Duck, Gregory J. ; Tack, Guido: MiniZinc: Towards a Standard CP Modelling Language. In: Bessière, Christian (Hrsg.): *Thirteenth International Conference on Principles and Practice of Constraint Programming* Bd. 4741. Providence, RI, USA : Springer-Verlag, sep 2007, S. 529–543. – URL http://www.gecode.org/paper.html?id=NethercoteStuckeyEa:CP:2007

[Perron 2011] Perron, Laurent: Operations Research and Constraint Programming at Google. In: Lee, Jimmy (Hrsg.): *Principles and Practice of Constraint Programming – CP 2011*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, S. 2. – ISBN 978-3-642-23786-7

[Sebastiani und Trentin 2015] Sebastiani, Roberto ; Trentin, Patrick: OptiMathSAT: A Tool for Optimization Modulo Theories. In: *Proc. International Conference on Computer-Aided Verification, CAV 2015* Bd. 9206, Springer, 2015

[Sedghamiz u. a. 2017] Sedghamiz, H ; Chen, W ; Rice, M ; Whitley, D ; Broderick, G: Selecting Optimal Models Based on Efficiency and Robustness in Multi-valued Biological Networks. In: *2017 IEEE 17th International Conference on Bioinformatics and Bioengineering (BIBE)*, oct 2017, S. 200–205

[Sedghamiz u. a. 2018] Sedghamiz, Hooman ; Morris, Matthew ; Craddock, Travis J A. ; Whitley, Darrell ; Broderick, Gordon: High-fidelity discrete modeling of the HPA axis : A study of regulatory plasticity in biology. In: *In Submission BMC Systems Biology* (2018), S. 1–21