Symbolic Computation of Petri Nets

Andres Iglesias[1] and Sinan Kapcak[2]

[1] Department of Applied Mathematics and Computational Sciences,
University of Cantabria, Avda. de los Castros,
s/n, E-39005, Santander, Spain
iglesias@unican.es
http://personales.unican.es/iglesias
[2] Department of Mathematics, Izmir Institute of Technology,
Urla, Izmir, Turkey
sinankapcak@iyte.edu.tr

Abstract. Petri nets are receiving increasing attention from the scientific community during the last few years. They provide the users with a powerful formalism for describing and analyzing a variety of information processing systems such as finite-state machines, concurrent systems, multiprocessors and parallel computation, formal languages, communication protocols, etc. Although the mathematical theory of Petri nets has been intensively analyzed from several points of view, the symbolic computation of these nets is still a challenge, particularly for general-purpose computer algebra systems (CAS). In this paper, a new *Mathematica* package for dealing with some Petri nets is introduced.

1 Introduction

Petri nets (PN) are receiving increasing attention from the scientific community during the last few years. Most of their interest lies on their ability to represent a number of events and states in a distributed, parallel, nondeterministic or stochastic system and to simulate accurately processes such as concurrency, sequentiality or asynchronous control [1,3]. Petri nets provide the users with a very powerful formalism for describing and analyzing a broad variety of information processing systems both from the graphical and the mathematical viewpoints. Since its inception in the early 60s, they have been successfully applied to many interesting problems including finite-state machines, concurrent systems, multiprocessors and parallel computation, formal languages, communication protocols and many others.

Although the mathematical fundamentals of Petri nets have been analyzed by using many powerful techniques (linear algebraic techniques to verify properties such as place invariants, transition invariants and reachability; graph analysis and state equations to analyze their dynamic behavior; simulation and Markov-chain analysis for performance evaluation, etc.), and several computer programs for PN have been developed so far, the symbolic computation of these nets is still a challenge, particularly for general-purpose computer algebra systems (CAS).

Y. Shi et al. (Eds.): ICCS 2007, Part II, LNCS 4488, pp. 235–242, 2007.

In this paper, a new *Mathematica* package for dealing with some Petri nets is introduced. The structure of this paper is as follows: Section 2 provides a gentle introduction to the basic concepts and definitions on Petri nets. Then, Section 3 introduces the new *Mathematica* package for computing them and describes the main commands implemented within. The performance of the code is also discussed in this section by using some illustrative examples. Conclusions and further remarks close the paper.

2 Basic Concepts and Definitions

A *Petri net* (PN) is a special kind of directed graph, together with an initial state called the initial marking (see Table 1 for the mathematical details). The graph of a PN is a bipartite graph containing *places* $\{P_1, \ldots, P_m\}$ and *transitions* $\{t_1, \ldots, t_n\}$. Figure 1 shows an example of a Petri net comprised of three places and six transitions. In graphical representation, places are usually displayed as circles while transitions appear as vertical rectangular boxes. The graph also contains arcs either from a place P_i to a transition t_j (*input arcs* for t_j) or from a transition to a place (*output arcs* for t_j). These arcs are labeled with their weights (positive integers), with the meaning that an arc of weight w can be understood as a set of w parallel arcs of unity weight (whose labels are usually omitted). In Fig. 1 the input arcs from P_1 to t_3 and P_2 to t_4 and the output arc from t_1 to P_1 have weight 2, the rest having unity weight.

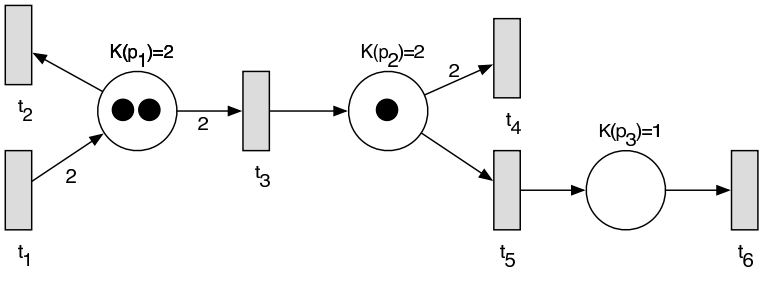

Fig. 1. Example of a Petri net comprised of three places and six transitions

A *marking* (state) assigns to each place P_i a nonnegative integer, k_i. In this case, we say that P_i *is marked with* k_i *tokens*. Graphically, this idea is represented by k_i small black circles (tokens) in place P_i. In other words, places hold tokens to represent predicates about the world state or internal state. All markings are denoted by vectors $\mathcal{M}$ of length m (the total number of places in the net) such that the i-th component of $\mathcal{M}$ indicates the number of tokens in place P_i. From now on the initial marking will be denoted as $\mathcal{M}_0$. For instance, the initial marking (state) for the net in Figure 1 is $\{2, 1, 0\}$.

