Petri nets analysis using incidence matrix method inside ATOM^3

Alejandro Bellogín Kouki
Universidad Autónoma de Madrid
alejandro . bellogin @ uam . es

June 13, 2008

1 Introduction

As Murata said in [3]:

> Petri nets are a graphical and mathematical modeling tool applicable to many systems. (...) As a graphical tool, Petri nets can be used as a visual-communication aid similar to flow charts, block diagrams and networks. (...) As a mathematical tool, it is possible to set up state equations, algebraic equations and other mathematical models governing the behaviour of systems.

In this work we focus in the mathematical point of view, and we develop some applications that can be derived from it in a meta-modelling and model-transforming tool called ATOM^3.

Analysis methods for Petri nets may be classified into the following three groups:

1. Coverability (reachability) tree method
2. Matrix-equation approach
3. Reduction or decomposition techniques

The first method should be able to apply to all classes of nets, but is limited to small ones due to the complexity of the state-space explosion. On the other hand, the last two methods are very powerful but only applicable to special situations or subclasses of Petri nets.

The notation used herein is the following: the set P is the set of places, T is the set of transitions and F is the set of all arcs of a Petri Net. Besides, the number of tokens contained in a place p_i is represented as $M(p_i) = M_i$, and the net marking is defined as the vector of the markings for each place, i.e. $M = (M_1, \cdots, M_n)$.

2 Preliminary steps

Given a Petri net (from now on PN), if we want to analyse it from an algebraic point of view, we need to get its *incidence matrix*. It is an $n \times m$ matrix, for a PN with n transitions and m places, and its typical entry is given by

$$a_{ij} = a_{ij}^{+} + a_{ij}^{-}$$

where $a_{ij}^{+} = w(i,j)$ is the weight of the arc from transition i to its output place j, and $a_{ij}^{-} = w(j,i)$ is the weight of the arc from transition i from its input place j. This leads to the following state equation for a Petri net:

$$M^k = M^{k-1} + A^t u_k$$

where $A = [a_{ij}]$ is the incidence matrix, u_k is an $n \times 1$ column vector of $n-1$ 0's and one nonzero entry, indicating the transition fired, and M^j is the net marking after j transitions have been fired.

In this work, we are going to show some interesting properties easily derived from this incidence matrix, since it is a very powerful technique. For example, it is easy to see that transition i is enabled at marking M if and only if $a_{ij}^{-} \leq M(j), \forall j \in P$.

For the following sections, we need to know the incidence matrix, as well as the input matrix ($[a_{ij}^{-}]$) and the output matrix ($[a_{ij}^{+}]$), therefore, the first things our program has to collect are these matrices.

Another technical detail we have to deal with is to store an unambiguous relation between the vectors used in the algebraic operations and the names used by the user, in order to return human-readable and understandable information from our analysis. For this reason, we use (Python's) dictionary data type in a proper manner to obtain a biyection between vector's coordinates and transition/place's names.

In the next sections, we are going to explain how we have used these matrices in order to obtain results with using the techniques herewith listed:

- Subclassification or detection of particular structures
- Invariant analysis
- Detection of siphons and traps
- Reachability to a marking

In this list, some of the properties listed are structural properties (because they only depend on the topological structure, they do not depend on the initial marking), such as subclassification and detection of siphons and traps.

In table 1 we can see some tools and libraries used and/or tested during the development of this work.

Name	**Functionality**	**Finally used**
Matlab	System solver	No
Numarray	Python's library to work with matrices	Yes
Pipe	Tool for creating and analysing PNs	Yes
slepc4py	Python's library to wrap SLEPc eigensolver package	No

Table 1: Tools or libraries checked

3 Subclassification of Petri nets

There are some well-known, interesting properties only possible to study in some classes of PNs. In this section, we are going to show how we have used the incidence matrix to classify its corresponding net.

We use the following notation:

$\bullet t = \{p : (p,t) \in F\}$ is the set of input places of t
$t\bullet = \{p : (t,p) \in F\}$ is the set of output places of t
$\bullet p = \{t : (t,p) \in F\}$ is the set of input transitions of p
$p\bullet = \{t : (p,t) \in F\}$ is the set of output places of p

All Petri nets considered in this section are ordinary, what means that all of its arc weights are 1's. In [2] we can find references for converting generalized PN into ordinary PN. Actually, this transformation is always possible.

3.1 State machine

In this kind of net, each transition has exactly one input place and exactly one output place:

$$|\bullet t| = |t \bullet| = 1 \forall t \in T$$

This condition, given the input and output matrices, is very simple:

$$\forall i \quad \sum_j |a_{ij}^{\triangleright}| = 1, \quad \triangleright \in \{-, +\}$$

In other words, in all the *rows* of input and output matrices must be a nonzero element, and only one.

