Journal of Applied Sciences

ISSN 1812-5654

Journal of Applied Sciences 8 (24): 4668-4673, 2008
ISSN 1812-5654

A Polynomial-Time Decomposition Algorithm for a Petri Net Based on Indexes of Places

Q. Zeng, X. Hu, J. Zhu and H. Duan
College of Information Science and Engineering, Shandong University of Science and Technology,
579 Qianwangang Street, Huangdao, Qingdao 266510, China

Abstract: This study proposes an algorithm for the decomposition approach of Petri nets based on indexes of places and analyzes the complexity of the given algorithm. The main data structures required and four key functions contained in the algorithm are firstly addressed. It is proved that the proposed decomposition algorithm is a polynomial-time algorithm.

Key words: Petri net, index of place, decomposition, polynomial-time algorithm, complexity analysis

INTRODUCTION

As models for physical systems, Petri nets are well suited to describe and analyze systems with concurrency, synchronization and conflicts (Murata, 1989; Zeng and Duan, 2007; Wang *et al*., 2000). However, with the increase of the node number in a Petri net, its structure will be more complex, so it is difficult to analyze the properties of the net system. Traditionally, in order to overcome this difficulty, some solutions including decomposition, reduction, composition and net operation are introduced by many researchers. Lee *et al*. (1987) gives several generalized reduction methods of Petri nets. In Esparza (1994), a set of reduction rules are proposed that make it possible to reduce all and only live and bounded free choice Petri nets to a circuit containing one place and one transition. The reduction algorithm is shown to require polynomial time in the size of the system. In Samir (1999), two analytical decomposition techniques are proposed for computing the transient state space solution of large stochastic PN (SPN) models of MINs and HINs. A large scale SPN model is partitioned into smaller submodels. These submodels are compressed and combined to calculate the entire net.

Recently, Zeng (2007) proposed a decomposition method for structure-complex Petri nets based on the indexes of places. This decomposition method is very useful for property analysis of structure-complex Petri nets since the structure of the decomposition net is well-formed (Zeng, 2007). The language and process relations are analyzed during the decomposition and present a method to obtain the language of a Petri net in and to present the process of a structure-complex Petri net respectively. However, only the decomposition method is presented for a Petri net based on the indexes of places in all the research results (Zeng, 2007). The decomposition algorithm especially a polynomial-time decomposition algorithm for a Petri net based on indexes of places is not addressed in earlier study (Zeng, 2007).

In this study, a decomposition algorithm is proposed for a Petri net based on indexes of places and analyzes the time complexity of the given algorithm. The main data structures required by the decomposition algorithm and the key functions contained in the algorithm are firstly addressed. Finally, a polynomial-time decomposition algorithm for a Petri net is proposed.

DECOMPOSITION OF A PETRI NET BASED ON THE INDEXES OF PLACES

To save space, it is assumed that the readers are familiar with the basic definitions of Petri nets (Murata, 1989; Zeng and Duan, 2007; Wang *et al*., 2000). Some of the essential terminology and notations related to this study are defined as follows. Convenient to define, it is supposed that $K = \omega$ and $W = 1$ in the Petri net. And also assume that the Petri net discussed in this study is finite and connected and is not an S-Net (Zeng, 2004).

Definition 1: (Zeng, 2007): Let $\Sigma = (P,T; F,M_0)$ be a Petri net, a function $f: P \to \{1,2,..., k\}$ is said to be an index function defined on the place set if $\forall p_1, p_2 \in P$, $(p_1^\bullet \cap p_2^\bullet \neq \phi) \vee ({}^\bullet p_1 \cap {}^\bullet p_2 \neq \phi) \to f(p_1) \neq f(p_2)$. f(p) is named as the index of place p.

Definition 2: (Zeng, 2007): Let $\Sigma = (P,T; F,M_0)$ be a Petri net, $f: P \to \{1,2,..., k\}$ be the index function on the places of Σ. Perti net $\Sigma_i = (P_i,T_i; F_i,M_{0i})$ $(i \in \{1,2,..., k\})$ is said to be the decomposition net of Σ based on the index function f if Σ_i satisfies the following conditions.

