A Polynomial-time Decomposition Algorithm for Petri Nets
Based on Indexes of Transitions

Qingtian Zeng
College of Information Science and Engineering, Shandong University of Science and Technology,
Qingdao 266510, China

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

Key words: Petri net, index of transition, decomposition, polynomial-time algorithm

INTRODUCTION

As models for physical systems, Petri nets are well suited to describe and analyze systems with concurrency, synchronization and conflicts (Murata, 1989; Zeng, 2004; Zeng and Duan, 2007; Wang and Zeng, 2008). 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. Decomposition of Petri nets is one of very useful methods for property analysis of structure-complex systems. Kwang et al. (1987) gave several generalized reduction methods of Petri nets. In Koriem (1999), two analytical decomposition techniques are proposed for computing the transient state space solution of large Stochastic Petri Net (SPN) models of Multistage Interconnection Networks (MINs) and Hierarchical Interconnection Networks (HINs). A large scale SPN model is partitioned into smaller submodels. The submodels are compressed and combined to calculate the entire net. In Esparza (1994), many reduction rules are introduced 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 Zeng (2007), two decomposition methods are proposed for structure-complex Petri nets based on the indexes of places and transitions, respectively. The decomposition methods proposed in Zeng (2007) are very useful for property analysis of structure-complex Petri nets since the structure of the decomposition net is well-formed by Zeng (2007, 2008) and Cui et al. (2011). The language and process relations are analyzed during the decomposition and a method is proposed to present the process of a structure-complex Petri net (Zeng, 2008), respectively. However, we only presented the decomposition method for a Petri net based on the indexes of places or transitions in the previous research results (Zeng, 2007, 2008; Cui et al., 2011).

To find a decomposition algorithm especially a polynomial-time decomposition algorithm for a Petri net is also as important as the decomposition method. Unlike the traditional work on Petri net decomposition (Kwang et al., 1987; Koriem, 1999; Esparza, 1994; Zeng, 2007), this study addresses a polynomial-time algorithm for Petri net decomposition. Recently, we presented a polynomial-time algorithm was presented for the decomposition approach based on indexes of places (Zeng et al., 2008). This study gives a decomposition algorithm for a Petri net based on indexes of transitions and analyzes the complexity of the given algorithm. The main data structures and key functions contained in the algorithm will be addressed in detail. It is proved that the proposed decomposition algorithm is a polynomial-time algorithm. It provides a useful computation method for the decomposition of structure-complex Petri nets.

DECOMPOSITION OF A PETRI NETS BASED ON THE INDEXES OF TRANSITIONS

Some of the essential terminology and notations related to this study are defined first. To save space, more concepts of Petri nets can be seen (Murata, 1989; Zeng
Definition 1: Let $\Sigma = (S, T; F, M_0)$ be a Petri net, a function $f: T \rightarrow \{1, 2, \ldots, k\}$ is said to be an index function defined on the transition set if:

$$\forall t_i, t_j \in T, (t'_i \cap t'_j = \emptyset) \vee (t'_i \cap t'_j \neq \emptyset) \rightarrow f(t_i) \neq f(t_j)$$

$f(t)$ is named as the index of the transition $t$ (Zeng, 2007).

Definition 2: Let $\Sigma = (S, T; F, M_0)$ be a Petri net, $f: T \rightarrow \{1, 2, \ldots, k\}$ be the index function on the transitions of $\Sigma$. The Petri net $\Sigma_i = (S, T_i; F, M_0)$ ($i \in \{1, 2, \ldots, k\}$) is said to be the decomposition net of $\Sigma$ based on the index function $f$ if $\Sigma_i$ satisfies the following conditions (Zeng, 2007):

$$T_i = \{t \in T | f(t) = i\}$$

(1)

$$S_i = \{s \in S | \exists t \in T_i, s \in t \cup t'\}$$

(2)

$$F_i = \{(S \times T_i) \cup (T_i \times S)\} \cap F$$

(3)

$$M_{0i} = \Gamma_{S \rightarrow S_i} M_0$$

(4)

where, $\Gamma_{S \rightarrow S_i}$ is the projection of $M_0$ such that:

$$\forall s \in S_i \Gamma_{S \rightarrow S_i} M_0(s) = M_0(s)$$

Simply, $\Sigma_i$ is named as the index decomposition net of $\Sigma$.