3.2 Marked graph

Each place has exactly one input transition and exactly one output transition:

$$|\bullet p| = |p \bullet| = 1 \forall p \in P$$

This is very similar to the previous case:

$$\forall j \quad \sum_i |a_{ij}^{\triangleright}| = 1, \quad \triangleright \in \{-, +\}$$

Equivalent to the following restriction: in all the *columns* of input and output matrices must be a nonzero element, and only one. Actually, this property is equivalent to check if the *transpose net*[1] is a state machine.

3.3 Free-choice net

In these nets, every arc from a place is either a unique outgoing arc or a unique incoming arc to a transition:

$$\forall p \in P, |p \bullet| \leq 1 \text{ or } \bullet (p\bullet) = \{p\}$$

We can not express this condition in a simpler way, but we can give an algorithm:

1. For each place in the input matrix (output place is equivalent to input transition):
   (a) If it has more than one element nonzero (the place has two or more output transitions):
      i. For each transition from this place, check that it has only one element nonzero in its row (its only input is this place). If it is not true, the net is not a free-choice net.

[1] Where places play the role of transitions and viceversa, i.e. its incidence matrix is the transpose of the original matrix.

3.4 Extended free-choice net

It is a PN such that

$$p_1 \bullet \cap p_2 \bullet \neq \emptyset \Rightarrow p_1 \bullet = p_2 \bullet \quad \forall p_1, p_2 \in P$$

Like in the previous case, we can only give an algorithm:

1. For each place p in the input matrix:
   (a) For each place $q \neq p$:
      i. Check that if their set of output transitions have empty intersection. If it is the case, continue. If not, the intersection has to be equal to the set of output transition of each place, i.e. both places must have the same output transitions, if it is not true, this net is not an extended free-choice net.

3.5 Asymmetric choice net

Finally, this type of net is characterised by the following equation:

$$p_1 \bullet \cap p_2 \bullet \neq \emptyset \Rightarrow p_1 \bullet \subseteq p_2 \bullet \text{ or } p_1 \bullet \supseteq p_2 \bullet \quad \forall p_1, p_2 \in P$$

This is a more relaxed form of the previous net, where the last *equal* condition is smoothed by the subset condition:

1. For each place p in the input matrix:
   (a) For each place $q \neq p$:
      i. Check that if their set of output transitions have empty intersection. If it is the case, continue. If not, the set of output transition of one place must contain the other, or viceversa, if it is not true, this net is not an asymmetric choice net.

4 Invariant analysis

The problem of invariance can be defined as (given the incidence matrix A):

- x is a T-invariant (transition invariant) if

$$A^t x = 0$$

- y is a P-invariant (place invariant) if

$$Ay = 0$$

That is, we want to find some properties (equations) in places and transitions that stay constant in the net. In this work, we have only considered to find place invariants, since we think these are more interesting than transition invariants, but it is important to notice that the problem is equivalent, since one system of equations is the transposition of the other.

At first sight, we thought this problem was a simple eigenvector problem, where the invariants are the eigenvectors. But we found two problems:

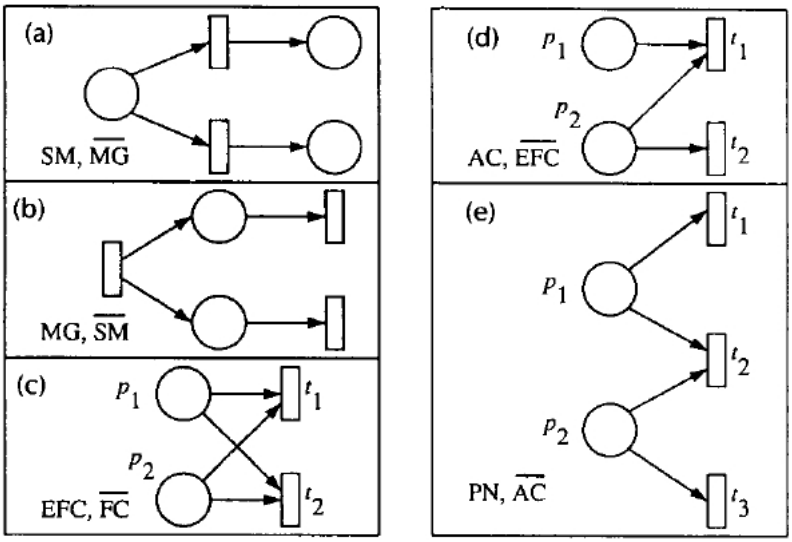

Figure 1: Examples of different classification for some PNs. SM stands for state machine, MG for marked graph, FC for fre-choice, EFC for extended free-choice and AFC for asymmetric free-choice. The last one is out of any of this category, and it belongs to the more general Petri net category.