Corresponding Author: Qingtian Zeng, College of Information Science and Engineering, Shandong University of Science and Technology, 579 Qianwangang Street, Huangdao, Qingdao 266510, China

$$P_i = \{p \in P \mid f(p) = i\} \tag{1}$$

$$T_i = \{t \in T \mid \exists p \in P_i, t \in {}^{\bullet}p \cup p^{\bullet}\} \tag{2}$$

$$F_i = \{(P_i \times T_i) \cup (T_i \times P_i)\} \cap F \tag{3}$$

$$M_{0i} = \Gamma_{P \to P_i} M_0 \tag{4}$$

Simply, Σ_i is said as the index decomposition net of Σ.

Definition 3: Zeng (2004): A Petri net $\Sigma = (P,T;F,M_0)$ is an S-Net iff $\forall t \in T: |{}^{\bullet}t| \leq 1$ and $|t^{\bullet}| \leq 1$.

Theorem 1: (Zeng, 2007): Let $\Sigma_i = (P_i,T_i;F_i,M_{0i})$ ($i \in \{1, 2,..., k\}$) be the index decomposition net based on index of place of a Petri net $\Sigma = (P,T;F,M_0)$, then Σ_i is an S-net.

A structure-complex Petri net is shown in Fig. 1. We use the decomposition method of Definition 2 to decompose Σ.

A function f is first defined on the place set such that $f(p_1) = f(p_6) = 1$, $f(p_2) = f(p_3) = 2$, $f(p_4) = f(p_5) = f(p_7) = 3$. It can prove that f satisfies all the conditions in Definition 1. Based on the method of Definition 2, three index decomposition net systems Σ_1, Σ_2 and Σ_3 are obtained and shown in Fig. 2. Obviously, Σ_1, Σ_2 and Σ_3 are S-Nets.

More discussions about this decomposition method can be found by Zeng (2004, 2007). The decomposition results with the method of definition 2.2 are usually not

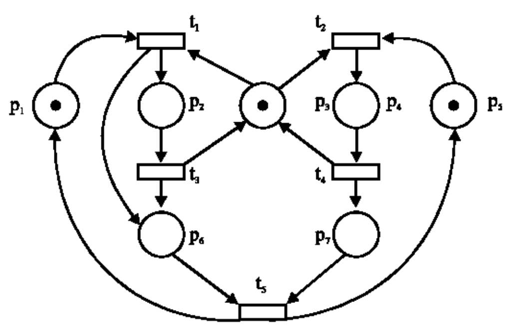


Fig. 1: A Petri net

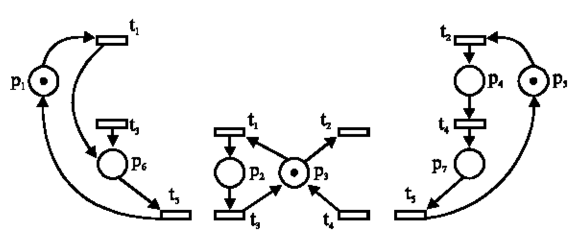


Fig. 2: Three decomposition subnets of the Petri net shown in Fig. 1

unique. With the results discussed by Zeng (2004, 2007), it is usually to take the case that k is with the minimal value as the best result. In the following sections, we will present a polynomial-time algorithm for the decomposition approach. Firstly, the main data structures and key functions contained in the algorithm will be addressed.

MAIN DATA STRUCTURES

Firstly, the main data structures to store each component of a Petri net are presented, including its flow relation, the input and output set of each transition, each place and its tokens.

