Petri Net Basis Reachability Space Generator
User Quick Guide

Zou, Minqiang & Ma, Ziyue

March 25, 2021

Contents

1 General Information **2**

2 PBSG Quick Guide **2**

2.1 Some Basics of Petri Nets 2

2.2 Input the Net 2

2.3 Input the Set of Explicit Transitions 4

2.4 Compute the Basis Reachability Graph 4

3 Contact Us **5**

References **6**

1 General Information

Petri Net Basis Reachability Space Generator (PBSG) is a software for the computation of the basis reachability graph of a Petri net. PBSG has been developed in the Lab of Systems Control & Automation, a research group of School of Electro-Mechanical Engineering, Xidian University.

The technical details of the Basis Reachability Graph (as well as basis markings and so on) can be found in the research paper [1]. The preliminary version of basis marking technique is reported in [2, 3]. In [4] the first "basis reachability graph" constructed for the purpose of fault diagnosis in Petri nets. The advantages of this technique is that only part of the reachability space, the so-called *basis markings*, is enumerated. All other markings in the reachability set can be characterized by a linear algebraic system. Hence, in many Petri net problems using basis markings can reduce the computational load and greatly reduce the state explosion problem. So far, the basis marking approach has been effectively used to solve the diagnosability [5], controllability [6], and opacity problems [7] in Petri nets.

2 PBSG Quick Guide

The main functions of the PBSG software are quite self-explanatory. In brief, to use PBSG, you need to input a Petri net $\langle N, M_0\rangle$, input a set of explicit transitions T_E, click the button "Start", take a sip of coffee and see the result.

2.1 Some Basics of Petri Nets

A Petri net is a four-tuple $N = (P, T, Pre, Post)$, where P is a set of m *places*, T is a set of n *transitions*, $Pre : P \times T \to \mathbb{N}$ and $Post : P \times T \to \mathbb{N}$ are the *pre-* and *post-incidence matrices* in $\mathbb{N}^{m\times n}$. A marked net $\langle N, M_0\rangle$ is a net N with an initial marking M_0, an m-component vector that assigns each place a non-negative integer number of tokens.

2.2 Input the Net

PBSG accepts two types of input for a net N with an initial marking M_0. The designated source file will be referred to as the Net File. The #places/#transitions in the net is limited to 99 at most.

From .ndr file (data file of TINA toolbox)

PBSG can take the Petri net $\langle N, M_0 \rangle$ from a designated data file of TINA toolbox. TINA (http://www.laas.fr/tina/) is a powerful tool for the editing and analysis of Petri Nets. You may draw a Petri net N in TINA with initial marking M_0, save it, and then select the saved .ndr file as the input in PBSG. PBSG will automatically read the net and the initial marking in the .ndr file. Note that when creating your net in TINA please do not add temporal information (i.e., time) on transitions to your net.

From .txt file (plain text file)

PBSG can take the net $\langle N, M_0 \rangle$ from a designated .txt file that is edible by any plain text processor such as Notepad. The text Net File must be in the following format.

- The first line contains two integers m and n that denote the number of places and transitions in your net, respectively.
- The next line is a comment line that contains a word "Pre".
- Then, in the next m lines, each line contains n integers (separated by comma). These lines denotes the Pre matrix.
- The next line is a comment line that contains a word "Post".
- Then, in the next m lines, each line contains n integers (separated by comma). These lines denotes the $Post$ matrix.
- The next line is a comment line that contains a word "M0".
- Then, the next line contains m integers (separated by comma) which denotes the initial marking M_0.

The three comment lines are just for the users' convenience to identify their input content: PBSG does not validate the content of the comment lines (so, after declaring m and n, you must sequentially place Pre, $Post$, and M_0 as described above). Although the content of the comment lines are not validated, **the lines must not be omitted**.

2.3 Input the Set of Explicit Transitions

PBSG takes the set of explicit transitions from a designated plain text file (.txt), called TE File. The TE File contains a single line that consists of the names of explicit transitions, separated by comma.