1. The incidence matrix is not, generally, a square matrix. Therefore, the general algorithm to find eigenvalues is not directly applicable. This can be partially overcome calculating the pseudoinverse, but there would be still some problems when calculating the characteristic equation.

2. Even in the square matrix situation, the eigenvector found does not have the information we want. For example, in the net shown in figure 2, the sum of tokens between p1 and p4 is constant, so $M(p1) + M(p4) = 1$ is an invariant. However, the matrix of eigenvectors (as columns) is

$$V = \begin{pmatrix} -0.0000 & -0.6325 & 0.8165 & 0.0000 \\ 0.5000 & -0.3162 & -0.4082 & -0.5000 \\ -0.5000 & -0.3162 & -0.4082 & 0.5000 \\ 0.7071 & -0.6325 & -0.0000 & 0.7071 \end{pmatrix}$$

   The only column that can suggest this invariant is the second, in which the first and the fourth component have the same value. So, given this matrix, it is not straightforward the translation into (place) invariants.

One algorithm found in [2] was tried, but the final version uses PIPE library[2] to find the minimal generating set of all place invariants, since it worked properly in all tested examples. It uses the algorithm proposed by D'Anna and Trigila in [1] (very good performance).

5 Siphons and traps

First of all, we have to define what these concepts mean:

[2] pipe2.sourceforge.net

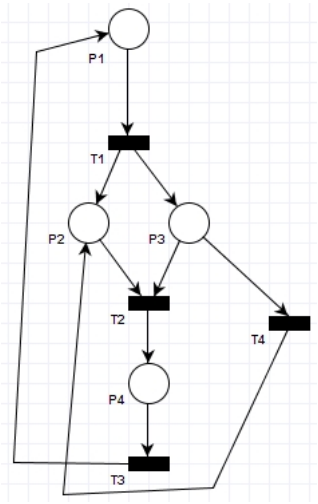

Figure 2: A PN with 4 places and transitions.

Siphon (or *deadlock*) is a set of places S such that every transition having an output place in S has an input place in S:

$$\bullet S \subseteq S\bullet$$

Trap is a set of places Q where every transition having an input place in Q has an output place in Q:

$$Q\bullet \subseteq \bullet Q$$

The first thing to notice is the word *set* in both definitions. This word implies the calculations over all the possible groups of places that can be created. Fortunately, the order is not important when creating these sets, therefore the number of possibilities is decreased. Actually, the total number of sets that have to be checked to be traps or siphons given a PN is

$$\sum_{k=1}^{m} \binom{m}{k} = \sum_{k=1}^{m} \frac{m!}{k!(m-k)!} = 2^m - 1, \text{ with } m = |P|$$

For each set of places, the sets of input and output transitions are generated. At this point, a simple comparison between sets (operation *subset of*) is carried out, and depending on the answer, we have a trap, a siphon or nothing at all[3].

6 Reachability to a marking

As we have already said, the problem of generating the reachability (coverability) graph of a specific PN is a very complex and time consuming task. This is the main motivation to try another technique for a similar objective: find if a marking is reachable given another initial marking. Although with this approach we will not have all the predictive capacity that the reachability graph can give, we can make some assumptions and prove if a marking is reachable under these assumptions. This can be seen equivalent to a *partial* reachability graph, but with the main benefit of ease and speed in its calculation.

[3]For example, in the problem of philosophers, our program finds 255 siphons and 255 traps when 4 philosophers come into play (this net has 8 transitions and 10 places).