Store of flow relation: Because any Petri net can be determined by its input matrix and output matrix[1], we use output $A^+ = [a_{ij}^+]_{n\times m}$ matrix and input matrix $A^- = [a_{ij}^-]_{n\times m}$ to represent the structure of a Petri net, where

$$a_{ij}^+ = \begin{cases} 1 & \text{if} \quad (t_i, s_j) \in F \\ 0 & \text{otherwise} \end{cases}$$

and

$$a_{ij}^- = \begin{cases} 1 & \text{if} (s_j, t_i) \in F \\ 0 & \text{otherwise} \end{cases}$$

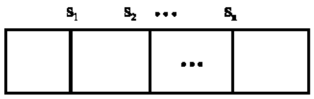
$A^+ = [a_{ij}^+]_{n\times m}$ and $A^- = [a_{ij}^-]_{n\times m}$ can be stored by a two-dimension array, respectively.

Store of the input and output set of each transition: The input set and output set of each transition are stored by a one-dimension array with |S|+1 length, respectively. An Array at represents the input set of a transition t, while bt represents the output set of t, which are shown as followings. If a place s_i belongs to at (or bt), the (i+1)th position in at (or bt) will be set as 1, otherwise be set as 0.

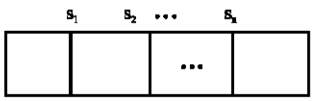
at	...	...	...

bt	...	...	...

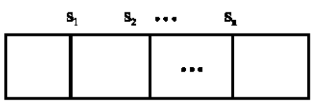
Store of a set X_k(k = 1, 2…): The places with same indexes will be put into a set X_k (k = 1, 2…) and X_k(K = 1,2…) is stored by a one-dimension array with |S| length. If the (i)th position in X_k is set as 1, it means s_i belongs to X_k(k = 1,2…). Otherwise, the corresponding position will be set as 0.



Store of index of each place: The index of each place is stored by a one-dimension array with |S| length, P_k (k = 1, 2…). If the index of s_i is l, the corresponding position of s_i in will be set as l in P_k (k = 1, 2…).



Store of tokens: A one-dimension array with |S| length, Q will be used to store all the tokens in the places. If s_i contains l tokens, the corresponding position of s_i in Q will be set as l.



Store of a set Y_u(u = 1, 2…): Another set Y_u(u = 1, 2…) are used to store the places of each decomposition net. Y_u(u = 1, 2…) is also represented by a one-dimension array with |S| length. If the (i)th position in Y_u is set as 1, it means s_i belongs to Y_u(u = 1, 2…) Otherwise, the corresponding position will be set as 0.

ALGORITHM DESIGN

The main point in the decomposition algorithm is to obtain the index of each place. According to the index of each place, the decomposition S-Nets can be obtained by the outface subnet of the places with same index. At the initialization step of the algorithm, we put all the places into set X_1. For each place in X_1, denoted by p, let $\lambda[p] = |\{p' \mid \exists t \in T, \{p, p'\} \subseteq {}^{\bullet}t \text{ or } \{p, p'\} \subseteq t^{\bullet}\}|$.

In the following step, we select a place p in X_l (l = 1, 2,...) such that $\lambda[p] \neq 0$ and move p from X_l to X_{l+1}. If there is a place moved out, the value $\lambda[p]$ for each place in X_l will be updated. If $\lambda[p]$ for each place in X_l is not 0, the selecting and moving operations will be repeated on X_l; Otherwise, repeat the selecting and moving operations on X_{l+1}. After the selecting and moving operations on completed on all sets, the places in one set can be assigned with one same index.

Key functions: Firstly, four key functions contained in the decomposition algorithm are presented which are Mark (X_k), Move (X_k,y), Divide (X_k) and Outface (Y_u).

