Parsybone manual
Version 1.2

Adam Streck
Discrete Biomathematics, FU Berlin

May 17, 2013

1 Introduction

1.1 Preface

Parsybone is a CLI based tool for synthesis of discrete kinetic parameters in gene regulatory networks [2] and their further analysis. Regulatory networks are provided using a so-called *Thomas network* formalism [8] and analyzed using a method of *colored* LTL model checking [7]. The model checking procedure is based on the behavioral constrains provided in the form of Büchi automata [1]. Features of this tool include:

- Reduction of the parameter space by application of edge constrains [6].
- Enumeration of the shortest witnesses [7] for synthesized parameters.
- Enumeration of the Robustness [7] values for synthesized parameters.
- Computation in a distributed environment.

This manual describes the usage of the version 1.2 of the tool and the version 1.2 of the associate modeling language.

1.2 Availability

The tool is freely available under the GNU GPLv3 license. The latest version of the tool can be obtained from the GitHub repository at https://github.com/sybila/Parsybone.

1.3 Compiling Parsybone

The only currently available version of the program is a source code archive, which has to be compiled before the tool can be run. For this a GNU C++ compiler of version at least 4.6.0 must be used (MinGW can serve as a replacement on the Windows platform).

To successfully compile the code you also need the Boost libraries [4] which can be obtained at http://www.boost.org/. The code has been tested using the 1.50.0 version.

The tool can be then compiled by invoking the GNU make utility in the main directory (the one containing the file *Makefile*).

2 Basics

In this section we provide a concise introduction to the topic of the Thomas formalism, sufficient for understanding our modeling method and language. For more complex description of the Thomas formalism we would recommend the reader to refer to more elaborate articles, such as [8]. Readers well-established in the topic can skip this section alltogether.

To help with understanding of concepts of the Thomas formal method we demonstrate basic notions on a very simple example of a regulatory network, depicted in Figure 1. This network has two boolean components named cA and B, each regulated by both itself and the other component. This a bit unconventional naming has been choosen to demonstrate variability of modeling language later in the Section 3.

The model is a boolean one, which means that there are only two activity levels for each component, roughly corresponding to the situations where its concentration is below threshold or above it. The threshold marks a concentration boundary whose crossing usually causes the component to change its regulatory effect. Boolean component therefore usually works as a switch.

Components of such a network are usually well-know, conversely to their regulatory effects that are hard to obtain from biological measurements [5]. The Parsybone is designed to solve the task of determining those effects or at least to narrow the set of posibilities thus helping with further analysis. Possible regulatory effects of a component are given by the so-called *logical parameters*. Each component has usually several logical parameters that

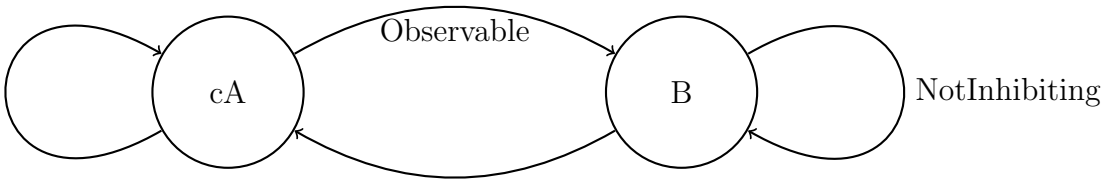

Figure 1: Simple example network with two components.

specify towards which activity level the component inclines based on the current activity levels of its regulators e.g. a self-regulating component can stop its production when a certain concentration level is reached.

The user is required to specify the model as a set of components and their interactions. Having such a model, the Parsybone generates all possibile combinations of logical parameters for the model, creating a set of *parametrizations*. This set is usually called *parametrization space* and it can be viewed as a set of all behavioural possibilities for the model. As a second input, the Parsybone takes a specification of some behaviour the model must be able to reproduce. Parametrizations that do not allow such a behaviour are then removed from the set of possibilities.

Apart from basic boolean properties we also allow the model to be created using multiple extensions:

- Multi-valued components, allowing to change the effect of only a subset of its regulatory effects at a time.
- Edge labels, bounding possible effects of the regulations.
- Partial parametrizations, allowing to reduce the parametrization space before the analysis itself.
- Basal values, specifying general case behaviour of components.