Definition 3: A Petri net $\Sigma = (S, T; F, M_0)$ is a T-Net iff $\forall s \in S, |s| \leq 1$ and $|s| \leq 1$ (Zeng, 2007).

Theorem 1: Let $\Sigma = (S, T; F, M_0)$ ($i \in \{1, 2, \ldots, k\}$) be the decomposition net based on index of transition of a Petri net $\Sigma_i = (S, T; F, M_0)$, then $\Sigma_i$ is a T-net (Zeng, 2007).

Proof: In the decomposition nets $\Sigma_i = (S, T_i; F, M_0)$ ($i \in \{1, 2, \ldots, k\}$), we assume that there is at least one net is not a T-net. Without loss of generality, let $\Sigma_i = (S, T_i; F, M_0)$ (1 $\leq i \leq k$) be not a T-net. Because $\Sigma_i = (S, T_i; F, M_0)$ is not a T-net, there is at least one place $s \in S$ such that $|s| > 1$ or $|\bullet s| > 1$ according to the definition of T-net.

- In the case $|s| > 1$, it means that the place $s$ has at least two input transitions. Without loss of generality, we assume that the input transitions of $s$ are $t_i$ and $t_j$, thus:

$$t_i \cap t_j = \emptyset$$

- According to Definition 1, $f(t_i) \neq f(t_j)$. According to Definition 2, $t_i$ and $t_j$ should be decomposed into two subnet, so $t_i$ and $t_j$ can not be the input transitions of $s$ in $\Sigma_i = (S, T_i; F, M_0)$. Thus, the assumption is not correct.

- In the case $|\bullet s| > 1$, the conflict can also be obtained using the similar proof process. Therefore, in the decomposition nets ($\Sigma_i = (S, T_i; F, M_0)$ ($i \in \{1, 2, \ldots, k\}$) there is no any net is not a T-net. The theorem is proved.

The Petri net shown in Fig. 1 is the model for the problem of the dining philosophers. We use the decomposition method of Definition 2 to decompose $\Sigma$. 

---

**Fig. 1:** The Petri net model for the problem of the dining philosophers.
A function \( f \) is first defined on the place set such that:
\[
\begin{align*}
    f(t_1) &= f(t_2) = 1, \\
    f(t_3) &= f(t_4) = 2, \\
    f(t_5) &= f(t_6) = 3, \\
    f(t_7) &= f(t_8) = 4, \\
    f(t_9) &= f(t_{10}) = 5
\end{align*}
\]

It can be proven that \( f \) satisfies all the conditions in Definition 1. Based on the method of Definition 2, five indecomposable net systems \( \Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4, \) and \( \Sigma_5 \) are obtained. Each \( \Sigma_i (i = 1, 2, 3, 4, 5) \) is shown in Fig. 2. The semantic of the decomposition approach is clear. We can see that each \( \Sigma_i (i = 1, 2, 3, 4, 5) \) shown in Fig. 2 is the model for one philosopher.

More discussion about the decomposition approach based on the indexes of transitions can be seen by Zeng (2007). Based on the results presented by Zeng (2007), to find a polynomial-time algorithm for the decomposition approach is the future work to be addressed. In the following sections, we present a polynomial-time algorithm for the decomposition approach. Firstly, the main data structures and key functions contained in the algorithm are addressed first.

**MAIN DATA STRUCTURES**

Firstly, the main data structures to store the components 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 (Murata, 1989), we use output matrix:
\[
A^* = [a_{ij}]_{m \times n}
\]
and input matrix:
\[
A^- = [a_{ij}]_{m \times n}
\]
to represent the structure of a Petri net, where,
\[
a^*_{ij} = \begin{cases} 
1 & \text{if} \ (t_i, s_j) \in F \\
0 & \text{otherwise}
\end{cases} \quad \text{and} \quad a^-_{ij} = \begin{cases} 
1 & \text{if} \ (s_i, t_j) \in F \\
0 & \text{otherwise}
\end{cases}
\]
can be stored by a two-dimension array, respectively.

