Information Technology Journal 10 (1): 106-112, 2011
ISSN 1812-5638
© 2011 Asian Network for Scientific Information

Recognition Algorithm Design and Complex Analysis for Languages of S-Nets

Tingting Cui, Qingtian Zeng and Dongli Zhang
College of Information Science and Engineering,
Shandong University of Science and Technology, 266510, People's Republic of China

Abstract: S-Net is a kind of structure-simple Petri nets and its behaviors are easy to be specified. In this study, we observed the classification of S-Nets and then gave the recognition method of every kind of S-Nets. A finite automaton is constructed first that can be used to recognize the language of a bounded Petri net. Based on the finite automaton constructed, the language recognition algorithm for each kind of S-Nets is proposed and the time complexity of the proposed algorithm is discussed. By several groups of experimental data, the time complexities of the proposed algorithms are checked.

Key words: Petri net, petri net language, language recognition

INTRODUCTION

As models for modeling and analyzing distributed systems, Petri nets are used in a wide range of domains. The transition firing sequences can specify the behaviors of the Petri nets and Petri nets languages are recognized as a set of transition firing sequences. Problems of Petri nets languages are well-studied in recent years and some relatively perfect solutions have been given by many researchers. However, most of them regard Petri nets as language generators (Zeng and Wu, 2002, 2004; Zeng *et al*., 2004; Jiang and Lu, 2001; Jiang *et al*., 1997). By now, there is only a few people (Jiang *et al*., 1998) recognize the possibility of utilization of Petri nets as languages identifiers. Jiang *et al*. (1998) gave several recognition algorithms for Petri nets languages, including a serial algorithm and two parallel algorithms, but these algorithms are extended for ordinary Petri nets and are too complicated. S-Net is a kind of simple-structure Petri nets with good structure and behavior property. Zeng and Wu (2002) gave languages properties and description methods for S-Nets. In this study, we study the recognition problems for languages of S-Nets, how to solve them and also their time complexity. At last, by several groups of experimental data, the time complexities of these language recognition algorithms are verified. The solution in this study will be of great benefit to the further study of language recognition problems for structure-complex Petri nets.

BASIC CONCEPTS AND NOTATIONS OF S-NETS

Definition 1 (Wu, 2006): A triple N = (S T; F) is named as a net iff

(1) $S \cup T \neq \varnothing$
(2) $S \cap T = \varnothing$
(3) $F \subseteq ((S \times T) \cup (T \times S))$
(4) dom (F) ∪ cod (F) = $S \cup T$

where, dom (F) = $\{x \in S \cup T \mid \exists y \in S \cup T: (x, y) \in F\}$ and cod (F) = $\{x \in S \cup T \mid \exists y \in S \cup T: (y, x) \in F\}$

Definition 2 (Wu, 2006): Let N = (S, T; F) be a net. For any $x \in S \cup T$,
$^{\bullet}x = \{y \mid y \in S \cup T \wedge (y, x) \in F\}$ is named as the preset or input set of x and
$x^{\bullet} = \{y \mid y \in S \cup T \wedge (x, y) \in F\}$ is named as the postset or output set of x. $^{\bullet}x \cup x^{\bullet}$ is named as the extension of x.

Definition 3 (Wu, 2006): Let N = (S, T; F) be a net. The mapping M: $S \rightarrow \{0\ 1, 2,..\}$ is said to be the marking of a net N and (N, M) (which is the same as (S, T; F, M)) is named as a marking net.

Definition 4 (Zeng and Wu, 2002): Let N = (S, T; F) be a net.

(1) N is an S-graph if for any $t \in T$: $|^{\bullet}t| = |t^{\bullet}| = 1$. If N = (S, T; F) is an S-graph, Σ (N, M_0) is named as a marking S-graph
(2) N is an S-net if $\forall t \in T$, $|^{\bullet}t| \leq 1$ and $|t^{\bullet}| \leq 1$. If N = (S, T; F) is an S-net, Σ (N, M_0) is named as a marking S-net

According to Definition 4, an S-graph is obviously an S-net.

Definition 5 (Zeng and Wu, 2002): Let N = (S, T; F) be an S-net. For any $t \in T$, t is called a primitive transition of N if $^{\bullet}t = \varnothing$ and t is called a terminal transition if $t^{\bullet} = \varnothing$.