Mark (X_k) //Obtain $\lambda[p]$ for each place in X_k
INPUT: X_k
OUTPUT: $\lambda[p]$ for each place in X_k
{
Step 0: for each $y \in X_k$, $\lambda[y] \leftarrow 0$. Let $i \leftarrow 1$.
Step 1: If i > |S|, halt.
Step 2: For each t_i, if there exists $x \in X_k$ and $x \neq y$ such that $y \in {}^{\bullet}t_i$ and $x \in {}^{\bullet}t_i$
DO $\lambda[y] \leftarrow \lambda[y]+1$
Step 3: If there exists $x \in X_k$ and $x \neq y$ such that ${}^{\bullet}x \cap {}^{\bullet}y \neq \emptyset$ and $x^{\bullet} \cap y^{\bullet} \neq \emptyset$,
go to step5
Step 4: For each t_i, if there exists $x \in X_k$ and $x \neq y$ such that $y \in t^{\bullet}_i$ and $x \in t^{\bullet}_i$,
DO $\lambda[y] \leftarrow \lambda[y]+1$
Step 5: Let $i \leftarrow i+1$, go to step1.
}

Move (X_k,y) // Move place y from X_k to X_{k+1}
INPUT: X_k and place y
OUTPUT: Set X_k and X_{k+1}
{
Put y into X_{k+1};
Search y in X_k and delete y from X_k.
}

Divide (X_k) //Divide X_k into Y_j, Y_{j+1}, Y_{j+2}…
INPUT: X_k
OUTPUT: Sets Y_j, Y_{j+1}, Y_{j+2}…
{
While (there exist places in X_k)
Do
{
Move the first place into Y_j;
for each place $x \in Y_j$
for each place $y \in X_k$
for i = 1 to |T|
if ($x \in {}^{\bullet}t_i$ and $y \in t^{\bullet}_i$) or ($x \in t^{\bullet}_i$ and $y \in {}^{\bullet}t_i$), then move y into Y_j
end for
end for
end for
j = j+1;
}
}

Outface (Y_j) //Output the outface subnet of places in Y_j
INPUT: Y_j
OUTPUT: Outface subnet N_j
{for each place $y \in X_k$
 for i =1 to |T|
 if there is an edge connecting t_i and y
  then t_i and y are connected
 end for
 end for }

Polynomial-time decomposition algorithm: A polynomial-time decomposition algorithm for Petri nets based on indexes of places is presented here.
INPUT: a Petri net
$\Sigma = (N, M_0) = (S, T; F, M_0)$
OUTPUT: Decomposition subnets of Σ
Step 1: // To obtain the presets and postset of each transition
for i = 1 to |T|
 for j = 1 to |S|
if there is an edge from S_j to t_i then $S_j \in {}^{\bullet}t_i$
 if there is an edge from t_i to S_j then $S_j \in t^{\bullet}_i$
  end for
 end for
Step 2. Store the markings of Σ in M
Step 3. Put all places of Σ into X_1
Step 4. Mark(X_1)
Step 5. // obtain the index of each place
 for k = 1 to |S|
  for j = 1 to |S|
 select $y \in X_k$ such that λ[y] is not 0;
  Move (X_k, y);
  Mark(X_k) //update λ[y]
  If λ[y] of each place in X_k is 0, then break; //quit and execute next loop;
  end for
  Mark(X_{k+1})
  If each λ[y] of place in X_{k+1} is 0, then break;
 end for
Step 6.// Divide X_1, X_2,.... into Y_1, Y_2,....
  for *i* = 1 to k
  Divide (X_k)
  end for
Step 7. //Output each subnet of Σ
  for j = 1 to |S|
  Outface (Y_j)
  end for
 Step 8.// Output markings of each subnet
  for *i* = 1 to |S|
   for j = 1 to |S|
  if s_i is in subnet N_j, then add M[k] tokens to s_i
  end for
 end for.

COMPLEXITY ANALYSIS OF THE ALGORITHM

Firstly, we analyze the complexity of four key functions required by the decomposition algorithm:

- In function Mark(X_k), each "if" loop executes finite number of judgments and assignments, so the time complexity is only related to the layers of loops. There are three "for" loops, so the time complexity of the full function is $O(n^2m)$, where n = |S| and m = |T|, and the same to the followings
- In function Move(X_k, y), the worst case of step "Search y in X_k and delete y from X_k" is to go through the whole set X_k, so the time complexity of this function is only O (n)
- In function Divide(X_k), from the number of layers of the "for" loops, it can be determined that the time complexity of this function is also $O(n^2 m)$
- In function Outface (Y_j), each "if" loop executes finite number of judgments and assignments, so its time complexity is O (nm)
  In the main algorithm, because the inner "for" loop executes finite number of judgments and assignments, the time complexity of the first step is O(nm). In Step 2 and Step 3, there are n times for assignments, so the time complexity of Step 2 and Step 3 is O(n), respectively. The time complexity of Step 4 is actually same to that of the function Mark(X_k), so it is $O(n^2 m)$. In Step 5, the step "select $y \in X_k$ such that λ[y] is not 0" is actually searching a place whose index is non-zero, so the worst time complexity of this step is O(n). The "if" loop is actually to go through the whole set X_k, so the time complexity of the inner "for" loop is $O(n^2 m)$. There are two outside layers of "for" loops, so the time complexity of Step 5 is $O(n^4 m)$. In Step 6, the inner function is Divide(X_k), the time complexity of this step is $O(n^3 m)$. The time complexity of Step 7 is mainly determined by the "for" loop and the function Outface(Y_j), so the time complexity of this step is $O(n^2 m)$. In Step 8, there are n times for search and assignments, so the time complexity of this Step is $O(n^2)$
- According to the time complexity of each step in the main algorithm, the time complexity of the whole algorithm is $O(nm+n+n+n^2m+n^4m+n^3 m+n^2m+n^2)$ $=O(n^4m)$. Therefore, the algorithm proposed in this paper is a polynomial-time decomposition algorithm

EXAMPLE

Take the Petri net in Fig. 1 as an example to show the implementation process of the algorithm proposed in the article.

Step 1: Assignment the input and output set of each transition

$^\bullet t_1 = \{p_1, p_3\}$, $^\bullet t_2 = \{p_3, p_5\}$, $^\bullet t_3 = \{p_2\}$
$^\bullet t_4 = \{p_4\}$, $^\bullet t_5 = \{p_6, p_7\}$, $t_1^\bullet = \{p_2, p_6\}$ $t_2^\bullet = \{p_4\}$, $t_3^\bullet = \{p_3, p_6\}$, $t_4^\bullet = \{p_3, p_7\}$, $t_5^\bullet = \{p_1, p_5\}$.

Step 2: Put the number of tokens of each place to set M, so we get the set M = $\{p_1(1), p_2(0), p_3(1), p_4(0), p_5(1), p_6(0), p_7(0)\}$

Step 3: Put all places of Σ into X_1, then we get $x_1 = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$

Step 4: Assign mark to each place in X_1. Let s(k) represent that the mark of place s is k, so we can obtain $P_1 = \{p_1(2), p_2(1), p_3(4), p_4(0), p_5(2), p_6(3), p_7(2)\}$

Step 5: Decompose the net. Choose one place p_1 from X_1 whose mark is non-zero, and move it to X_2, so $X_2 = \{p_1\}$. Update the mark of each place in X_1, and store them in P_1, then $P_1 = \{p_2(1), p_3(3), p_4(0), p_5(1), p_6(3), p_7(2)\}$. Continue selecting places from X_1 and moving it to X_2 . Without loss of generalization, place p_2 whose mark is non-zero is chosen and moved to X_2 , so $X_2 = \{p_1, p_2\}$. Update the mark of each place in X_1, so $P_1=\{p_3(3), p_4(0), p_5(1), p_6(2), p_7(2)\}$. Select place p_3 whose mark is non-zero, and move it to X_2, so $X_2= \{p_1, p_2, p_3\}$. Update the mark of each place in X_1, $P_1 = \{p_4(0), p_5(0), p_6(1), p_7(1)\}$. Select place p_6 and move it to X_2 , so $X_2 = \{p_1, p_2, p_3, p_6\}$. Update the mark of each place inX_1 again, $P_1= \{p_4(0), p_5(0), p_7(0)\}$, so $X_1 = \{p_4, p_5, p_7\}$, and the selecting and moving operations on P_1 have been finished. Next, repeat the selecting and moving operations on X_2 such that $X_2 = \{p_1, p_2, p_3, p_6\}$. Obtain the mark of each place in X_2, and store them in P_2, so $P_2 = \{p_1(1), p_2(1), p_3(2), p_6(1)\}$. Select the place p_1 whose mark is non-zero, and move it to X_3, so $X_3 = \{p_1\}$. Update the mark of each place in X_2, $P_2 = \{p_2(1), p_3(1), p_6(1)\}$. Select place p_6 and move it to X_3 so as to obtain $X_3 = \{p_1, p_6\}$. Update the mark of each place in X_2, $P_2 = \{p_2(0), p_3(0)\}$, so $X_2 = \{p_2, p_3\}$, and the selecting and moving operations on P_2 have been finished. Next, repeat the selecting and moving operations on X_3 such that $X_3 = \{P_1, P_6\}$. To obtain the mark of each place in X_3 and store them in P_3, so $P_3 = \{P_1(0), P_6(0)\}$. The selecting and moving operations on P_3 have been finished, and the Step 5 is also finished

