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 \( K = 0 \) 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 \( f : P \rightarrow \{1, 2, \ldots, k\} \) is said to be an index function defined on the place set if \( \forall p_1, p_2 \in P, (q_1 \cap q_2 \neq \emptyset) \Rightarrow f(p_1) \neq f(p_2) \). \( f(p) \) is named as the index of place \( p \).

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

\[ T_i = \{ t \in T | \exists p \in P, t \in \{ p \cup p^* \} \} \]  
(2)

\[ E_i = \{ (P_i \times T_i) \cup (T_i \times P_i) \} \cap F \]  
(3)

\[ M_a = \Gamma_{\pi_a} M_0 \]  
(4)

Simply, \( \Sigma_i \) is said as the index decomposition net of \( \Sigma \).

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

**Theorem 1**: (Zeng, 2007): Let \( \Sigma_i = (P_i, T_i, F_i, M_i) \) (i.e \{1, 2, ..., k\}) be the index decomposition net based on index of place of a Petri net \( \Sigma = (P, T, F, M_0) \), then \( \Sigma_i \) is an S-net. A structure-complex Petri net is shown in Fig. 1. We use the decomposition method of Definition 2 to decompose \( \Sigma \).

A function \( f \) is first defined on the place set such that \( f(p_1) = f(p_2) = 1, f(p_3) = f(p_4) = 2, f(p_5) = f(p_6) = f(p_7) = 3 \). It can prove that \( f \) satisfies all the conditions in Definition 1. Based on the method of Definition 2, three index decomposition net systems \( \Sigma_0, \Sigma_1 \) and \( \Sigma_2 \) are obtained and shown in Fig. 2. Obviously, \( \Sigma_2, \Sigma_1 \) and \( \Sigma_0 \) 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 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 \( A^+ = [a^+_{i,n}] \) matrix and input matrix \( A^- = [a^-_{i,n}] \) to represent the structure of a Petri net, where

\[ a^+_{i} = \begin{cases} 1 & \text{if } (i, s) \in F \\ 0 & \text{otherwise} \end{cases} \]

and

\[ a^-_{i} = \begin{cases} 1 & \text{if} (s, i) \in F \\ 0 & \text{otherwise} \end{cases} \]

\( A^+ = [a^+_{i,n}] \) and \( A^- = [a^-_{i,n}] \) 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 \( t_1 \), while bt represents the output set of \( t \), which are shown as followings. If a place \( s \) belongs to at (or bt), the \((i+1)\)th position in at (or bt) will be set as 1, otherwise be set as 0.

\[ \text{at} = \begin{array}{c} \ldots \ldots \ldots \end{array} \]

\[ \text{bt} = \begin{array}{c} \ldots \ldots \ldots \end{array} \]

**Store of a set \( X_s(k = 1, 2...) \)**: The places with same indexes will be put into a set \( X_s \) \( (k = 1, 2...) \) and \( X_s(k = 1, 2...) \) is stored by a one-dimension array with \(|S|\) length. If the \((i)\)th position in \( X_s \) is set as 1, it means \( s \) belongs to \( X_s(k = 1, 2...) \). 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 $s_k$ is $l$, the corresponding position of $s_k$ in $P_k$ 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 $s_k$ contains $l$ tokens, the corresponding position of $s_k$ in $Q$ will be set as $l$.

**Store of a set $Y_u (u = 1, 2...)$:** Another set $Y_u (u = 1, 2...)$ is 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 output subnet of the places with same index. At the initialization step of the algorithm, we put all the places into set $X_i$. For each place in $X_i$, denoted by $p$, let $\lambda[p] = \{ (p, q) \in T \mid p \neq q \}$ or $(p, q) \in T^*$. If there is a place moved out, the value $\lambda[p]$ for each place in $X_i$ will be updated. If $\lambda[p]$ for each place in $X_i$ is not 0, the selecting and moving operations will be repeated on $X_i$. Otherwise, repeat the selecting and moving operations on $X_{i+1}$. 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 $(X_b)$, Move $(X_b, y)$, Divide $(X_b)$ and Outface $(Y_c)$.