Table 1. Mathematical definition of a Petri net

A Petri net (PN) is an algebraic structure $\mathcal{PN} = (\mathbf{P}, \mathbf{T}, \mathbf{A}, \mathbf{W}, \mathcal{M}_0)$ comprised of:

- a finite set of places, $\mathbf{P} = \{P_1, P_2, \ldots, P_m\}$,
- a finite set of transitions, $\mathbf{T} = \{t_1, t_2, \ldots, t_n\}$,
- a set of arcs, $\mathbf{A}$, either from a place to a transition (input arcs) or from a transition to a place (output arcs):
  $$\mathbf{A} \subseteq (\mathbf{P} \times \mathbf{T}) \bigcup (\mathbf{T} \times \mathbf{P})$$
- a weight function: $\mathbf{W} : \mathbf{A} \to \mathbb{N}^q$ (with $q = \#(\mathbf{A})$)
- an initial marking: $\mathcal{M}_0 : \mathbf{P} \to \mathbb{N}^m$

If $\mathcal{PN}$ is a finite capacity net, we also consider:

- a set of capacities, $\mathbf{C} : \mathbf{P} \to \mathbb{N}^m$
- a finite collection of markings (states) $\mathcal{M}_i : \mathbf{P} \to \mathbb{N}^m$

The dynamical behavior of many systems can be expressed in terms of the system states of their Petri net. Such states are adequately described by the changes of markings of a PN according to a *firing rule* for the transitions: a transition t_j is said to be *enabled* if each input place P_i of t_j is marked with $w_{i,j}$ tokens, where $w_{i,j}$ is the weight of the arc from P_i to t_j. For instance, transitions t_2, t_3 and t_5 are enabled, while transitions t_4 and t_6 are not. Note, for example, that transition t_4 has weight 2 while place P_2 has only 1 token, so arc from P_2 to t_4 is disabled. If transition t_j is enabled, it may or may not be fired (depending on whether or not the event represented by such a transition occurs). A firing of transition t_j removes $w_{i,j}$ tokens from each input place P_i of t_j and adds $w_{j,k}$ tokens to each output place P_k of t_j, $w_{j,k}$ being the weight of the arc from t_j to P_k. In other words, if transition t_j is fired, all input places of t_j have their input tokens removed and a new set of tokens is deposited in the output places of t_j according to the weights of the arcs connecting those places and t_j. For instance, transition t_3 removes one token from place P_1 and adds one token to place P_2, thus changing the previous marking of the net.

A transition without any input place is called a *source transition*. Note that source transitions are always enabled. In Figure 1 there is only one source transition, namely t_1. A transition without any output place is called a *sink transition*. The reader will notice that the firing of a sink transition removes tokens but does not generate new tokens in the net. Sink transitions in Figure 1 are t_2, t_4 and t_6. A couple (P_i, t_j) is said to be a self-loop if P_i is both an input and an output place for transition t_j. A Petri net free of self-loops is called a *pure* net. In this paper, we will restrict exclusively to pure nets.

Some PN do not put any restriction on the number of tokens each place can hold. Such nets are usually referred to as *unfinite capacity net*. However, in most practical cases it is more reasonable to consider an upper limit to the number of tokens for a given place. That number is called the *capacity* of the place. If all

places of a net have finite capacity, the net itself is referred to as a *finite capacity net*. All nets in this paper will belong to this later category. For instance, the net in Figure 1 is a finite capacity net, with capacities 2, 2 and 1 for places P_1, P_2 and P_3, respectively.

If so, there is another condition to be fulfilled for any transition t_j to be enabled: the number of tokens at each output place of t_j must not exceed its capacity after firing t_j. For instance, transition t_1 in Figure 1 is initially disabled because place P_1 has already two tokens. If transitions t_2 and/or t_3 are applied more than once, the two tokens of place P_1 will be removed, so t_1 becomes enabled. Note also that transition t_3 cannot be fired initially more than once, as capacity of P_2 is 2.

