Parsybone manual

Adam Streck
Systems Biology laboratory

September 23, 2012

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 [7] and analyzed using a method of *colored* LTL model checking [6]. The model checking procedure is based on the behavior description provided in the form of Büchi automaton [1]. Features of this tool include:

- Reduction of parameter space by application of edge constrains [5].
- Enumeration of shortest witnesses [6] for synthesized parameters.
- Enumeration of Robustness [6] values for synthesized parameters.
- Computation in distributed environment.

This manual describes usage of version 1.0 of the tool and version 1.0 of the associate modeling language.

1.2 Availability

The tool is freely available under the GNU GPLv3 license. Latest version of the tool can be obtained from the GitHub repository at https://github.com/sybila/Parsybone, version described in this manual is available at https://github.com/sybila/Parsybone/tree/1.0.

1.3 Compiling Parsybone

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 you need a compiler at least partially aware of the *C++11* standard, we recommended one of the following:

- Microsoft Visual Studio 10.0
- MinGW 4.5.1
- GCC 4.5.1

Specified versions have been tested, but newer should work as well. Also note that in this version of the GNU compilers it is necessary to use argument `-std=gnu++0x` when invoking the compilation.

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

2 Modeling

Models are described using an internal modeling language, based on XML syntax [3], called *DBM* (Discrete biological model). 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 file has to have a `.dbm` suffix, and to abide by the rules of XML files.

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 employed version of the DBM language.

Independent parts of the language are described in detail within following sections.

2.1 Example

We provide an example of a model of a small network. The syntax used in this example is quite nonuniform so it can present many possible options of writing a file. Because the model has to abide by the XML rules, character `&` must be described using an XML entity [3] `&`.

```
<MODEL ver="1.0">
    <STRUCTURE>
        <SPECIE undef="basal">
            <REGUL source="1" threshold ="1" />
            <REGUL source="0" />
            <LOGIC formula="(second&amp;!0)" />
        </SPECIE>
        <SPECIE name="second">
            <REGUL source="0" observ="1"/>
            <REGUL source="second" label="+" />
            <PARAM context="" value="?" />
            <PARAM context="second" value="1" />
```

```
            <PARAM context="0,1" />
        </SPECIE>
    </STRUCTURE>
    ... specification of a property ...
</MODEL>
```

The model specifies a simple network with two components, named 0 and *second*. Each of the components have two regulations - one from itself and one from the other component, all of these having threshold equal to 1. Formula $second \wedge \neg 0$ specifies all kinetic parameters of the component 0. For the *second* component the possible parametrizations space is reduced by requirements that regulation from 0 must be observable and that effect of its self-regulation must be positive, if any. Also, target value of its self-regulation must always be 1.

Main purpose of the tool is picking parametrizations that satisfy some property. Description of this property can be given in one of two possible ways - either as an Büchi automaton or as a time series, whose specification is inserted between `</STRUCTURE>` and `</MODEL>` tags. Time series is only a specialized version of Büchi automaton, but as will be explained later Parsybone is optimized for its usage and it is therefore easier to compute and also to describe. To demonstrate the difference between the two, we present a single property described using each formalism. This property assures that the model is able to reproduce a time series in whose first measurement the component 0 has value 0, in the second measurement are values of both components equal and in the third measurement the 0 component has value 1 and the *second* component has value 0. First we describe this property as a Büchi automaton:

```
    <AUTOMATON>
        <STATE final="0">
            <EDGE target="0" label="tt" />
            <EDGE target="1" label="0=0" />
        </STATE>
        <STATE>
            <EDGE target="1" label="tt" />
            <EDGE target="last" label="((0=0&amp;second=0)|(0=1&amp;1=1))" />
        </STATE>
        <STATE name="last">
            <EDGE target="2" label="tt" />
            <EDGE target="3" label="(0=1&amp;1=0)" />
        </STATE>
        <STATE final="1">
            <EDGE target="4" label="tt" />
        </STATE>
    </AUTOMATON>
```

And now the same property described as a time series:

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

For this example model, there are four possible parametrizations that satisfy the constrains given by edge labels. Out of these, only two are feasible - meaning that they allow to reproduce the specified behavior.

2.2 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 current version of the tool.

2.3 Regulatory network description

- `STRUCTURE`
  - Occurrence: single, mandatory.
  - Type: pair.
  - Parent: MODEL.
  - Description: encloses 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: 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 specie can be addressed using the name *0*).
  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. *observ*
      * Occurrence: optional.

