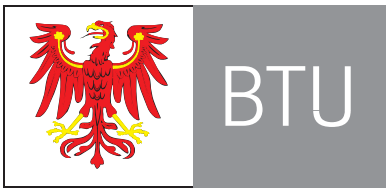
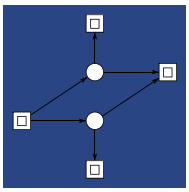


Brandenburgische Technische Universität Cottbus



Manual for Snoopy2Prism Export

Fei Liu

20 March 2009

Data Structures and Software Dependability
Department of Computer Science
Brandenburg University of Technology at Cottbus

Funding Number: 0315449H

1 Introduction

The function of the Snoopy2Prism Export is to transform stochastic Petri nets (SPN) produced by Snoopy [Sno] to files that can be read and analyzed by Prism software [Pri].

2 Main Features

The main features of the Snoopy2Prism Export can be summarized as follows:

2.1 Different boundedness setting modes for places

The Snoopy2Prism Export provides two kinds of boundedness setting modes according to the availability of boundedness information.

Setting boundedness maximum

When no boundedness information for places of stochastic Petri nets can be obtained from external files, the program will automatically set the boundedness of each place to a maximum constant. Then the user can arbitrarily assign a value to this maximum constant in the Prism software.

Setting boundedness information read from file

However, if the boundedness information can be obtained from other tools, such as IDD-CSL [IDD], the user will be asked to read the file containing the boundedness information and then the program will automatically set it in the export file. Besides, the program also sets a scale constant to boundedness of all the places, so that the user can scale up the boundedness.

2.2 Different ordering for places

The Snoopy2Prism Export provides six ways to order places so that the user can select among them to improve the efficiency of the Prism software. In the followings, we will explain these ordering ways with a simple stochastic Petri net, shown in Figure 1.

Plain ordering

In the plain ordering, the places of stochastic Petri nets will be exported in ascending order by identity (ID) assigned when they are modeled in Snoopy.

For the demo example, the plain ordering is: P1, P2, P3, P4, P5, P6.

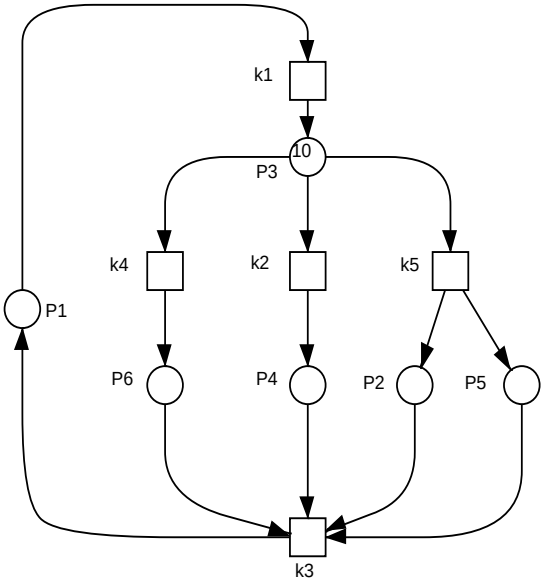


Figure 1: A simple stochastic Petri net. All transitions follow mass action kinetics with parameters k1=1, k2=0.4, k3=0.1, k4=0.5, k5=0.5.

Reverse ordering

In contrast with the plain ordering, in the reverse ordering the places of stochastic Petri nets will be exported in descending order by identity (ID) assigned when they are modeled in Snoopy.

For the demo example, the reverse ordering is: P6, P5, P4, P3, P2, P1.

Random ordering

In the random ordering, the places of stochastic Petri nets will be exported in a random way.

For the demo example, one random ordering is: P5, P2, P4, P3, P1, P6.