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

![Array as](image)

![Array bs](image)

**Store of a set \( X_k (k = 1, 2\ldots) \):** The transitions with same indexes will be put into a set \( X_k (k = 1, 2\ldots) \) and \( X_i \) \((k = 1, 2\ldots)\) is stored by a one-dimension array with \(|T|\) length. If the \((i)\)th position in \( X_k \) is set as 1, it means \( t_i \) belongs to \( X_k (k = 1, 2\ldots) \). Otherwise, the corresponding position will be set as 0.

![Array X](image)

**Store of index of each transition:** The index of each transition is stored by a one-dimension array with \(|T|\) length, \( P_k (k = 1, 2\ldots) \). If the index of \( t_i \) is 1, the corresponding position of \( t_i \) in \( Q \) will be set as \( l \) in \( P_k (k = 1, 2\ldots) \).
Store of tokens: A one-dimension array with |S| length, Q will be used to store all the tokens in the places. If s<sub>i</sub> contains 1 tokens, the corresponding position of s<sub>i</sub> in Q will be set as 1.

Store of a set Y<sub>u</sub> (u = 1, 2, ...): Another set Y<sub>u</sub> (u = 1, 2, ...) are used to store the transitions of each decomposition net Y<sub>u</sub> (u = 1, 2, ...) is also represented by a one-dimension array with |T| length. If the (i)th position in Y<sub>u</sub> is set as 1, it means t<sub>i</sub> belongs to Y<sub>u</sub> (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 transition. According to the index of each transition, the decomposition net can be obtained transaction with same index. At the initialization step of the algorithm, we put all the transition s into set X<sub>i</sub>. For each transition in X<sub>i</sub>, denoted by t, let:

\[
\lambda [t]=\{t, |P(t, t)\subseteq* p or (t, t)\subseteq* p\}
\]

In the following step, we select a transition t in X<sub>i</sub> (i = 1, 2, ...) such that \(\lambda [t] \neq 0\) and move t from X<sub>i</sub> to X<sub>s</sub>. If there is a transition moved out, the value \(\lambda [t]\) for each transition in X<sub>s</sub> will update with the same index. At the initialization step of the be updated. If \(\lambda [t]\) for each transition in X<sub>i</sub> is not equal to 0, the selecting and moving operations will be repeated on X<sub>i</sub>. Otherwise, repeat the selecting and moving operations on X<sub>i</sub>. After the selecting and moving operations on completed on all sets, the transitions 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<sub>n</sub>), Move (X<sub>n</sub>, y), Divide (X<sub>n</sub>) and Outface (Y<sub>u</sub>).

- **Mark(X<sub>n</sub>):** //Obtain \(\lambda [p]\) for each place in X<sub>n</sub>
  - INPUT: X<sub>n</sub>
  - OUTPUT: \(\lambda [p]\) for each transition in X<sub>n</sub>

  ```
  \{
  Step 0: for each p\in X_n, \lambda [p]=0. Let i=1.
  Step 1: If i\in [S], halt.
  Step 2: For each s<sub>i</sub>, if there exists \(s<sub>i</sub>, x\in s<sub>i</sub>\) and \(x\in s<sub>i</sub>\), DO \(\lambda [y]=\lambda [y]+1\)
  Step 3: If there exists \(i<sub>i</sub>, x\in s<sub>i</sub>\) and \(x\in s<sub>i</sub>\) such that:
  \(x\in \cdot y\in \mathbb{S}\), and \(x\in \cdot y\in \mathbb{S}\)
  go to Step 5
  Step 4: For each s<sub>i</sub>, if there exists \(s<sub>i</sub>, x\in s<sub>i</sub>\) and \(x\in s<sub>i</sub>\), DO \(\lambda [y]=\lambda [y]+1\)
  Step 5: let i = i + 1, goto Step 1.
  ```

- **Move(X<sub>n</sub>, y):** //Move transition y from X<sub>n</sub> to X<sub>n+1</sub>
  - INPUT: X<sub>n</sub> and transition y
  - OUTPUT: set X<sub>n</sub> and X<sub>n+1</sub>

  ```
  Put y into X<sub>n+1</sub>.
  Search y in X<sub>n</sub> and delete y from X<sub>n</sub>.
  ```