Corresponding Author: Qingtian Zeng, College of Information Science and Engineering,
Shandong University of Science and Technology, 266510, People's Republic of China

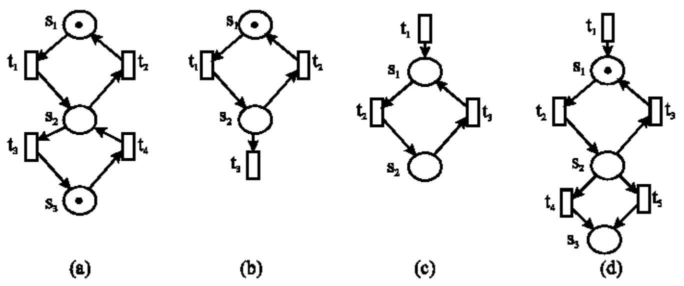

Fig. 1: (a-d) Four kinds of S-nets

The S-nets can be classified into five classes according to the primitive and terminal transitions, which are as follows.

(1) An S-net without primitive transitions or terminal transitions (i.e., an S-graph)
(2) A marking S-net without primitive transitions or terminal transitions (i.e., a marking S-graph)
(3) A marking S-net with terminal transitions but without primitive transitions
(4) An S-net with primitive transitions only
(5) A marking S-net with primitive transitions only

An S-graph is without markings, so it is impossible to fire any transition in an S-graph. In this study, we don't consider the language recognition of an S-graph. Figure 1 gives four S-nets, where (a) is a marking S-graph, (b) is a marking S-net with terminal transitions but without primitive transitions, (c) is an S-net with primitive transition only and (d) is a marking S-net with primitive transition only.

LANGUAGE RECOGNITION OF S-NETS

Definition 6: (Zeng and Wu, 2002): $L(\Sigma) = \{\sigma | \sigma \in T^* \wedge M_0 [\sigma > M_e \wedge (\forall s \in S\text{-}S_f), M_e(s) = 0\}$ is defined as the language of a Petri net $\Sigma = (S, T; F, M_0)$, where, $S_f \subseteq S$ is named as the end place set of Σ.

The common language operations include + (Selection) operation, • (Connection) operation and * (Kleen Closure) operation (Garg and Ragunath, 1992). || (Parallel) and α-Closure operations are also introduced by Garg and Ragunath (1992) and Zeng (2004).

Definition 7 (Zeng, 2004): The parallel operation (||) on the alphabet Ω is formally defined as follows:

(1) For all $a \in \Omega$, $a\|\varepsilon = \varepsilon\|a = \{a\}$, where, ε is the empty word of Ω.
(2) For all $a, b \in \Omega$, and all $\sigma_1, \sigma_2 \in \Omega^*$, $a\bullet\sigma_1\|b\bullet\sigma_2 = a\bullet(\sigma_1\|b\bullet\sigma_2)\cup b\bullet(a\bullet\sigma_1\|\sigma_2)$

Let $L(\Sigma_i)$ be the language of a Petri net $\Sigma_i = (P_i, T_i; F_i, M_{0i}, S_{fi})$ $(i \in \{1, 2\})$. The operation || of language $L(\Sigma_1)$ and $L(\Sigma_2)$ is defined as $L(\Sigma_1)\|L(\Sigma_2) = \{\sigma_1\|\sigma_2 | \sigma_1 \in L(\Sigma_1), \delta_2 \in L(\Sigma_2)\}$

Definition 8 (Garg and Ragunath, 1992): The α-closure of a language L is defined as:

$$L^{\alpha} = \bigcup_{i=0,1,\cdots} L^{(i)}$$

where, $L^i = L\|L^{(i-1)}$ $(I \geq 2)$
Example: $L = \{a\bullet b\bullet c\}$, then $L^{\alpha} = \{w | \forall s \in \text{Pref}(w)).\#(a, s) \geq \#(b, s) \geq \#(c, s) \wedge \#(a, w) \wedge \#(b, w) \wedge \#(c, w)\}$, where, Pr ef (w) is the prefix of w and # (a, w) is the number of a occurring in w.

Theorem 1: If $\Sigma = (S, T; F, M_0, S_f)$ is a marking S-graph, $L(\Sigma)$ can be recognized by a finite automaton.

Proof: Construct a finite automaton $FSM = (\Gamma, Q, \delta, q_0, q_f)$, as follows:

(1) $\Gamma \leftarrow T$
(2) $Q \leftarrow R(M_0)$
(3) $\delta \leftarrow \{(M_i, t, M_j) | M_i, M_j \in R(M_0) \wedge (M_i [t > M_j)\}$
(4) $q_0 \leftarrow M_0$
(5) $q_f \leftarrow M_e$ $(\forall s \in S\text{-}S_f, M_e(s) = 0)$

In order to prove that $L(\Sigma)$ can be recognized by $FSM = (\Gamma, Q, \delta, q_0, q_f)$, we should prove $L(\Sigma) = L(FSM)$, where L (FSM) is the language recognized by FSM.