- **Mark $(X_b)$** //Obtain $\lambda[p]$ for each place in $X_b$
  - INPUT: $X_b$
  - OUTPUT: $\lambda[p]$ for each place in $X_b$
  - Step 0: for each $y \in X_b$, $\lambda[y] \leftarrow 0$. Let $i \leftarrow 1$.
  - Step 1: If $i > |S|$, halt.
  - Step 2: For each $t_o$ if there exists $x \in X_o$ and $x \neq y$ such that $y \in \text{t}_o$ and $x \in \text{t}_o$.
    - Do $\lambda[y] \leftarrow \lambda[y] + 1$
  - Step 3: If there exists $x \in X_o$ and $x \neq y$ such that $x \cap y \neq \emptyset$ and $x \cap y \neq \emptyset$.
    - Go to step 5
  - Step 4: For each $t_o$ if there exists $x \in X_o$ and $x \neq y$ such that $y \in \text{t}_o$ and $x \in \text{t}_o$.
    - Do $\lambda[y] \leftarrow \lambda[y] + 1$
  - Step 5: Let $i \leftarrow i + 1$, go to step 1.

- **Move $(X_b, y)$** //Move place $y$ from $X_b$ to $X_{b+1}$
  - INPUT: $X_b$ and place $y$
  - OUTPUT: Set $X_b$ and $X_{b+1}$
  - Step 1: Put $y$ into $X_{b+1}$.
  - Step 2: Search $y$ in $X_b$ and delete $y$ from $X_b$.

- **Divide $(X_b)$** //Divide $X_b$ into $Y_i, Y_{i+1}, Y_{i+2}$...
  - INPUT: $X_b$
  - OUTPUT: Sets $Y_i, Y_{i+1}, Y_{i+2}$...
  - While (there exist places in $X_b$) do
    - Move the first place into $Y_i$;
      - for each place $x \in Y_i$
        - for each place $y \in X_b$
          - for $i = 1$ to $|T|$ if $(x \in t_i$ and $y \in \text{t}_i$) or $(x \in t_i$ and $y \in \text{t}_i$), then move $y$ into $Y_i$
      - end for
      - end for
      - end for
      - $j = j + 1$;
    - }
Outface \((Y_j)\)  //Output the outface subnet of places in \(Y_t\)
INPUT: \(Y_t\)
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 \(\Sigma = (N, M, \cdot, \cdot, F, P, M_0)\)
OUTPUT: Decomposition subnets of \(\Sigma\)
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 \(S_j \in \cdot t_i\)
        if there is an edge from \(t_i\) to \(S_j\), then \(S_j \in t_i\cdot\)
      end for
    end for
  end for
Step 2: Store the markings of \(\Sigma\) in \(M\)
Step 3: Put all places of \(\Sigma\) into \(X_k\)
Step 4: Mark\((X_k)\)
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 \(\lambda[y]\) is not 0;
      Move\((X_k, y)\);
      Mark\((X_k)\) //update \(\lambda[y]\)
      If \(\lambda[y]\) of each place in \(X_k\) is 0, then break; //quit and execute next loop;
    end for
  end for
Step 6: //Divide \(X_1, X_2, ..., \) into \(Y_1, Y_2, ...\)
  for \(i =1\) to \(|k|\)
    Divide\((X_i)\)
  end for
Step 7. //Output each subnet of \(\Sigma\)
  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 \(s_j\) is in subnet \(N_i\), 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 \(O(n'm)\), where \(n = |S|\) and \(m = |T|\), and the same to the followings.
- In function Move\((X_k, y)\), the worst ease 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 \(O(n'2m)\).
- In function Outface \((Y_j)\), each “if” loop executes finite number of judgments and assignments, so its time complexity is \(O(n'm)\).

In the main algorithm, because the inner “for” loop executes finite number of judgments and assignments, so the time complexity of the first step is \(O(n'm)\). 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 \(O(n'm)\). In Step 5, the step “select \(y \in X_k\) such that \(\lambda[y]\) 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 \(O(n'2m)\). There are two outside layers of “for” loops, so the time complexity of Step 5 is \(O(n'2m)\). In Step 6, the inner function is Divide\((X_k)\), the time complexity of this step is \(O(n'2m)\). 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 \(O(n'2m)\). In Step 8, there are \(n\) times for search and assignments, so the time complexity of this step is \(O(n'2m)\).