These extension allow for more precise model description than the basic boolean network formalism as presented above, but their explanation is beyond scope of this section. In case of furter interest please refer to [7].

3 Modeling

Models are described using an internal modeling language, based on the XML syntax [3], called *DBM* (Discrete biological model). The model is provided within a single DBM file that holds specification for the model together with static and dynamic constrains of the parametrization space. Such a file has to have a `.dbm` suffix, and to abide by the overall rules for XML files.

3.1 Model example

Every model must be enclosed within a pair `MODEL` tag. This tag has a single mandatory attribute `ver`, which must contain a floating point number specifying the employed version of the DBM language. A detailed description of the whole language is provided later in this section, here we present, as an example, model file for the network depicted in Figure 1. This example model has a quite non-uniform syntax, which has been chosen on purpose to present different possibilities of model description.

```
<MODEL ver="1.0">
    <STRUCTURE>
        <SPECIE undef="basal" name="cA">
            <REGUL source="B" threshold ="1" label="Observable"/>
            <REGUL source="cA" />
        </SPECIE>
        <SPECIE>
            <REGUL source="cA" />
            <REGUL source="B" label="+" />
            <PARAM context="" value="?" />
            <PARAM context="B" value="1" />
            <PARAM context="cA,B" value="0,1" />
        </SPECIE>
    </STRUCTURE>
    ... specification of a property ...
</MODEL>
```

As can be seen, the model is a structure with two species, both being affected by two regulations. For the component B, the possible parametrizations space is reduced by requirements that the regulation from cA must be observable and that the effect of its self-regulation must be positive, if any. Also, the logical parameter of self-regulation of the component B must always be 1. As a result, the parametrization space is reduced to four possibilities.

3.2 Model property

The main purpose of the tool is picking parametrizations that satisfy some property. The description of this property can be given in one of two possible ways - either as Büchi automaton or as a time series, whose specification is inserted between `</STRUCTURE>` and `</MODEL>` tags. A time series is merely a Büchi automaton specialization, but as will be explained later the Parsybone is optimized for its usage and provides additional features if the time series is employed. To demonstrate the difference between the two, we present a single property described using each formalism. This property assures that the model in Figure 1 is able to reproduce a time series composed of the following three measurements:

1. $cA = 0 \wedge (B = 0 \vee B = 1)$
2. $cA \Leftrightarrow B$
3. $cA = 1 \wedge B = 0$

Only two out of four parametrizations allow for reproduction of this time series. To obtain them, we can describe the time series either using the Büchi automaton:

```
<AUTOMATON>
    <STATE final="0">
        <EDGE target="0" label="tt" />
        <EDGE target="1" label="cA=0" />
    </STATE>
    <STATE>
        <EDGE target="1" label="tt" />
        <EDGE target="last" label="((cA=0 &amp; B=0) | (cA=1 &amp; B=1))" />
    </STATE>
    <STATE name="last">
        <EDGE target="last" label="tt" />
        <EDGE target="3" label="(cA=1 &amp; B=0)" />
    </STATE>
    <STATE final="1">
        <EDGE target="3" label="tt" />
    </STATE>
</AUTOMATON>
```

Or using the time series directly:

```
<SERIES>
    <EXPR values="cA=0" />
    <EXPR values="((cA=0 &amp; B=0) | (cA=1 &amp; B=1))" />
    <EXPR values="(cA=1 &amp; B=0)" />
</SERIES>
```

As can be seen, the second method makes the model quite shorter and shuld be used for description of time series.

3.3 Model holder description

- `MODEL`
  - Occurrence: single, mandatory.
  - Type: pair.
  - Parent: none.
  - Description: encloses the whole model.
  - Attributes:
    1. *ver*
       - Occurrence: mandatory.
       - Value: must be 1.0 in the current version of the tool.

3.4 Regulatory network description

- `STRUCTURE`

- Occurrence: single, mandatory.
- Type: pair.
- Parent: MODEL.
- Description: encloses the definition of a regulatory network.
- Attributes: none.