First, we prove $L(\Sigma) \subseteq L(FSM)$. For any σ that can be recognized by the marking S-graph $\Sigma = (S, T; F, M_0, S_f)$, it means $\sigma \in L(\Sigma)$. Thus, there should exist a transition t connecting with the place with tokens in it and this transition can be fired for finite times (suppose the number is n). After firing, the token will move into the output places of t for finite times and finally reaches the set of the end places S_f. Let, $\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_n$ trace the firing sequence 1, 2,..., n and $\sigma \leftarrow \sigma_1\bullet\sigma_2\bullet\sigma_3\bullet\ldots\bullet\sigma_n$. It is easy to prove that σ_j $(j \in \{1, 2, \ldots n\})$ can be recognized by FSM, so $\sigma \in L(FSM)$.

Then we prove L (FSM)$\subseteq$L(Σ). $\forall\sigma \in$ L (FSM), then $\sigma\leftarrow \sigma_1\bullet\sigma_2\bullet\sigma_3\bullet\ldots\bullet\sigma_n$ and $\forall\sigma_j \in$ L (FSM) (j $\in$ {1, 2,..n}). Tracing every σ_j in Σ, we find σ_j is a transition firing sequence fired by a transition t connecting with the place with token in it. So, the token will move into the output places of t and finally reach the set of the end places S_f. Therefore, $\sigma_j \in$ L (Σ), so $\sigma\leftarrow \sigma_1\bullet\sigma_2\bullet\sigma_3\bullet\ldots\bullet\sigma_n$ describe the behavior of a transition t connecting with a place with token in it fired then run into the output places and finally run into the set of end places S_f, so $\sigma \in$ L (Σ).

Theorem 2: If Σ (S, T; F,M_0, S_f) is a marking S-net with terminal transitions but without primitive transitions, L (Σ) can be recognized by a finite automaton.

Proof: For any S-net Σ = (S, T; F, M_0, S_f) with some terminal transitions, we can construct another S-net Σ = (S', T; F', M_0', S_f') without any terminal transitions, such that L (Σ) = L (Σ'). Σ' = (S', T; F', M_0', S_f') can be constructed as follows: S' $\leftarrow$ S$\cup${s_{out}}, F'$\leftarrow$F$\cup$ {(t, s_{out})|t $\in$ T, $t^\bullet = \varnothing$},

$$\forall s\in S^{'}: M_0^{'}(s) \leftarrow \begin{cases} M_0(s) & s \neq s_{out} \\ 0 & s = s_{out} \end{cases}$$

$S'_f$$\leftarrow$S$\cup$ {s_{out}} We can prove that L (Σ) = L (Σ').

We construct a finite automaton FSM (Γ, Q, δ, q_0, q_f), as follows:

(1) $\Gamma \leftarrow$ T
(2) Q $\leftarrow$ R(M'_0)
(3) $\delta \leftarrow$ {(M'_i, t, M'_j)|M_i, $M_j \in$ R (M'_0)$\wedge$ (M'_i [t>M'_j)}
(4) $q_0 \leftarrow M'_0$
(5) $q_f \leftarrow M'_e$ ($\forall$s $\in$ S'-S'_f, M'_e (s) = 0)

The proof that L (Σ) can be recognized by FSM is similar to that of Theorem 1, so the proof details are ignored here.

Theorem 3: If Σ = (S, T; F, M_0, S_f) is an S-net with primitive transitions only, L (Σ) can be recognized by the parallel of several same finite automatons.

Proof: For any S-net Σ = (S, T; F, M_0, S_f) with primitive transitions, if $\forall$s $\in$ S, M_0 (s) = 0, we can construct another S-net Σ' = (S', T; F', M'_0, S_f), such that S' $\leftarrow$ S$\cup${s_{in}}, F'$\leftarrow$F$\cup$ {(s_{in}, t)|t $\in$ T, $^\bullet$t = $\varnothing$},

$$\forall s\in S^{'}: M_0^{'}(s) \leftarrow \begin{cases} M_0(s) & s \neq s_{in} \\ 1 & s = s_{in} \end{cases}$$

According to Theorem 3 (Zeng and Wu, 2002), we know L (Σ) is an α-closure of a regular expression. We can prove that L (Σ) = L $(\Sigma')^\alpha$.

We construct a finite automaton FSM = (Γ, Q, δ, q_0, q_f), as follows:

- $\Gamma \leftarrow$ T
- Q $\leftarrow$ R(M'_0)
- $\delta \leftarrow$ {(M'_i, t, M'_j)|M_i, $M_j \in$ R (M'_0) $\wedge$ (M'_i [t>M'_j)}
- $q_0 \leftarrow M'_0$
- $q_f \leftarrow M'_e$ ($\forall$s $\in$ S'-S_f, M'_e (s) = 0)

and we can prove L (Σ') = L (FSM).