- **Divide (X<sub>n</sub>):** //Divide X<sub>n</sub> into, Y<sub>1</sub>, Y<sub>2</sub>, Y<sub>3</sub>,...
  - INPUT: X<sub>n</sub>
  - OUTPUT: sets Y<sub>1</sub>, Y<sub>2</sub>, Y<sub>3</sub>,...

  ```
  While (there exists transitions in X<sub>n</sub>)
  Do
  Move the first transition into Y<sub>j</sub>;
  for each transition \(s<sub>i</sub>, x\in s<sub>i</sub>\) for each transition \(y\in X<sub>i</sub>\) for i = 1 to |S|
  if \((x\in s<sub>i</sub>, y\in s<sub>i</sub>) or (x\in s<sub>i</sub>, y\in s<sub>i</sub>)\), then move y into y<sub>j</sub>
  end for
  end for
  end for
  j = j + 1;
  ```

- **Outface (Y<sub>j</sub>):** //Output the outface subnet of transitions in Y<sub>j</sub>
  - INPUT: Y<sub>j</sub>
  - OUTPUT: outface subnet N<sub>j</sub>

  ```
  (for each transition \(s<sub>i</sub>, y\in X<sub>i</sub>\)
  for i = 1 to |S|
  if there is an edge connecting s, then y
  then s, and y are connected
  end for
  ```

A polynomial-time decomposition algorithm: Now we present a polynomial-time decomposition algorithm for Petri nets based on indexes of transitions.
INPUT: a Petri net \( \Sigma = (N, M) = (S, T; F, M_0) \)
OUTPUT: Decomposition subnets of \( \Sigma \)

Step 1: // To obtain the presets and postset of each place
for \( i = 1 \) to \( |S| \)
for \( j = 1 \) to \( |T| \)
if there is an edge from \( t \) to \( s \), then \( s \in \text{pre}(t) \)
if there is an edge from \( s \) to \( t \), then \( t \in \text{post}(s) \)
end for
end for

Step 2: Store the markings of \( \Sigma \) in \( M \)

Step 3: Put all transitions of \( \Sigma \) into \( X_t \)

Step 4: Mark \( X_o \)

Step 5: // obtain the index of each transition for \( k = 1 \) to \( |T| \)
for \( j = 1 \) to \( |T| \)
select \( x \in X_t \), such that \( \lambda_{[y]} \) is not 0
Move \( X_o, y \)
Mark \( X_o / \text{update}[y] \)
If \( \lambda_{[y]} \) of each transition in \( X_o \) is 0,
then break // quit and execute next loop
end for
Mark \( X_o / \text{update}[y] \)
end for

Step 6: // Divide \( X_o, X_p, \ldots \) into \( Y_1, Y_2, \ldots \)
for \( i = 1 \) to \( k \)
Divide \( X_o \)
end for

Step 7: // Output each subnet of \( \Sigma \)
for \( i = 1 \) to \( |T| \)
Output \( Y_i \)
end for

Step 8: // Output markings of each subnet
for \( i = 1 \) to \( |T| \)
for \( j = 1 \) to \( |S| \)
if \( s \) is in subnet \( N_i \), then add \( M[k] \) tokens to \( s \)
end for
end for

COMPLEXITY ANALYSIS OF THE ALGORITHM

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

- **Step 1**: In function Mark \( X_o \), 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 \( O(m'n) \), where \( n = |S| \) and \( m = |T| \) and the same to the followings.

- **Step 2**: In function Move \( X_o, y \), the worst case of step Search \( y \) in \( X_o \) and delete \( y \) from \( X_o \) is to go through the whole set \( X_o \), so the time complexity of this function is only \( O(m) \).

- **Step 3**: In function Divide \( X_o \), from the number of layers of the for loops, it can be determined that the time complexity of this function is also \( O(m'n) \).

- **Step 4**: In function Outface \( Y_i \), each if loop executes finite number of judgments and assignments, so its time complexity is \( O(m) \).