- `SPECIE`
  - Occurrence: multiple, mandatory.
  - Type: pair.
  - Parent: STRUCTURE.
  - Description: defines a single specie.
  - Attributes:
    1. *name*
       * Occurrence: optional.
       * Value: string containing letters and numbers.
       * Default: Capital letter, starting from `A`.
       * Description: name of the specie under which it will be further addressed.
    2. *undef*
       * Occurrence: optional.
       * Value: basal/param/error.
       * Default: param.
       * Description: tells the system how it should handle values of regulatory contexts that are not specified. Basal means using a basal value, param means using all possible values and error causes error in case there are unspecified parameters.
    3. *max*
       * Occurrence: optional.
       * Value: natural number.
       * Default: 1.
       * Description: maximal activation level this specie can occur in.
    4. *basal*
       * Occurrence: optional.
       * Value: positive integer.
       * Default: 0.
       * Description: basal activation level of this specie - the value towards which the specie tends if not specified otherwise.

- `REGUL`
  - Occurrence: multiple, mandatory.
  - Type: solo.
  - Parent: SPECIE.

- Description: defines a single incoming regulation of the parent specie.
- Attributes:
  1. *source*
     * Occurrence: mandatory.
     * Value: name or the ordinal number of a specie.
     * Description: name of the specie that regulates this one.
  2. *threshold*
     * Occurrence: optional.
     * Value: natural number.
     * Default: 1.
     * Description: lowest activation level of the source specie that activates the regulation.
  3. *label*
     * Occurrence: optional.
     * Value: a string, see Sec. 3.7.
     * Default: Free.
     * Description: describes nature of the regulation.

- `PARAM`
  - Occurrence: multiple.
  - Type: solo.
  - Parent: SPECIE.
  - Description: defines a single kinetic parameter.
  - Attributes:
    1. *context*
       * Occurrence: mandatory.
       * Value: comma separated list of active regulations, given by a name or an ordinal number of a regulator.
       * Description: defines the exact regulatory context in which this kinetic parameter is applied.
    2. *value*
       * Occurrence: optional.
       * Value: positive integer or ?.
       * Default: ?.
       * Description: specifies target value of the specie in this regulatory context, which must be one of possible activation levels of the specie. Character ? means that an exact number is unknown and a parametrization for each value between 0 and the maximal activation level of the specie is created.

3.5 Büchi automaton description

- `AUTOMATON`
  - Occurrence: single, present if and only if there is no sibling SERIES tag.
  - Type: pair.
  - Parent: MODEL.
  - Description: encloses decription of a Büchi automaton.
  - Attributes: none.

- `STATE`
  - Occurrence: multiple, mandatory.
  - Type: solo.
  - Parent: AUTOMATON.
  - Description: defines a single state of the automaton. The first state in the description is also considered to be the initial state of the automaton.
  - Attributes: none.

  1. *name*
     - Occurrence: optional.
     - Value: string containing letters and numbers.
     - Default: ordinal number of the tag, counting from zero.
     - Description: name of the specie under which it will be further addressed. System also uses its ordinal number (so the first state can be addressed using the name *0*).
  2. *final*
     - Occurrence: optional.
     - Value: Boolean.
     - Default: 0.
     - Description: specifies if the state is final (1) or not (0).

- `EDGE`
  - Occurrence: multiple, mandatory.
  - Type: solo.
  - Parent: STATE.
  - Description: defines an edge leading from the parent state.
  - Attributes:
    1. *target*
       - Occurrence: mandatory.
       - Value: name or the ordinal number of the target state.
    2. *label*
       - Occurrence: mandatory.

        * Value: logical formula (see Section 3.8), variables are atomic propositions (see Section 3.9).
        * Description: conditions that must be met for the edge to be transitive.

3.6 Time series description

- `SERIES`
  - Occurrence: single, present if and only if there is no sibling AUTOMATON tag.
  - Type: pair.
  - Parent: MODEL.
  - Description: encloses definition of a time series.
  - Attributes: none.

- `EXPR`
  - Occurrence: multiple, mandatory.
  - Type: solo.
  - Parent: SERIES.
  - Description: a single measurement in the time series.
  - Attributes:
    1. *values*
       * Occurrence: mandatory.
       * Value: logical formula (see Section 3.8), variables of the formula must be atomic propositions (see Section 3.9).
       * Description: conditions that must be met for the measurement to be reproduced.

3.7 Edge label

There are two basic labels:

+ Meaning that the there must be a regulation whose parameter value increases if we add this regulator.

- Has an opposite meaning.