According to the time complexity of each step in the main algorithm, the time complexity of the whole algorithm is \(O(nmn'n'2m'n'2m'n'2m'n'2m'n'2m'n'2m'n'2m'n'2m'n'2m'n'2m'n'2m'n'2m'n'2m'n'2m'n'2m'n'2m'n'2m'n'2m'n'2m'n'2m'n')\). 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

\[ t_1 = \{p_2, p_3\}, t_2 = \{p_3, p_4\}, t_3 = \{p_2\} \]

\[ t'_1 = \{p_4\}, t'_2 = \{p_5, p_6\}, t'_3 = \{p_2, p_3\}, t'_4 = \{p_5\}, t'_5 = \{p_6\} \]

**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 \( \Sigma \) into \( X_n \), then we get \( x_i = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\} \)

**Step 4:** Assign mark to each place in \( X_n \). 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 \( p_i \) from \( X_n \) whose mark is non-zero, and move it to \( X_n \), so \( X_n = \{p_i\} \). Update the mark of each place in \( X_n \), and store them in \( P_i \), then \( P_i = \{p_1(1), p_2(3), p_3(0), p_4(1), p_5(3), p_6(2), p_7(2)\} \). Continue selecting places from \( X_n \) and moving it to \( X_n \). Without loss of generalization, place \( p_i \) whose mark is non-zero is chosen and moved to \( X_n \), so \( X_n = \{p_i\} \). Update the mark of each place in \( X_n \), so \( P_i = \{p_1(3), p_2(0), p_3(1), p_4(2), p_5(2)\} \). Select place \( p_i \) whose mark is non-zero, and move it to \( X_n \), so \( X_n = \{p_i, p_j\} \). Update the mark of each place in \( X_n \), \( P_i = \{p_1(0), p_2(0), p_3(1), p_4(1)\} \). Select place \( p_i \) and move it to \( X_n \), so \( X_n = \{p_i, p_j, p_k\} \). Update the mark of each place in \( X_n \), \( P_i = \{p_1(0), p_2(0), p_3(1), p_4(1)\} \). Select place \( p_i \) whose mark is non-zero, and move it to \( X_n \), so \( X_n = \{p_i, p_j\} \). Update the mark of each place in \( X_n \), \( P_i = \{p_1(1), p_2(1), p_3(1)\} \). Select place \( p_i \) and move it to \( X_n \), so as to obtain \( X_n = \{p_i, p_j\} \). Update the mark of each place in \( X_n \), \( P_i = \{p_1(0), p_2(0)\} \), so \( X_n = \{p_i, p_j\} \), and the selecting and moving operations on \( P_i \) have been finished. Next, repeat the selecting and moving operations on \( X_n \) such that \( X_n = \{p_i, p_j, p_k\} \). Obtain the mark of each place in \( X_n \) and store them in \( P_i \), so \( P_i = \{p_1(1), p_2(2), p_3(1)\} \). Select place \( p_i \) whose mark is non-zero, and move it to \( X_n \), so \( X_n = \{p_i\} \). Update the mark of each place in \( X_n \), \( P_i = \{p_1(1), p_2(2), p_3(1)\} \). Select place \( p_i \) and move it to \( X_n \), so as to obtain \( X_n = \{p_i, p_j\} \). Update the mark of each place in \( X_n \), \( P_i = \{p_1(0), p_2(0)\} \), so \( X_n = \{p_i, p_j\} \), and the selecting and moving operations on \( P_i \) have been finished. Next, repeat the selecting and moving operations on \( X_n \) such that \( X_n = \{p_i, p_j, p_k\} \). To obtain the mark of each place in \( X_n \) and store them in \( P_i \), so \( P_i = \{p_1(0), p_2(0)\} \). The selecting and moving operations on \( P_i \) have been finished, and the Step 5 is also finished.

**Step 6:** To obtain connected subnets. \( Y_1 = \{P_n, P_{n+1}, P_{n+2}\} \), \( Y_2 = \{P_n, P_{n+1}\} \), and \( Y_3 = \{P_n, P_{n+1}\} \)

**Step 7:** Output the outface subnet. Firstly, we output the outface subnet of \( Y_1 \). For place \( P_n \), if there is an edge which connects \( t_i \) and \( P_n \), then there will be an edge connecting \( t_i \) and \( P_n \). Repeat the processing on \( P_n \) then we get the subnet of \( \Sigma_n \). Using the same method, we can get the subnets of \( \Sigma_n \) and \( \Sigma_n \).

**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 \( M \). \( p_i \) has one token in \( \Sigma_n \). In \( \Sigma_n \), \( p_i \) has one token and \( p_i \) has one token in \( \Sigma_n \).

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. 60603050, 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