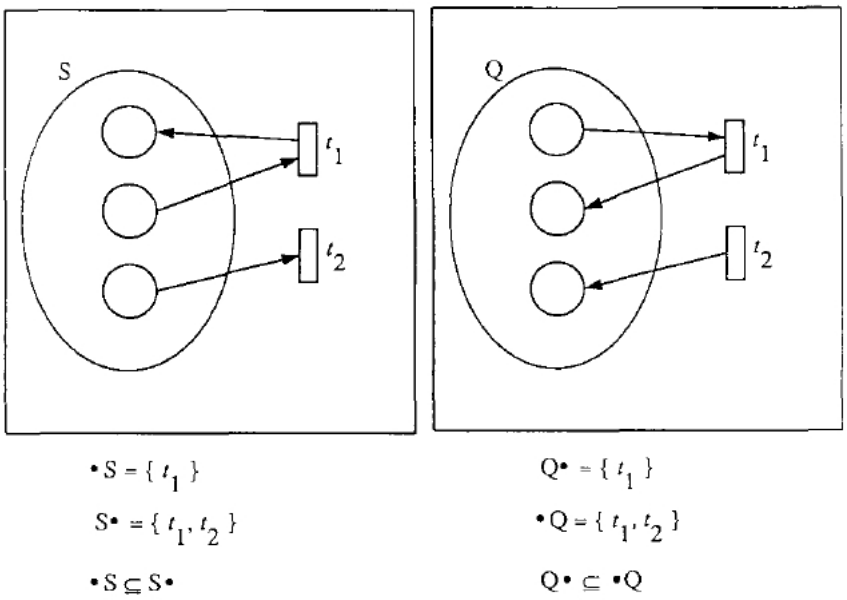

Figure 3: Examples of a siphon (left) and a trap (right).

We use the state equation mentioned earlier, that can be rewritten for a firing sequence (or Parikh vector)[4] as:

$$M^d = M^0 + A^t \sum_{k=1}^{d} u_k = M^0 + A^t \vec{U}$$

The first step in this process is to find all the sequences $\vec{U}$ that can potentially reach to the marking M^d in d firings. After that, we have to take into account the order, since the algebraic operation can leave the net in an inconsistent (impossible) state in some point of the sequence.

In order to make all this possible, we have to deal with two combinatorial problems:

- Calculate all the possible firing sequences. At this point we have to limit the number of times each transition can be executed.
- Calculate all the permutations of a given sequence of transitions, to consider the order.

Because of these two combinatorial problems, we study the performance of our solution for different configurations in the next section.

6.1 Performance notes

The goal of this section is to provide a view about the general performance expected by the system when calculating reachability to a marking, depending on some parameters, such as the number of places and transitions of the current PN, the maximum number of executions for each transition and the chosen marking (this can be a very important point, because if the marking is impossible, no Parikh vector will

[4] We need to consider a sequence of transitions because a marking should, and generally will, not be reachable only through one transition, but through a combination of several transitions.

be generated and the system's reply will be very fast, unlike the marking requires a lot of transitions, which forces to generate a huge amount of permutations).

Actually, the two numbers we have to consider are the following:

- The number of all the possible sequences: it is equivalent to the number of all possible variations (permutations with repetition) of L numbers in n positions; more precisely, this number is L^n, where $n = |T|$ and L the maximum number a transition can be executed.
- The number of permutations: $s!$ with s the number of transitions involved in a particular sequence.

We can see in table 2 some execution times and results for different nets and system's configurations. It is important to note that even for simple nets, the choice of L is crucial.

$\|P\|$	$\|T\|$	L	Time 1 (secs.)	Time 2 (secs.)	Valid sequences found	Max. length of sequence
3	4	1	1	1	1	3
3	4	3	1	4	1	7
3	4	5	1	38	2	8
3	4	10	1	38	2	8
4	3	1	1	1	1	2
4	3	3	1	1	1	6
4	3	5	1	540	1	9
4	3	10	2	460	0 (HE)	0
10	8	1	1	2 (HE)	8	7
10	8	2	1	860 (HE)	17	9
10	8	3	1	754 (HE)	11	7

Table 2: Time spent to obtain a solution for the *reachability to a marking* problem (if any). *Time 1* is the time to get all possible firing sequences, while *Time 2* is the time spend to check if the ordered sequences are valid. HE means that a *heap error* has occurred.

The combinatorial explosion shown in the table is a problem with no solution, but it could be possible to improve the performance of this analysis. We have not been able to obtain a better solution, but we leave it as an open problem to work in the future.

7 Future work

From all the analysis already explained, it could be possible to suggest some reduction rules (preserving system properties) to the user if some specific configurations have been found. For instance, if a self-loop place is found, the program could advise the user about this situation and she could do something appropriate. We find this is a valuable information from the user point of view, and that can be made easily analysing some properties about the incidence matrix.

Another thing already mentioned important is to continue working on the performance issue when deciding about the reachability to a marking.

8 Conclusions

Algebraic analysis is a very powerful technique, allowing interesting applications such as the ones seen in this work. Theoretically, it is more appropriate for big Petri nets than other analysis methods, like reach-

ability graph, but in practice, some of these applications require to work with too many combinations, what makes algebraic analysis unfeasible in these cases.

In this work, we have provided a dictionary between some properties that can be analised in a Petri net, and the equivalent equations/algorithms that the corresponding matrices (related with the Petri net) must hold.

References

[1] M. D. Anna and S. Trigila. Concurrent system analysis using petri nets: an optimized algorithm for finding net invariants. *Comput. Commun.*, 11(4):215–220, August 1988.

[2] René David and Hassane Alla. *Discrete, Continuous, and Hybrid Petri Nets*. Springer, 1 edition, November 2004.

[3] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.