3 The *Mathematica* Package for Petri Nets

In this section a new *Mathematica* package for dealing with Petri nets is introduced. For the sake of clarity, the main commands of the package will be described by means of its application to some Petri net examples. In particular, in this paper we will restrict to the case of pure and finite capacity nets, a kind of nets with many interesting applications. We start our discussion by loading the package:

```
In[1]:= <<PetriNets‘
```

According to Table 1, a Petri net (like that in Figure 1 and denoted onwards as `net1`) is described as a collection of lists. In our representation, `net1` consists of three elements: a list of couples $\{place, capacity\}$, a list of transitions and a list of arcs from places to transitions along with its weights:

```
In[2]:= net1={{{p1,2},{p2,2},{p3,1}},{t1,t2,t3,t4,t5,t6},
              {{p1,t1,2},{p1,t2,-1},{p1,t3,-2},{p2,t3,1},
               {p2,t4,-2},{p2,t5,-1},{p3,t5,1},{p3,t6,-1}}};
```

Note that the arcs are represented by triplets $\{place, transition, weight\}$, where positive value for the weights mean output arcs and negative values denote input arcs. This notation is consistent with the fact that output arcs add tokens to the places while input arcs remove them. Now, given the initial marking $\{2, 1, 0\}$ and any transition, the `FireTransition` command returns the new marking obtained by firing such a transition:

```
In[3]:= FireTransition[net1,{2,1,0},t2];
```
Out[3]:= {1,1,0}

Given a net and its initial marking, an interesting question is to determine whether or not a transition can be fired. The `EnabledTransitions` command returns the list of all enabled transitions for the given input:

```
In[4]:= EnabledTransitions[net1,{2,1,0}];
```
Out[4]:= {t2,t3,t5}

The `FireTransition` command allows us to compute the resulting markings obtained by applying these transitions onto the initial marking:

```
In[5]:= FireTransition[net1,{2,1,0},#]& /@ %;
```
Out[5]:= {{*1,1,0*},{*0,2,0*},{*2,0,1*}}

Note that, since transition $t1$ cannot be fired, an error message is returned:

```
In[6]:= FireTransition[net1,{2,1,0},t1];
```
Out[6]:= FireTransition: Disabled transition: t1 cannot be fired for the given net and the {*2,1,0*} *marking.*

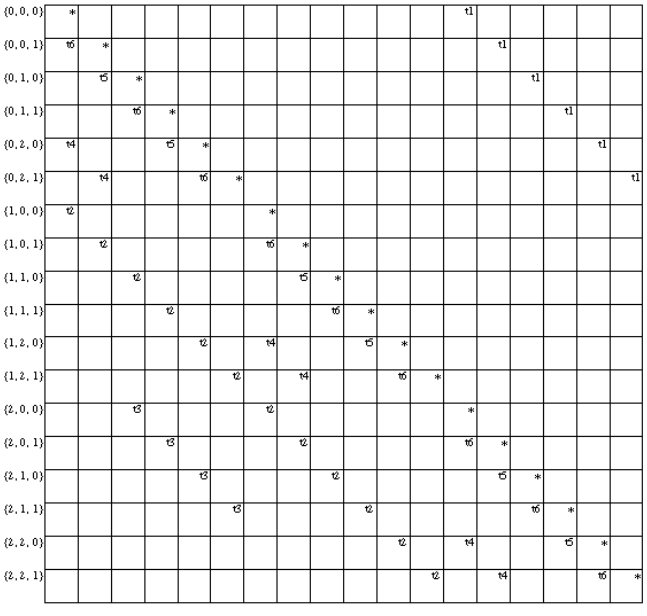

Fig. 2. The reachability graph for the Petri net `net1` and the initial marking $\{2, 1, 0\}$

From *Out[4]* and *Out[5]*, the reader can easily realize the successive application of the `EnabledTransitions` and `FireTransition` commands allows us to obtain all possible markings and all possible firings at each marking. However, this is a tedious and time-consuming task to be done by hand. Usually, such markings and firings are graphically displayed in what is called a *reachability graph*. The next input returns the reachability graph for our Petri net and its initial marking:

```
In[7]:= ReachabilityGraph[net1,{2,1,0}];
```
Out[7]:= See Figure 2

Figure 2 can be interpreted as follows: the outer column on the left provides the list of all possible markings for the net. Their components are sorted from the left to the right according to the standard lexicographic order. For any marking, the row in front gives the collection of its enabled transitions. For instance, the enabled transitions for the initial marking $\{2, 1, 0\}$ are $\{t2, t3, t5\}$ (as expected from *Out[4]*), while they are $\{t1, t4, t6\}$ for $\{0, 2, 1\}$. Given a marking and one