In order to prove L (Σ) can be recognized by the parallel of an FSM, we should prove L (Σ) = L $(\Sigma')^\alpha$ = L $(FSM)^\alpha$, where L (FSM) is the language recognized by FSM.

First, we prove that L (Σ) $\subseteq$ L $(FSM)^\alpha$. For any σ that can be recognized by the S-net Σ (S, T; F, M_0, S_f), it means $\sigma \in$ L (Σ). Because $\forall$s $\in$ S, M_0 (s) = 0, there must be a primitive transition t can be fired for finite times (suppose the number is n). So after firing, a finite number of token of Σ move into the output places of t and finally reach the set of end places S_f. Let, σ_1, σ_2, σ_3,..., σ_n trace the transition firing sequence of token 1, 2,..., n, then $\sigma\leftarrow\sigma_1\|\sigma_2\|\sigma_3\|\ldots\|\sigma_n$. It is obvious that each σ_j (j $\in$ {1, 2,... n}) can be recognized by FSM. Therefore, $\sigma \in$ L $(FSM)^\alpha$.

Then, we prove that L $(FSM)^\alpha$ $\subseteq$L (Σ). $\forall\sigma \in$ L $(FSM)^\alpha$, then $\sigma\leftarrow\sigma_1\|\sigma_2\|\sigma_3\|\ldots\|\sigma_n$ and $\forall\sigma_j \in$ L (FSM) (j$\in$ {1, 2,... n}), for all σ_j, σ_j is a transition firing sequence that can be traced by firing one primitive transition t in Σ and the tokens produced by transition t move to the output places and finally reach the set of the end places S_f. So, $\sigma_j \in$ L(Σ). Therefore, $\sigma\leftarrow\sigma_1\|\sigma_2\|\sigma_3\|\ldots\|\sigma_n$ traces the behaviors of firing primitive transition and the tokens produced move into the terminal place set with n times. So $\sigma \in$ L (Σ).

Theorem 4: Σ = (S, T; F, M_0, S_f) is a marking S-net with primitive transition only, L (Σ) can be recognized by the parallel of several finite automatons.

Proof: For any marking S-net Σ = (S, T; F, M_0, S_f) with a primitive transition, we can construct another two S-nets Σ_1 = (S_a, T; F_a, M_{01}, S_f) and Σ_2 = (S_a, T; F_a, M_{02}, S_f) without any primitive transition, such that L (Σ) = L (Σ_1)$\|$L $(\Sigma_2)^\alpha$. Σ_1 = (S_a, T; F_a, M_{01}, S_f), Σ_2 = (S_a, T; F_a, M_{02}, S_f) can be constructed as follows: $S_a\leftarrow$ S$\cup${s_{in}}, $F_a$$\leftarrow$F$\cup$ {(t, s_{in})|t $\in$ T, $^\bullet$t = $\varnothing$}.

$$\forall s\in S_a: M_{01}(s) \leftarrow \begin{cases} M_0(s) & s \neq s_{in} \\ 0 & s = s_{in} \end{cases}$$

and

$$\forall s\in S_a: M_{02}(s) \leftarrow \begin{cases} 0 & s \neq s_{in} \\ 1 & s = s_{in} \end{cases}$$

We can prove that $L(\Sigma) = L(\Sigma_1) \parallel L(\Sigma_2)^{\alpha}$.

We construct two finite automatons $FSM_1(\Gamma, Q_1, \delta_1, q_{01}, q_f)$ and $FSM_2(\Gamma, Q_2, \delta_2, q_{02}, q_f)$, as follows:

- $\Gamma \leftarrow T$
- $Q_1 \leftarrow R(M_{01})$, $Q_2 \leftarrow R(M_{02})$
- $\delta_1 \leftarrow \{(M_{i1}, t, M_{j1}) | M_{i1}, M_{j1} \in R(M_{01}) \wedge (M_{i1}[t{>}M_{j1})\}$
  $\delta_2 \leftarrow \{(M_{i2}, t, M_{j2}) | M_{i2}, M_{j2} \in R(M_{02}) \wedge (M_{i2}[t{>}M_{j2})\}$
- $q_{01} \leftarrow M_{01}$, $q_{02} \leftarrow M_{02}$
- $q_f \leftarrow M_e$ $(\forall s \in S_a\text{-}S_f, M_e(s) = 0)$

The proof that $L(\Sigma)$ can be recognized by two paralleling automatons FSM_1 and FSM_2, is similar to that of Theorem 1 and that of Theorem 3, so the proof details are ignored here.