- If the net is input from an .ndr file, all names of explicit transitions in the TE File must occur in the net (case-sensitive).
- If the net is input from a .txt file, according to the columns of the Pre and $Post$ matrices, the name of the transitions in the net are automatically named ‘t00’, ‘t01’, ‘t02’, and so on. As a result, the name of explicit transitions in the TE File should be something like “t02, t05, ...” (letter ‘t’ case-sensitive).

2.4 Compute the Basis Reachability Graph

After selecting the Net File and the TE File, you can click the “Start” button to proceed. Before, please be sure that the “Terminate” button is released (otherwise the “Start” button is frozen).

When the “Start” button is clicked, PBSG will first read the files and validate the input data. PBSG can recognize common types of invalid inputs, e.g., the initial marking is not non-negative. In particular, PBSG can detect if the residue subnet of T_E (i.e., the *implicit subnet*) contains cycles: in such a case the designated set T_E is invalid.

If everything is fine, PBSG will start computing the basis reachability graph of the net $\langle N, M_0\rangle$ with T_E. The time it takes may vary which depends on (i) the size of the net, and (ii) the size of the corresponding basis reachability graph. During the computation, in the log window PBSG will continuously report the number of basis markings it has computed so far (for every 10,000 new basis markings).

Empirically, for a moderate size of net (8 places, 9 transitions) with 10^6 *basis markings, the computation is done in about 30 seconds on a desktop computer equipped with i5-3470 (a CPU launched in 2012) on Windows 10 platform. The peak memory usage in this run is 150 megabytes.*

On the other hand, **PBSG does not detect the boundedness of the net nor of the basis reachability graph.** It is known [4] that the basis reachability graph of an unbounded net may be infinite. In such a case, PBSG will not terminate. Since PBSG continuously reports the number of basis markings it has computed, if you observe in the log window that the number of basis markings has far exceeded your expectation and is still counting, then there may be a problem with your net.

In such a case (or for other reasons), you can clicking the ”Terminate” button to stop computing. It may take a while for PBSG to stop, and when done you will see a message in the log window. However, the ”Terminate” button will **NOT** release automatically. To start a new run, you need to click the ”Terminate” button again to release it, which defreezes the ”Start” button.

3 Contact Us

If you find any problems when using PBSG, please email to zou_minqiang@163.com (Minqiang Zou) and/or maziyue@xidian.edu.cn (Ziyue Ma).

References

[1] Z. Ma, Y. Tong, Z. Li, and A. Giua. Basis marking representation of Petri net reachability spaces and its application to the reachability problem. *IEEE Transactions on Automatic Control*, 62(3):1078–1093, 2017.

[2] A. Giua and C. Seatzu. Fault detection for discrete event systems using Petri nets with unobservable transitions. In *Proceedings of Joint 44th Conference on Decision and Control and 2005 European Control Conference*, pages 6323–6328, Seville, Spain, 2005.

[3] George Jiroveanu, René K. Boel, and Behzad Bordbar. On-line monitoring of large Petri net models under partial observation. *Discrete Event Dynamic Systems*, 18(3):323–354, 2008.

[4] M. Cabasino, A. Giua, and C. Seatzu. Fault detection for discrete event systems using Petri nets with unobservable transitions. *Automatica*, 46(9):1531–1539, 2010.

[5] N. Ran, H. Su, A. Giua, and C. Seatzu. Codiagnosability analysis of bounded Petri nets. *IEEE Transactions on Automatic Control*, 63(8):1192–1199, 2018.

[6] Z. Ma, Z. Li, and A. Giua. A method to verify the controllability of language specifications in Petri nets based on basis marking analysis. In *Proceedings of the 54th IEEE Conference on Decision and Control*, pages 1675–1681, Osaka, Japan, 2015.

[7] Y. Tong, Z. W. Li, C. Seatzu, and A. Giua. Verification of state-based opacity using Petri nets. *IEEE Transactions on Automatic Control*, 62(6):2823–2837, 2017.