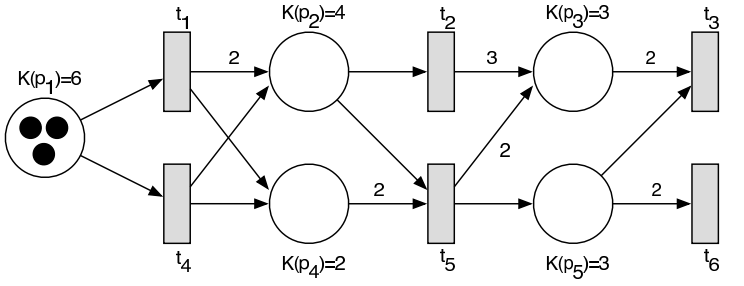

Fig. 3. Example of a Petri net comprised of five places and six transitions

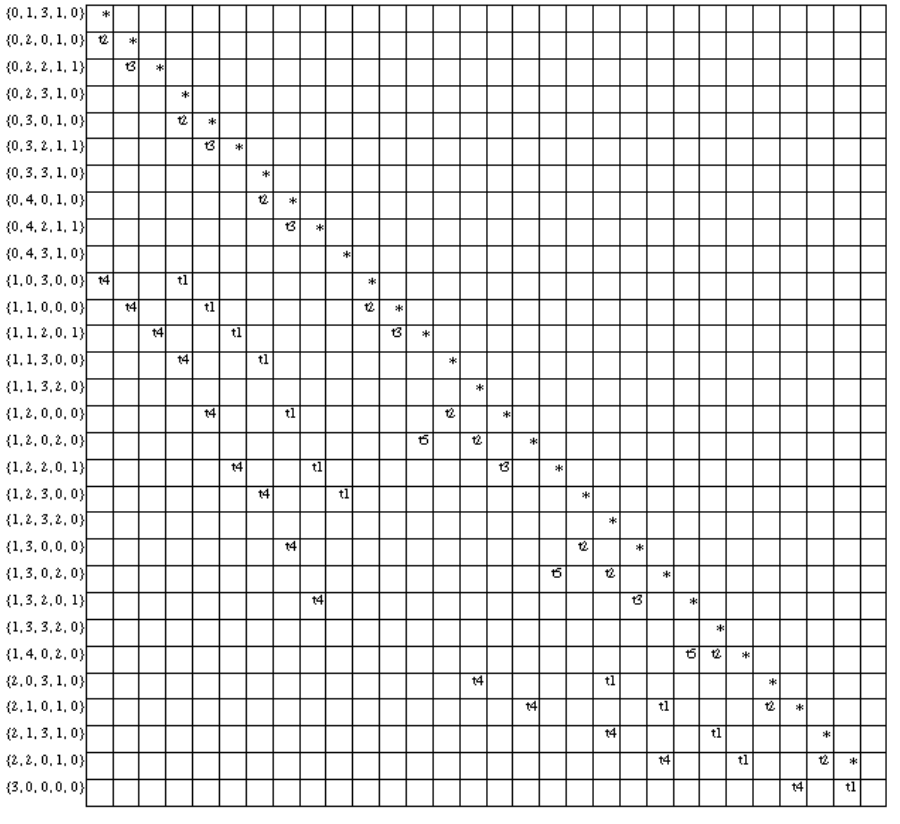

Fig. 4. Reachability graph for the Petri net in Figure 3

of its enabled transitions, you can determine the output marking of firing such transition by simply moving up/down in the transition column until reaching the star symbol: the marking in that row is the desired output. By this simple procedure, results such as those in *Out[5]* can readily be obtained.