In the main algorithm, because the inner for loop executes finite number of judgments and assignments, the time complexity of the first step is \( O(mn) \). In step 2, there are \( n \) times for assignments, so the time complexity of Step 2 is \( O(n) \). In step 3, the time complexity is \( O(mn) \). The time complexity of step 4 is actually same to that of the function Mark \( X_o \), so it is \( O(m'n) \). In step 5, the step select \( y \) of \( X_o \) 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 \( O(m) \). The if loop is actually to go through the whole set \( X_o \), so the time complexity of the inner for loop is \( O(m'n) \). There are two outside layers of “for” loops, so the time complexity of step 5 is \( O(m'n) \). In step 6, the inner function is Divide \( X_o \) and the time complexity of this step is \( O(m'n) \). The time complexity of step 7 is mainly determined by the “for” loop and the function Outface \( Y_i \), so the time complexity of this step is \( O(m'n) \). In step 8, there are \( n \) times for search and assignments, so the time complexity of this step is \( O(m'n) \).

According to the time complexity of each step in the main algorithm, the time complexity of the whole algorithm is \( O((m+n+m+n+m'n+m'n+m'n+m'n) = O(m'n) \). Therefore, the algorithm proposed in this study is a polynomial-time decomposition algorithm.

EXAMPLE

We take the Petri net for the problem of the dining philosophers in Fig. 1 as an example to show the implementation process of the algorithm proposed in the study.

**Step 1**: Assignment the input and output set of each place.

\[
\begin{align*}
\lambda_1 &= \{t_{11}, t_{12}, t_{13}\}, \lambda_2 &= \{t_{21}, t_{22}, t_{23}\}, \lambda_3 &= \{t_{31}, t_{32}, t_{33}\}, \\
\lambda_4 &= \{t_{41}, t_{42}, t_{43}\}, \lambda_5 &= \{t_{51}, t_{52}, t_{53}\}, \\
\lambda_6 &= \{t_{61}, t_{62}, t_{63}\}, \lambda_7 &= \{t_{71}, t_{72}, t_{73}\}, \\
\lambda_8 &= \{t_{81}, t_{82}, t_{83}\}, \lambda_9 &= \{t_{91}, t_{92}, t_{93}\}, \\
\lambda_{10} &= \{t_{101}, t_{102}, t_{103}\}, \\
\lambda_{11} &= \{t_{111}, t_{112}, t_{113}\}, \\
\lambda_{12} &= \{t_{121}, t_{122}, t_{123}\}, \\
\lambda_{13} &= \{t_{131}, t_{132}, t_{133}\}, \\
\lambda_{14} &= \{t_{141}, t_{142}, t_{143}\}, \\
\lambda_{15} &= \{t_{151}, t_{152}, t_{153}\}, \\
\lambda_{16} &= \{t_{161}, t_{162}, t_{163}\}, \\
\lambda_{17} &= \{t_{171}, t_{172}, t_{173}\}, \\
\lambda_{18} &= \{t_{181}, t_{182}, t_{183}\}, \\
\lambda_{19} &= \{t_{191}, t_{192}, t_{193}\}, \\
\lambda_{20} &= \{t_{201}, t_{202}, t_{203}\}, \\
\lambda_{21} &= \{t_{211}, t_{212}, t_{213}\}, \\
\lambda_{22} &= \{t_{221}, t_{222}, t_{223}\}, \\
\lambda_{23} &= \{t_{231}, t_{232}, t_{233}\}, \\
\lambda_{24} &= \{t_{241}, t_{242}, t_{243}\}, \\
\lambda_{25} &= \{t_{251}, t_{252}, t_{253}\}, \\
\lambda_{26} &= \{t_{261}, t_{262}, t_{263}\}, \\
\lambda_{27} &= \{t_{271}, t_{272}, t_{273}\}, \\
\lambda_{28} &= \{t_{281}, t_{282}, t_{283}\}, \\
\lambda_{29} &= \{t_{291}, t_{292}, t_{293}\}, \\
\lambda_{30} &= \{t_{301}, t_{302}, t_{303}\}.
\end{align*}
\]

**Step 2**: Put the number of tokens of each place to set \( M \), so we get the set \( M = \{p_{11} (1), p_{12} (1), p_{13} (0), p_{21} (1), p_{22} (1), p_{23} (0), p_{24} (1), p_{25} (1), p_{26} (0), p_{27} (1), p_{28} (1), p_{29} (0), p_{30} (1), p_{31} (1), p_{33} (0), p_{35} (0)\} \).