ALGORITHM DESIGN AND COMPLEXITY ANALYSIS

Here, we present the recognition algorithms for the languages of the four kinds of S-nets and analyze the complexity of the proposed algorithms. First, a construction algorithm for a finite automaton that can be used to recognize the language of a bounded Petri net is proposed. This construction algorithm is applied for the language recognition of each kind of S-net.

A construction algorithm for a finite automaton
Algorithm 1: Pr oFSM (Σ)

Input: A bounded Petri net $\Sigma = (S, T; F, M_0, S_f)$
Output: A finite automaton $FSM = (\Gamma, Q, \delta, q_0, q_f)$ used to recognize the language of Σ

```
Begin:
Step 0: $\Gamma \leftarrow T$, $Q \leftarrow \{M_0\}$, $\delta \leftarrow \varnothing$, $q_0 \leftarrow \{M_0\}$, $q_f \leftarrow \{M_e\}$, ($\forall s \in S\text{-}S_f$, $M_e(s) = 0$). Let $M_0$ be the original state and label it new
Step 1: While there exists a state labeled by new Do
        Select any one state which is labeled as new and assume that it is M
Step 2: If there exists a state same as M in the directed path from $M_0$ to M then change the label of M with old and go to Step 1
Step 3: If $M = M_e$ then change the label of M as terminal state and go to Step 1
Step 4: For any $t \in T$such that $M[t >$ Do
        • Compute the state M' after firing t, where $M[t{>}M'$
        • Add a new state M', change the label of it as new and draw a directed arc from M to M'. The arc is labeled with t
        • $Q \leftarrow Q \cup \{M'\}$, $\delta \leftarrow \delta \leftarrow \{\{M, t, M'\}\}$
        End For
        Change the label of M as old, go to Step 1.
        End While.
Step 5: Output FSM $(\Gamma, Q, \delta, q_0, q_f)$.
End.
```

Theorem 5: The time complexity of Algorithm 1 is $O(k^2 mn)$, where n is the number of places, m is the number of transitions and k is the number of reaching states of the given Petri net.

Proof: Let, $\Sigma = (S, T; F, M_0, S_f)$ be a bounded Petri net. The number of its places is n, the number of its transitions is m and the number of its reaching states is k. In Algorithm 1, the factors determine its time complexity as follows:

(1) Store the initial state M_0: The time is n
(2) Determine the enabled transitions in state M: To find the transitions enabled in state M, we should check all the n places. If the marking of this place is not 0, its output transition can fire. The time to execute the above operate is n*m. Since there exist at most k states in the Petri net, the run time is not more than k*n*m
(3) Generate new state: Because the total number of transitions enabled in state M may be m, at most m new states will be generated. Then we should check whether the new state has already been generated. Every time we may check k times (the states of Petri net is k), so the time is k*m. Also, every state includes n units and the Petri net has k states. So the time to generate new state may be k*n*k*m
(4) Store state transfer function: The time is k

Above all, the time of the whole algorithm is n+k*n*m+k*n*k*m+k, so the time complexity is $O(k^2mn)$.

Recognition algorithm for the language of a marking S-graph:
Algorithm 2 s_ident (Σ, σ)

Input: A marking S-graph $\Sigma = (S, T; F, M_0, S_f)$ and the string to be recognized σ;
Output: OK or ERROR;
// If the output is OK, it means that the string σ can be recognized by the marking S-graph. If the output is ERROR, it means that the string σ can not be recognized.

```
Begin:
Step 1: Generate the finite automaton of a marking S-graph $\Sigma = (S, T; F, M_0, S_f)$ using ProFSM ($\Sigma$)
Step 2: $\sigma \leftarrow \sigma_1 \bullet \sigma_2 \bullet \sigma_3 \bullet \ldots \bullet \sigma_n$, $i \leftarrow 1$ and $M \leftarrow M_0$
Step 3: If i<n go to Step 4, else go to Step 5
Step 4: If there exists $\delta$ (M, $\sigma_i$, M), then M ← M', i ← i+1,
        go to Step 3, else output ERROR, End
Step 5: If M = $M_e$, output OK, else output ERROR
End.
```

Theorem 6: The time complexity of Algorithm 2 is O ($k^2mn + N_\sigma k^2$), where n is the number of places, m is the number of transitions, k is the number of reaching states of the given marking S-graph and N_σ is the length of the transition sequence to be recognized.