One can compose these labels using a logical formula over + and − or use one of the following predefined descriptions:

- Activating: $+$
- ActivatingOnly: $(+ \wedge \neg -)$
- Inhibiting: $-$
- InhibitingOnly $(- \wedge \neg +)$

- NotActivating $\neg+$;
- NotInhibiting $\neg-$
- Observable: $(+ \vee -)$
- NotObservable: $(\neg + \wedge \neg -)$
- Free: $true$

3.8 Formula construction

A formula is constructed using the following set of recursive rules:

1. tt and ff are formulas representing true or false respectively,
2. any variable is a formula,
3. for every formula A is $\neg A$ a formula,
4. for formulas A, B are $(A|B)$ and $(A\&B)$ formulas representing logical disjunction or conjunction respectively,
5. nothing else is a formula.

Note that in the model $\neg$ is denoted using `!` and & is denoted using `&`.

3.9 Atomic propositions

An atomic proposition is a string of the form: $specie * value$, where:

- $specie$ denotes the name or ordinal number of a specie,
- $*$ denotes comparison operator from the set $\{<, >, =\}$,
- $value$ denotes a positive integer with which the value of the specie is compared.

Note that in the model $<$ is denoted using `<` and $>$ is denoted using `>`.

4 Usage

Parsybone is a CLI-based tool and therefore all its settings are provided as text arguments when starting the tool - from that point on, there is no way to influence the process or its results. In this section we describe how and which arguments can be used and what results the computation process gives. It is also important to have in mind that the Parsybone is primarily designed for the time series analysis and some of its capabilities rely on its usage.

4.1 Execution

The syntax of the execution is:

```
parsybone
model.dbm
[-cdfmMrvwW]
[--data database_file]
[--file text_file]
[--min bitmask_file]
[--mout bitmask_file]
[--dist I N]
```

The first argument, *model.dbm*, is the only one that must be present and it gives the name of a file containing the specification of a model and a property to be checked on that model. The file also has to have a *.dbm* suffix, otherwise it will be rejected.

Other arguments tell the tool to alter the procedure somehow and their meaning is described in the following list. Techniques of their usage are explained in more detail later in this section.

-c display computation results on console

-d output computation results to an sqlite database by default the file is model.sqlite, can be changed using the `--data` switch

-f output computation results to a text file by default the file is model.out, can be changed using the `--file` switch

-M use an input mask for parametrization filtering by default the file is model.pbm, can be changed using the `--min` switch

-m output parametrizations as a mask by default the file is model.pbm, can be changed using the `--mout` switch

-r compute robustness of a time series

-v verbose (output progress to console)

-w compute witnesses and store them in encoded form

-W compute witnesses and store them in explicit form

- `--dist` used for distributed computation with two integers, denoting the I-th process out of N. Total - each of those tests only 1/N of the parametrization space.

4.2 Output format

There are four types of internal streams that are used within the program:

- statistics: This stream is visible only when the $-s$ switch is used, it prints on a standard output stream and each of its lines starts with a `#` symbol. This output contains mainly sizes of the structures and parametrizations sets used.
- verbose: This stream is visible only when the $-v$ switch is used, it prints on a standard output stream and each of its lines starts with a `*` symbol. Independent parts of the process announce their start through this stream. It is mainly important because it displays progress of a coloring procedure.
- errors: This stream prints on a standard error stream and each of its lines starts with a `!` symbol. Most error messages announce failure of the program.
- results: This stream prints on a standard output stream or to a file provided using the $-f$ switch and it is the only stream whose lines have no prefix. The syntax of the content of this stream is described in the following section.

4.3 Results syntax

The result of the computation is a list of feasible parametrizations, one per line, possibly extended with an outcome of additional analysis. Each of the lines is in the following form: `acceptable parametrization number : parametrization values : cost : robustness : {time series walks transitions}`

Running an analysis of our example model using $-rW$ as an argument results in the following output:

```
0:(0,1,0,0,0,0,1,0):3:0.5:{(0,1;0)>(1,1;1),(1,1;1)>(1,0;2)}
1:(0,1,0,0,1,0,1,0):3:0.5:{(0,1;0)>(1,1;1),(1,1;1)>(1,0;2)}
```

4.4 Results semantics

Each comma separated value represents some sort of information about the parametrization. The semantics of the independent values are defined as follows:

1. `acceptable parametrization number`: this number uniquely identifies the parametrization within a set of all possible parametrizations of the model.
2. `parametrization values`: a vector of target values for each component. The values are ordered so that a parameter of the component that is closer to the beginning of the model file is before a parameter of the component that is closer to the end. The parameters of a single component are then ordered ascendingly by the number of present regulators. If the number is the same, contexts are again ordered by their appearance in the file. For our example model we first have the target value of the

component 0 with no present regulators, then with the self-regulation, then only with the regulation from the second component etc.