**Step 3**: Put all transitions of \( \Sigma \) into \( X_o \), then we get \( X_o = \{t_{11}, t_{12}, t_{13}, t_{14}, t_{15}, t_{16}, t_{17}, t_{18}, t_{19}, t_{20}\} \).
Step 4: Assign mark to each place in $X_i$. Let $t(k)$ represent that the mark of transition $t$ is $k$, so we can obtain $P_0 = \{t_{11}(2), t_{12}(2), t_{21}(2), t_{22}(2), t_{31}(2), t_{32}(2), t_{41}(2), t_{42}(2), t_{51}(2), t_{52}(2)\}$.

Step 5: Decompose the net. Choose one transition $t_{ij}$ from $X_i$, whose mark is non-zero and move it to $X_s$, so $X_s = \{t_{ij}\}$. Update the mark of each transition in $X_i$ and store them in $P_i$, then, $P_i = \{t_{11}(1), t_{12}(2), t_{21}(2), t_{22}(1), t_{24}(2), t_{31}(1), t_{32}(2), t_{33}(2), t_{42}(2), t_{51}(2)\}$. Continue selecting transitions from $X_i$ and moving it to $X_s$. Without loss of generalization, transition $t_{ij}$ whose mark is non-zero is chosen and moved to $X_s$, so $X_s = \{t_{ij}\}$. Update the mark of each transition in $X_i$, so $P_i = \{t_{11}(1), t_{12}(1), t_{31}(1), t_{41}(1), t_{42}(2), t_{51}(2), t_{52}(2)\}$. Select transition $t_{ij}$ whose mark is non-zero and move it to $X_s$, so $X_s = \{t_{ij}\}$. Update the mark of each transition in $X_i$, again, $P_i = \{t_{11}(1), t_{12}(1), t_{31}(1), t_{32}(2), t_{41}(1), t_{42}(2), t_{51}(1), t_{52}(2)\}$. Select transition $t_{ij}$ and move it to $X_s$, and $X_i = \{t_{ij}, t_{ij}\}$. Update the mark of each transition in $X_i$, $P_i = \{t_{11}(1), t_{12}(1), t_{31}(1), t_{32}(2), t_{41}(1), t_{51}(1), t_{52}(2)\}$. Select transition $t_{ij}$ and move it to $X_s$, and $X_i = \{t_{ij}, t_{ij}\}$. Update the mark of each transition in $X_s$, $P_s = \{t_{11}(0), t_{12}(1), t_{31}(1)\}$. Select transition $t_{ij}$ and move it to $X_s$, and $X_i = \{t_{ij}, t_{ij}\}$. Update the mark of each transition in $X_i$, $P_i = \{t_{11}(0), t_{12}(1), t_{12}(1)\}$ and the selecting and moving operations on $P_i$ have been finished. Next, repeat the selecting and moving operations on $X_s$. At last, we can get $X_s = \{t_{11}, t_{12}\}$; $X_i = \{t_{11}, t_{12}\}$; $X_i = \{t_{11}, t_{12}\}$; $X_i = \{t_{11}, t_{12}\}$; $X_i = \{t_{11}, t_{12}\}$; $X_i = \{t_{11}, t_{12}\}$.

Step 6: To obtain connected subnets. $Y_i = \{t_{11}, t_{12}\}$; $Y_{ij} = \{t_{12}, t_{22}\}; Y_1 = \{t_{11}, t_{12}\}$; $Y_4 = \{t_{11}, t_{12}\}$; $X_i = \{t_{11}, t_{12}\}$.

Step 7: Output the outface subnet. Firstly, we output the outface subnet of $Y_i$. For transition $t_{ij}$, if there is an edge which connects $t_{ij}$ and $s_k$, then there will be an edge connecting $t_{ij}$ and $s_k$. Repeat the processing on $t_{ij}$, then we get the subnet of $\Sigma_i$. Using the same method, we can get the subnets of $\Sigma_i$, $\Sigma_i$, $\Sigma_i$, $\Sigma_i$, and $\Sigma_i$. 

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 $M$.

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