        * Value: + or - or ?.
        * Default: ?.
        * Description: describes nature of the regulation as activation (+) or inhibition (-) or any (?).
    4. *label*
        * Occurrence: optional.
        * Value: Boolean.
        * Default: 0.
        * Description: specifies if the regulation must be observable (1) or can be unobserved (0).

- `PARAM`
  - Occurrence: multiple, present if and only if there is no sibling LOGIC tag.
  - 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 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.

- `LOGIC`
  - Occurrence: single, present if and only if there is no sibling PARAM tag.
  - Type: solo.
  - Parent: SPECIE.
  - Description: defines parametrization of the specie using logical formula over the regulators.
  - Attributes:
    1. *formula*

    - Occurrence: mandatory.
    - Value: logical formula (see Section 2.6), variables must be names or ordinal numbers of regulators.
    - Description: result target value is given as a valuation of the formula, where a variable denoting a regulator is valuated to 1 if the regulation is active and to 0 otherwise.

2.4 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 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 2.6), variables are atomic propositions (see Section 2.7).
     * Description: conditions that must be met for the edge to be transitive.

2.5 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 2.6), variables of the formula must be atomic propositions (see Section 2.7).
       * Description: conditions that must be met for the measurement to be reproduced.

2.6 Formula construction

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 A formula is $\neg A$ formula,

4. for A, B formulas 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 `&`.

2.7 Atomic propositions

Atomic proposition is a space-less string in of the form: $specie * value$, where:

- *specie* denotes 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 `>`.

3 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 Parsybone is primarily designed for time series analysis and some of its capabilities rely on its usage.

3.1 Execution

Syntax of execution is:

```
Parsybone
    filename
    [-rsvwW]
    [-d num1 num2]
    [-f filename]
    [-m filename]
    [-M filename]
```

The first argument, $filename$, is the only one that must be present and it gives name of a file containing 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 further in this section.

- $-r$: for each feasible parametrization compute and display its Robustness.
- $-s$: display statistics during the computation (mainly sizes of graphs and parametrization sets used).
- $-v$: display progress of individual parts of the computation.
- $-w$: for each feasible parametrization compute and display transitions belonging to the shortest time series walks. Use numerical ID of states when describing transitions.
- $-W$: similar to $-w$, but use full values of states instead. This switch has a higher priority than $-w$.
- $-d\ num1\ num2$: test only subspace of the parametrization space. This subspace is given as a $num1$-th part from $num2$ total parts.
- $-f\ filename$: store feasible parametrizations and their properties in the given file.
- $-m\ filename$: test only parametrizations marked in the provided file.
- $-M\ filename$: store mask of feasible parametrizations in the given file.

3.2 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 its line 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 its line 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 its line 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. Syntax of content of this stream is described in the following section.

3.3 Results syntax

Result of the computation is a list of feasible parametrizations, one per line, possibly supplied 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 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)}
```

3.4 Results semantics

Each comma separated value represents some sort of information about the parametrization. Semantics of independent values are following:

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. Values are ordered so that parameter of a component that is closer to beginning of the model file is before parameter of a component that is closer to the end, parameters of a single component are then ordered in such manner that parameter whose context has less present regulators goes first and 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`: Cost value of this parametrizations given by a natural number. This value is only available for a time series property.

4. `robustness`: 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 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 current state of the Büchi automaton, separated by a semicolon.

   If the $-w$ switch was used, only a unique 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.

3.5 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. To save an information which parametrizations satisfy the property in a file, use the $-M$ switch followed by a name of the file. This file is then filled with a stream of bits where n-th bit is set to 1 if an 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.

3.6 Distribution

Computation can be easily executed in a distributed environment - in such 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. First number denotes ID of this process and the second one the total number of process employed - this tells current process which part of the parametrization space it should use.

3.7 Performance considerations

There are some thoughts user should keep in mind when creating a model to ensure its analysis is computable in a 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 number of parametrizations of a single component greatly - model that has a component with more than 5 regulators is not computable without any further reduction of parametrization space (a boolean component with six regulations has roughly 2^{20} kinetic parameters).

When checking a time series, the computation is much faster. This is given by the fact that this property belongs to a class of reachability properties that are much easier to check. On the other hand, search for Robustness or shortest time series walks adds some overhead (if employed). These procedures rise in complexity with length of the time series walk - the longer is 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 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] H. Klarner, H. Siebert, and A. Bockmayr. Time series dependent analysis of unparametrized thomas networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 99(PrePrints), 2012.

[6] 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.

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