Step 6: To obtain connected subnets. $Y_1 = \{P_4, P_5, P_7\}$, $Y_2 = \{P_2, P_3\}$, and $Y_3 =\{P_1, P_6\}$

Step 7: Output the outface subnet. Firstly, we output the outface subnet of Y_3. For place P_1, if there is an edge which connects t_i and P_1, then there will be an edge connecting t_i and P_1. Repeat the processing on P_6, then we get the subnet of Σ_1. Using the same method, we can get the subnets of Σ_2 and Σ_3

Step 8: Output the tokens of each place in each subnet. Note that the number of tokens of each place of the original net has already been stored in the set M. p_1 has one token in Σ_1. In Σ_2, p_3 has one token and p_5 has one token in Σ_3

The output result of the whole algorithm is shown in Fig. 2.

CONCLUSION

In order to analyze properties of structure-complex Petri nets, the decomposition for a Petri net based on the indexes of places is a very convenient and useful approach. This study proposes an algorithm for the decomposition approach. The main data structures required and four key functions contained in the decomposition algorithm are addressed in details. It is proved that the proposed decomposition algorithm is a polynomial-time algorithm which provides methods for property analysis of structure-complex Petri nets.

ACKNOWLEDGMENTS

The research work presented in this study is supported by the National Science Foundation of China under Grant No. 60603090, 90718011 and 60673053, the Excellent Young Scientist Foundation of Shandong Province of China under Grant No. 2006BS01019 and the Taishan Scholar Program of Shandong Province.

REFERENCES

Esparza, J., 1994. synthesis of live and bounded free choice petri nets. Inform. Comput., 114: 50-87.

Lee, K.H. *et al.*, 1987. Generalized Petri net reduction method. IEEE Trans. Syst. Man Cybernetics, 17: 297-303.

Murata, T., 1989. Petri nets, properties, analysis and applications. Proc. IEEE, 77: 541-577.

Samir, M.K., 1999. Fast and simple decomposition techniques for the reliability analysis of interconnection networks. J. Syst. Software, 45: 155-171.

Wang, H.Q., C.J. Jiang and S.Y. Liao, 2000. Behavior relations in synthesis process of Petri net models. IEEE Trans. Robotics Automation, 16: 834-843.

Zeng, Q., 2004. Behavior descriptions of structure-complex Petri nets based on synchronous composition. J. Software, 15: 327-337.

Zeng, Q.T., 2007. Two symmetrical decomposition methods for structure-complex Petri net and their applications. 8th ACIS International Conference on Software Engineering, SNPD 2007. Artificial Intelligence, Networking and Parallel/Distributed Computing, pp: 1101-1106.

Zeng, Q.T. and H. Duan, 2007. Behavior description for complex flexible manufacturing system based on decomposition of Petri net. Int. J. Comput. Syst. Sci. Eng., 22: 359-363.