Proof: Let, $\Sigma = (S, T; F, M_0, S_f)$ be a marking S-graph. The number of its places is n, the number of its transitions is m, the number of reaching states is k and the length of the transition sequence to be recognized is N_σ. In Algorithm 2, the factors determine its time complexity as follows:

(1) Generate finite automaton: In Algorithm 1, the time complexity is O (k^2mn)
(2) Store the transition sequence: The time is N_σ
(3) Recognize the transition sequence: If we want to check whether the transition sequence can be recognized by the Petri net, we should check every character of the sequence. Every time we should search all the state transition functions to determine which function matches. There exists k states, so there may be at most C^2_K state transition functions. The length of the transition sequence is N_σ, so the time is $N_\sigma * C^2_K$. So the time complexity is O (k^2N_σ)

Therefore, the time complexity of Algorithm 2 is O ($k^2mn + N_\sigma k^2$).

Recognition algorithm for the language of a marking S-net with terminal transitions but without primitive transitions:
Algorithm 3 s_ident2 (Σ, σ)

Input: A marking S-net $\Sigma = (S, T; F, M_0, S_f)$ with terminal transitions but without primitive transitions and the string to be recognized σ.
Output: OK or ERROR
// If the output is OK, it means that the string σ can be recognized by the marking S-net. If the output is ERROR, it means that the string σ can not be recognized. Steps are as folows:

```
Step 1: With Theorem 2, change $\Sigma = (S, T; F, M_0, S_f)$ to
        $\Sigma' = (S', T; F', M'_0, S'_f)$
Step 2: Generate the finite automaton of the marking
        S-graph $\Sigma' = (S', T; F', M'_0, S'_f)$ using ProFSM
        ($\Sigma'$)
Step 3: $\sigma \leftarrow \sigma_1 \bullet \sigma_2 \bullet \sigma_3 \bullet \ldots \bullet \sigma_n$, i ← 1 and M ← $M_0$
Step 4: If i<n go to Step 5, else go to Step 6
Step 5: If there exists $\delta$ (M, $\sigma_i$, M'), then M ← M', i ← i+1,
        go to Step 4; else output ERROR, End
Step 6: If M = $M_e$, output OK, else output ERROR
End.
```

Theorem 7: The time complexity of Algorithm 3 is O ($k^2mn+N_\sigma k^2$), where n is the number of places, m is the number of transitions, k is the number of the reaching states of the given marking S-net and N_σ is the length of the transition sequence to be recognized.

Proof: Let, $\Sigma = (S, T; F, M_0, S_f)$ be a marking S-net with terminal transitions but without primitive transitions. The number of its places is n, the number of its transitions is m, the number of its reaching states is k and the length of the transition sequence to be recognized is N_σ. The difference between Algorithm 2 and 3 is the process of the terminal transitions. The time to find and cancel terminal transitions is m*n and the complexity is O (mn). After the process, the total number of its places in the marking S-net becomes n+1, so the time complexity of generating finite automaton and recognition algorithm for the language becomes O (k^2m (n+1)+$N_\sigma k^2$), so the time complexity of Algorithm 3 is O ($k^2mn+N_\sigma k^2$).

Recognition Algorithm for the language of an S-net with primitive transitions only:
Algorithm 4 s_ident3 (Σ, σ)

Input: An S-net $\Sigma = (S, T; F, M_0, S_f)$ with primitive transitions only and the string to be recognized σ;
Output: OK or ERROR
// If the output is OK, it means that the string σ can be recognized by the S-net. If the output is ERROR, it means that the string σ can not be recognized. Steps are as folows:

```
Step 1:  Generate $\Sigma' = (S', T; F', M'_0, S'_f)$ with Theorem 3
Step 2:  Generate the finite automaton of the marking
         S-graph $\Sigma' = (S', T; F', M'_0, S'_f)$ using ProFSM
         ($\Sigma'$)
Step 3:  $\sigma \leftarrow \sigma_1 \| \sigma_2 \| \sigma_3 \| \ldots \| \sigma_n$, i ← 1, j ← 1 and M ← $M_0$
Step 4:  If j≠n, go to Step 5, else go to Step 10
Step 5:  If i≤n, go to Step 6, else go to Step 8
Step 6:  If $\sigma_i \neq 0$ go to Step 7, else i ← i+1 go to Step 4
Step 7:  If there exists $\delta$ (M, $\sigma_i$, M') then M ← M', $\sigma_i$←0,
         i ← i+1, j ← j+1, go to Step 4, else i ← i+1 go to
         Step 4
Step 8:  If M = $M_e$ then i ← 1, M ← $M_0$, go to Step 4, else
         go to Step 9
Step 9:  Output ERROR, End.
Step 10: Output OK, End
```

Theorem 8: The time complexity of Algorithm 4 is O ($k^2mn + DN_\sigma k^2$), where n is the number of places, m is the number of transitions, k is the number of reaching states of the given S-net and N_σ is the length of the transition sequence to be recognized and D is the number of primitive transitions included in transition sequence.