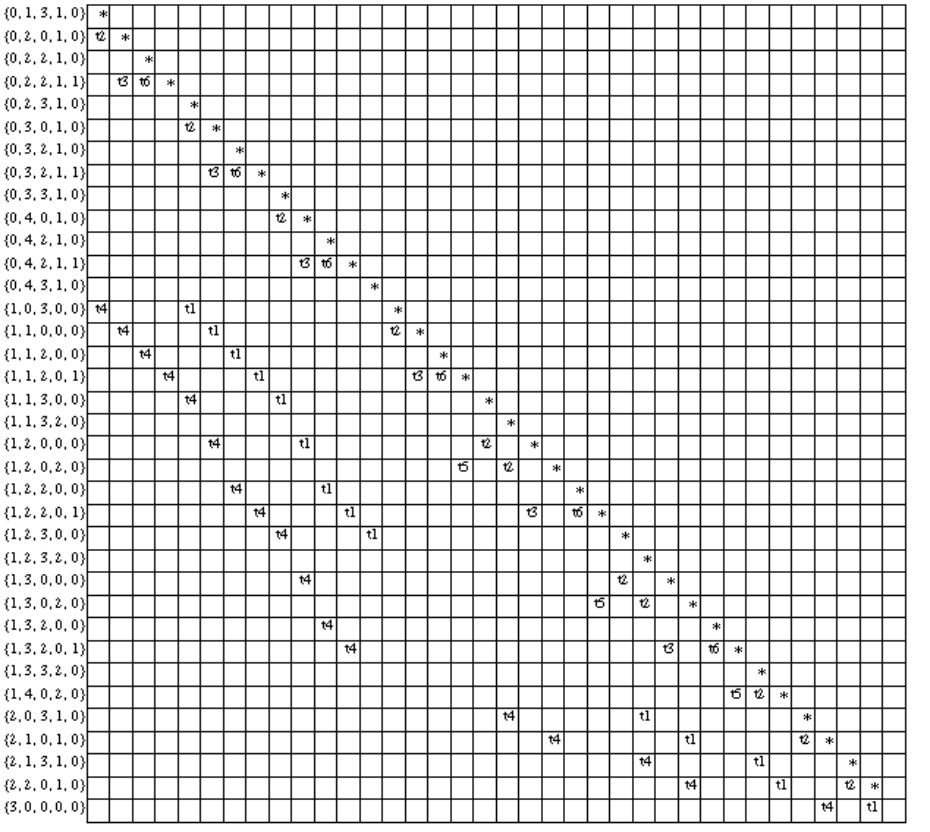

Fig. 5. Reachability graph after modifying the weight of the arc from P_5 to t_6

A second example of a Petri net is shown in Figure 3. This net, comprised of five place and six transitions, has many more arcs than the previous example. Consequently, its reachability graph, shown in Figure 4, is also larger. The *Mathematica* codes for defining the net and getting this graph are similar to those for the first example and, hence, have been intentionally omitted.

The net in Figure 3 exhibits a number of remarkable features: for instance, places P_1, P_2 and P_5 have more than one output transition, leading to nondeterministic behavior. Such a structure is usually referred to as a *conflict*, *decision* or *choice*. On the other hand, this net has no source transitions. This fact is reflected in the reachability graph, which has a triangular structure: entries appear only below the diagonal. As opposed to this case, the net in Figure 1 has one single source transition (namely, $t1$), the only element above the diagonal in its reachability graph.

It is worthwhile to mention that the place P_1 has only input arcs, meaning that its number of initial tokens can only decrease, but never increase. This means that the capacity of P_1 might be less without affecting current results.

On the other hand, the reachability graph in Figure 4 has some markings no transitions can be applied onto. Examples of such markings are $\{1,3,3,2,0\}$, $\{1,2,3,2,0\}$ or $\{0,4,3,1,0\}$ (although not the only ones). They are sometimes called *end markings*. Note that not end markings appear in the first net of this paper. Note also that the transition $t6$ is never fired (it never appears in the graph of Figure 4). By simply decreasing the weight of the arc from P_5 to t_6 to the unity, the transition becomes enabled, as shown in the new reachability graph depicted in Figure 5.

4 Conclusions and Further Remarks

In this paper, a new *Mathematica* package for dealing with finite capacity Petri nets has been introduced. The main features of the package have been discussed by its application to some simple yet illustrative examples. Our future work includes the application of this package to real problems, the extension to other cases of Petri nets, the implementation of new commands for the mathematical analysis of these nets and the characterization of the possible relationship (if any) with the functional networks and other networked structures [2,4,5].

This research has been supported by the Spanish Ministry of Education and Science, Project Ref. #TIN2006-13615. The second author also thanks the financial support from the Erasmus Program of the European Union for his stay at the University of Cantabria during the period this paper was written.

References

1. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE, **77**(4) (1989) 541-580
2. Echevarría, G., Iglesias, A., Gálvez, A.: Extending neural networks for B-spline surface reconstruction. Lectures Notes in Computer Science, **2330** (2002) 305-314
3. German, R.: Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets. John Wiley and Sons, Inc. New York (2000)
4. Iglesias, A., Gálvez, A.: A New Artificial Intelligence Paradigm for Computer-Aided Geometric Design. Lectures Notes in Artificial Intelligence, **1930** (2001) 200-213
5. Iglesias, A., Echevarría, G., Gálvez, A.: Functional networks for B-spline surface reconstruction. Future Generation Computer Systems, **20**(8) (2004) 1337-1353