3. `cost`: The Cost value of this parametrizations given by a natural number. This value is only available for a time series property.

4. `robustness`: The Robustness value of this parametrization given by a floating point number. This value is present only when the $-r$ switch has been used while checking a time series property.

5. {`time series walks transitions`}: A comma separated list of transitions corresponding to the shortest time series walks in the form `source>target`.

   If the $-W$ switch was used, this list contains a vector of current activation levels of each component, again ordered in the same manner as in which they are present in the model file, and the current state of the Büchi automaton separated by a semicolon.

   If the $-w$ switch was used, only the unique ID number of a state is used.

   If neither $-w$ nor $-W$ were used or the property checked is not a time series, this part remains empty.

4.5 Using multiple properties

It is possible to keep only parametrizations that satisfy multiple properties (this way for example bistability can be enforced). For this purpose, the tool must be run independently for each property that is to be tested and the underlying regulatory network must remain intact between computations so the parametrization spaces of the two match. To save the information which parametrizations satisfy the property in a file, use the $-M$ switch followed by the name of the file to create. This file is then filled with a stream of bits where the n-th bit is set to 1 if the n-th parametrization is feasible, otherwise it is set to 0. This file can be read in a subsequent computation using the $-m$ switch - this ensures that only parametrizations that were marked as feasible are used. It is also possible to create a chain of computations using both switches at once.

4.6 Distribution

Computation can be easily executed in a distributed environment - in such a case, the process does not take the whole parametrization space but only a part of it. This is done using the $-d$ switch with two numbers. The first number denotes the ID of this process and the second one the total number of processes employed - this tells the current process which part of the parametrization space it should use.

4.7 Performance considerations

There are some thoughts a user should keep in mind when creating a model to ensure its analysis is computable in reasonable time. The main performance issue is the size of parametrization space - adding new components is usually not a problem, but each new regulation increases the number of parametrizations of a single component greatly - a model that has a component with more than 5 regulators is not computable without any further reduction of the parametrization space (a boolean component with six regulators has roughly 2^{20} possible parametrizations).

When checking a time series, the computation is much faster in comparison with standard ω-regular property. This is due to the fact that this property belongs to the class of reachability properties that are much easier to check. On the other hand, computation of Robustness or shortest time series walks adds some overhead (if employed). These procedures rise in complexity with the length of the time series walk - the longer the distance between two successive measurements, the more complex this analysis gets - smaller differences between measurements or more precise definitions of measurements can help lower the complexity of these procedures.

References

[1] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[2] J. Barnat, L. Brim, A. Krejci, A. Streck, D. Safranek, M. Vejnar, and T. Vejpustek. On Parameter Synthesis by Parallel Model Checking. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(3):693–705, 2012.

[3] T. Bray, J. Paoli, C. M. Sperberg-Mcqueen, Eve, and F. Yergeau, editors. *Extensible Markup Language (XML) 1.0*. W3C Recommendation. W3C, fourth edition, Aug. 2003.

[4] R. Demming and D. Duffy. *Introduction to the Boost C++ Libraries; Volume I - Foundations*. Datasim Education Bv, 2010.

[5] M. Hecker, S. Lambeck, S. Toepfer, E. van Someren, and R. Guthke. Gene regulatory network inference: Data integration in dynamic modelsa review. *Biosystems*, 96(1):86 – 103, 2009.

[6] H. Klarner, H. Siebert, and A. Bockmayr. Time series dependent analysis of unparametrized thomas networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9:1338–1351, 2012.

[7] H. Klarner, A. Streck, D. Safranek, J. Kolcak, and H. Siebert. Parameter identification and model ranking of thomas networks. Technical Report FIMU-RS-2012-03, Masaryk University, 2012.

[8] R. Thomas. Regulatory networks seen as asynchronous automata: A logical description. *Journal of Theoretical Biology*, 153(1):1–23, Nov. 1991.