Proof: Let, $\Sigma = (S, T; F, M_0, S_f)$ be an S-net with primitive transitions only. The number of its places is n, the number of its transitions is m, the number of its reaching states is k and the length of the transition sequence to be recognized is N_σ and the number of primitive transitions included in transition sequence is D.

The difference between Algorithm 2 and 4 is the process of primitive transition and the recognition algorithm must be execute D times. The time to find and cancel primitive transition is m*n and therefore, the time complexity is O (mn). After the process, the total number of its places becomes n+1. According to Theorem 5, the time complexity of generating finite automaton is O ($k^2mn(n+1)$) and in Theorem 6, the time complexity of recognition algorithm executing one time is O ($N_\sigma k^2$), so the time complexity of the recognition algorithm executing D times is O ($DN_\sigma k^2$).

Therefore, the time complexity of Algorithm 4 is O ($k^2mn+DN_\sigma k^2$).

Recognition algorithm for the language of a marking S-net with primitive transitions only:
Algorithm 5 s_ident4 (Σ ,σ)

Input: A marking S-net $\Sigma = (S, T; F, M_0, S_f)$ with primitive transitions only and the string to be recognized σ;
Output: OK or ERROR
// If the output is OK, it means that the string σ can be recognized by the marking S-net. If the output is ERROR, it means that the string σ can not be recognized. Steps are as folows:

```
Step 1: Generate $\Sigma_1 = (S_a, T; F_a, M_{01}, S_f)$, $\Sigma_2 = (S_a, T; F_a, M_{02}, S_f)$ using Theorem 4
Step 2: Generate finite automaton $FSM_i$ of marking S-graph $\Sigma_i = (S_a, T; F_a, M_{0i}, S_f)$ using ProFSM ($\Sigma_i$) ($i \in \{1, 2\}$)
Step 3: $\sigma \leftarrow \sigma_1\|\sigma_2\|\sigma_3\|...\|\sigma_n$, $i \leftarrow 1$, $j \leftarrow 1$ and $M \leftarrow M_{01}$
Step 4: Delete the transition sequence already been recognized by $FSM_1$
    4.1 If $j \neq n$, go to 4.2, else go to Step 7
    4.2 If $i \leq n$, go to 4.3, else go to 4.5
    4.3 If $\sigma_i \neq 0$, go to 4.4, else $i \leftarrow i+1$ go to 4.1
    4.4 If there exists $\delta(M, \Sigma_i, M')$ then $M \leftarrow M'$, $\sigma_i \leftarrow 0$, $i \leftarrow i+1$, $j \leftarrow j+1$ go to 4.1, else $i \leftarrow i+1$ go to 4.1
    4.5 If $M = M_e$ then $i \leftarrow 1$, $M \leftarrow M_{02}$ go to Step 5, else go to Step 6
Step 5: Delete the transition sequence already been recognized by $FSM_2$
    5.1 If $j \neq n$, go to 5.2, else go to Step 7
    5.2 If $i \leq n$, go to 5.3, else go to 5.5
    5.3 If $\sigma_i \neq 0$ go to 5.4, else $i \leftarrow i+1$ go to 5.1
    5.4 If there exists $\delta(M, \Sigma_i, M')$ then $M \leftarrow M'$, $\sigma_i \leftarrow 0$, $i \leftarrow i+1$, $j \leftarrow j+1$ go to 5.1, else $i \leftarrow i+1$ go to 5.1
    5.5 If $M = M_e$ then $i \leftarrow 1$, $M \leftarrow M_{02}$ go to 5.1, else go to Step 6
Step 6: Output ERROR, End
Step 7: Output OK, End
```

Theorem 9: The time complexity of Algorithm 5 is O ($k^2mn + DN_\sigma k^2$), where n is the number of places, m is the number of transitions, k is the number of reaching states and N_σ is the length of the transition sequence to be recognized and D is the number of primitive transitions included in the transition sequence.

Proof: Let, $\Sigma = (S, T; F, M_0, S_f)$ be a marking S-net with primitive transitions only and the number of its places is n, the number of its transitions is m, the number of its reaching states is k and the length of the transition sequence to be recognized is N_σ and the number of primitive transitions included in transition sequence is D.

With Algorithm 5, we know there are two finite automatons, so the recognition time includes two parts. One is similar with Algorithm 2 and another is similar with Algorithm 4. According to Algorithm 2, 4 and Theorem 6, 8, it is easy to prove that the time complexity of Algorithm 4.5 is O ($k^2mn + DN_\sigma k^2$).