Noack1 ordering

Noack1 ordering [Noa99] adopts a greedy algorithm, which uses the structural information of the Petri net and suggests that variables assigned to adjacent places of a transition should lie close to each other in the ordering [Tov08]. The ordering π is constructed in a bottom-up manner. Assuming that $x_1 <_\pi x_2 <_\pi \cdots <_\pi x_{|P|}$,where $|P|$ is the number of the places, we assign places to the variables starting from the variable $x_{|P|}$. To select a place for a variable x_i, we compute weights $w(p)$ for all places $p \in P \cap \overline{S}$, where P denotes a set of all the places, and S denotes a set of places already assigned to some variable. A place

p with the highest weight $w(p)$ is assigned then to the variable x_i. The S and $w(p)$ can be expressed as follows.

$$S = \bigcup_{i<j\leq|P|} Pl(x_j)$$

where $Pl(x_j)$ represents the place that x_j denotes.

$$w(p) = \frac{\sum_{\substack{t\in{}^{\cdot}p,\\ |{}^{\cdot}t|>0,\\ |t^{\cdot}|>0}} (\frac{0.1f(t)+M(t)}{|{}^{\cdot}t|} + \frac{2U(t)}{|t^{\cdot}|}) + \sum_{\substack{t\in p^{\cdot},\\ |{}^{\cdot}t|>0,\\ |t^{\cdot}|>0}} (\frac{0.2g(t)+2U(t)}{|t^{\cdot}|} + \frac{M(t)+1}{|{}^{\cdot}t|})}{|{}^{\cdot}p \cup p^{\cdot}|}$$

where

$$f(t) = \begin{cases} 0, & M(t) > 0 \\ 1, & M(t) = 0 \end{cases}$$

$$g(t) = \begin{cases} 0, & U(t) > 0 \\ 1, & U(t) = 0 \end{cases}$$

$$M(t) = \sum_{p_s\in S} m(t)$$

$$U(t) = \sum_{p_s\in S} u(t)$$

where

$$m(t) = \begin{cases} 1, & t \in{}^{\cdot} p_s \\ 0, & \text{else} \end{cases}$$

$$u(t) = \begin{cases} 1, & t \in p_s{}^{\cdot} \\ 0, & \text{else} \end{cases}$$

This heuristics allows us to compute automatically quite good ROIDD variable ordering for most of the nets we face.

For the demo example, the results of Noack1 weight calculating for each step is shown in Table 1, where * implies that the place has been ordered. First the program calculated the weight for each place only in terms of the initial value. Because P3 has the highest weight (0.9) then it was ordered. After that, the program continued to compute the weight for the remaining places, and then P1(1.5125), P2(1.625), P5(2.25), P4(1.875), P6(2) were also ordered. Then the Noack1 ordering of the demo example was gotten, which was: P6, P4, P5, P2, P1, P3.

Table 1: Weight calculating results of Noack1

Step	P1	P2	P3	P4	P5	P6
1	0.6125	0.275	0.9	0.275	0.275	0.275
2	1.5125	0.725	*	0.725	0.725	0.725
3	*	1.625	*	1.625	1.625	1.625
4	*	*	*	1.75	2.25	1.75
5	*	*	*	1.875	*	1.875
6	*	*	*	*	*	2

Table 2: Weight calculating results of Noack2

Step	P1	P2	P3	P4	P5	P6
1	0.6625	0.3	0.925	0.325	0.3	0.325
2	1.56	0.75	*	0.775	0.75	0.775
3	*	1.65	*	1.675	1.65	1.675
4	*	1.775	*	*	1.775	1.8
5	*	1.9	*	*	1.9	*
6	*	*	*	*	2.5	*

Noack2 ordering

Noack2 ordering [Noa99] adopts the same greedy algorithm, which uses the structural information of Petri nets and suggests that variables assigned to adjacent places of a transition should lie close to each other in the ordering. The minute difference between Noack1 and Noarck2 only lies in the weight calculating. For the Noack2, the $w(p)$ is calculated as follows.

$$w(p) = \frac{\sum_{\substack{t\in {}^{\cdot}p,\\ |{}^{\cdot}t|>0}} \frac{0.1f(t)+M(t)}{|{}^{\cdot}t|} + \sum_{\substack{t\in {}^{\cdot}p,\\ |t^{\cdot}|>0}} \frac{0.1g(t)+2U(t)}{|t^{\cdot}|} + \sum_{\substack{t\in |p^{\cdot}|,\\ |{}^{\cdot}t|>0,\\ |t^{\cdot}|>0}} \left(\frac{0.2g(t)+2U(t)}{|t^{\cdot}|} + \frac{M(t)+1}{|{}^{\cdot}t|}\right)}{|{}^{\cdot}p \cup p^{\cdot}|}$$

For the demo example, the results of Noack1 weight calculating for each step is shown in Table 2. The program performed similar steps to the Noack1 ordering, and the resulted Noack2 ordering was gotten, which was: P5, P2, P6, P4, P1, P3.

From the results of Noack1 and Noack2 ordering, it can be seen that both of Noack1 and Noack2 ordering follow the general idea given above, but they are still somewhat different because of their slight difference in the weight calculating formula.

User-defined ordering

In the user-defined ordering, the user is asked to provide files in which the places have been ordered with user-defined ordering modes.

2.3 Transforming different types of functions for transitions

The user can set different types of functions for transitions, such as Main, BioMassAction and BioLevelInterpretatio in Snoopy, and all these functions will be automatically transformed to Prism files.

2.4 Error handling and locating for input files

The Snoopy2Prism Export can find errors in input files such as files containing bounedness files or user-defined ordering files, and also can locate and handle the errors in the files.

Bibliography

[Pri] Prism, *http://www.prismmodelchecker.org*

[Sno] Snoopy, *http://www-dssz.informatik.tu-cottbus.de*

[IDD] IDD-CSL, *http://www-dssz.informatik.tu-cottbus.de*

[Tov08] A. Tovchigrechko, *Efficient symbolic analysis of bounded Petri nets using interval decision diagrams (submitted version)*, BTU Cottbus, Ph. D. Thesis, October 2008

[Noa99] A. Noack, *A ZBDD Package for Efficient Model Checking of Petri Nets*, BTU Cottbus, Technical report, 1999

Appendix: Snoopy2Prism export file

In this appendix, the generated Snoopy2Prism export file for the demo example is listed, in which the boundedness is set to a maximum and the Noack1 ordering is selected.

```
ctmc
const int Max;
const int N;
module  Noack1export

P6:  [ 0..Max ]  init 0;
P4:  [ 0..Max ]  init 0;
P5:  [ 0..Max ]  init 0;
P2:  [ 0..Max ]  init 0;
P1:  [ 0..Max ]  init 0;
P3:  [ 0..Max ]  init 10*N;

[k1]
(P1 >= 1) & (P3 <= Max-1 )
 -> 1 * P1 :
(P1' = P1-1) & (P3' = P3+1);

[k2]
(P3 >= 1) & (P4 <= Max-1 )
 -> 0.4 * P3 :
(P3' = P3-1) & (P4' = P4+1);

[k3]
(P2 >= 1) & (P4 >= 1) & (P6 >= 1) & (P5 >= 1) & (P1 <= Max-1 )
 -> 0.1 * P2 * P4 * P6 * P5 :
(P2' = P2-1) & (P4' = P4-1) & (P6' = P6-1) & (P5' = P5-1) & (P1' = P1+1);

[k4]
(P3 >= 1) & (P6 <= Max-1 )
 -> 0.5 * P3 :
(P3' = P3-1) & (P6' = P6+1);

[k5]
(P3 >= 1) & (P2 <= Max- 1 ) & (P5 <= Max-1 )
 -> 0.5 * P3 :
(P3' = P3-1) & (P2' = P2+1) & (P5' = P5+1);

endmodule
```