EXPERIMENT TESTS

We implemented all Algorithms in Section 4 with VC++ 6.0 and with the programs we have tested the time

Table 1: Running time of the test example

S-net	Transition sequence	Parameter (m, n, N_σ, k, D)	Recognizability	Running time (sec)
Figure 1a	$t_1t_2t_4t_3t_1t_3t_4t_4t_2t_2t_1t_1$	m = 4, n = 3, N_σ = 12, k = 6	OK	256
Figure 1a	$t_1t_2t_4t_3t_1t_3t_4t_4t_2t_1t_1t_1$	m = 4, n = 3, N_σ = 12, k = 6	ERROR	249
Figure 1b	$t_1t_2t_1t_2t_1t_2t_1t_2t_1t_2t_1t_2t_1t_3$	m = 3, n = 2, N_σ = 14, k = 3	OK	84
Figure 1b	$t_1t_2t_1t_2t_1t_2t_1t_2t_1t_2t_1t_2t_1t_3$	m = 3, n = 2, N_σ = 14, k = 3	ERROR	83
Figure 1c	$t_1t_2t_3t_2t_1t_1t_2t_3t_2t_2t_1t_2t_3t_2$	m = 3, n = 2, N_σ = 14, k = 3, D = 4	OK	126
Figure 1c	$t_1t_2t_3t_2t_1t_1t_2t_2t_1t_2t_3t_2t_3t_1$	m = 3, n = 2, N_σ = 14, k = 3, D = 5	ERROR	100
Figure 1d	$t_2t_3t_1t_2t_2t_3t_3t_2t_3t_2t_4t_2t_5$	m = 5, n = 3, N_σ = 13, k = 3/4	OK	282
Figure 1d	$t_2t_3t_1t_2t_2t_3t_3t_2t_3t_2t_4t_1t_2t_3t_3$	m = 5, n = 3, N_σ = 13, k = 3/4, D = 2	ERROR	300

complexity of all kinds of S-net. The running time is in Table 1. From Table 1, we can see all the running times of the given test examples are consistent with the conclusions about the time complexity of the proposed algorithms.

CONCLUSIONS

Although, perfect solutions to problems of Petri net have been given by many researches, most of them regard Petri net as language generators and only a few people (Jiang *et al*., 1998) recognize the possibility of utilization of Petri net as language generators. Jiang *et al*. (1998) give several recognition algorithms for Petri nets languages, but they are too complicated to use for ordinary Petri nets. S-net is a kind of structure-simple Petri nets and the recognition algorithms for S-net language will be of great benefit to the further study of languages recognition for ordinary Petri nets.

In this study, first we give the recognition algorithms of S-nets that are classified according to the primitive and terminal transitions. Second, we analyze the time complexity of these algorithms. Finally, by several groups of experimental data, the time complexities of those proposed algorithms are checked.

ACKNOWLEDGMENT

This study is supported partly by the NSFC under Grant No. 60603090, 90718011 and 50875158; the Excellent Young Scientist Foundation of Shandong Province of China under Grant No. BS2009DX004; the Special Fund for Fast Sharing of Science Paper in Net Era by CSTD (20093718110008); the Open Project of Computer Architecture Lab of ICT, CAS (No. ICT-ARCH200807); the Research Foundation of Shandong Educational Committee under Grant No. J08LJ77; Shandong Natural Science Foundation for Distinguished Young scholars of China No. JQ200816 and the Taishan Scholar Program of Shandong Province.

REFERENCES

Garg, V.K. and M.T. Ragunath, 1992. Concurrent regular expression and their relationship to Petri nets. Theoretical Comput. Sci., 96: 285-304.

Jiang, C.J., Y.P. Zheng and S.G. Shu, 1997. Design and optimization of system based on language of Petri net. J. China Inst. Commun., 18: 27-33.

Jiang, C.J., Z.H. Wu and C.H. Wang, 1998. Recognizer of PN language. Acta Electronic Sinica, 26: 127-129.

Jiang, C.J. and W.M., Lu, 2001. On properties of concurrent system based on Petri net language. J. Software, 12: 512-520.

Wu, Z.H., 2006. Introduction to Petri Net. China Machine Press, Beijing.

Zeng, Q.T. and Z.H. Wu, 2002. The language characters analysis of analogous s-graph. Comput. Sci., 29: 120-122.

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

Zeng, Q. and Z. Wu, 2004. Language behavior description of structure-complex. Petri net based on decomposition. J. Syst. Eng., 19: 300-306.

Zeng, Q.T., Z.H. Wu and B.X. Ma, 2004. Process and language expression of Petri net. J. China Comput. Syst., 25: 654-658.