Learning of finite coded Petri net languages

S. Kannamma[1], D.G. Thomas[2] and K. Rangarajan[3]
[1]Department of Mathematics, S.D.N.B. Vaishnav College for Women
Chennai - 600 044. E-mail : Kannamma_s@yahoo.com
[2]Department of Mathematics, Madras Christian College
Chennai - 600 059. E-mail : dgthomasmcc@yahoo.com

Abstract

We present a class of Coded Petri net languages and study some algebraic properties. The purpose of introduction of this language is to bring out its usefulness in learning theory. We introduce an algorithm for learning a finite coded Petri net language and its running time is bounded by a polynomial function of given inputs.
Key words : Codes, Petri net, Learning, Concurrency, Automata.

1 Introduction

E. M. Gold [4] has established that languages containing all finite sets and one infinite set is not identifiable in the limit from positive data, which leads to the fact that regular languages is not identifiable in the limit from positive data. This negative result has initiated a search for subclasses of regular languages, having the desirable inference property.

Angluin [1] has shown that k-reversible languages, which form a subclass of regular languages, are identifiable in the limit from positive data. Other subclasses include Szilard languages of regular grammar [6], strictly regular languages [11] and code regular languages [3].

Analogous to code regular language, we define coded Petri net language and analyse its properties. An algorithm similar to the one in [5, 12] is used to learn finite coded Petri net language (from a given positive sample) whose initial and final markings are the same. The algorithm develops a marked graph whose transitions are code words and whose language contains the given positive sample.

2 Basic concepts and definitions

We now recall the notion of Petri net languages as given by J. L. Peterson [10]. The definition of Petri net is in style to the definition in automata theory. It defines a machine, the Petri net (machine) automaton. This point of view leads to some interesting results in formal language theory and automata theory.

Definition 2.1 *A Petri net structure is a four tuple* $C = (P, T, I, O)$, *where* $P = \{p_1, p_2, \ldots, p_n\}$ *is a finite set of places,* $T = \{t_1, t_2, \ldots, t_m\}$ *is a finite set of transitions,* $n, m \geq 0$; $P \cap T = \phi$; $I : T \to P^{\infty}$ *is the input function from transitions to bags of places and* $O : T \to P^{\infty}$ *is the output function from transitions to bags of places.*

Definition 2.2 *Petri net marking is an assignment of tokens to the places of a Petri net. Tokens can be thought to reside in the places of a Petri net. The number and position of tokens may change during the execution of a Petri net. The tokens are used to define the execution of a Petri net. The marking can also be defined as an n-vector* $\mu = (\mu_1, \mu_2, \ldots, \mu_n)$, *where* $n = |P|$; μ_i = *number of tokens in pi,* $i = 1, 2, \ldots, n$. *(Also, we can write* $\mu(pi) = \mu_i$*).*

Definition 2.3 *A marked Petri net* $M = (C, \mu)$ *is a Petri net Structure* $C = (P, T, I, O)$ *and a marking* μ. *This is also written as* $C = (P, T, I, O, \mu)$.

Definition 2.4 (Execution rule for marked Petri net) *A Petri net executes by firing transitions. A transition may fire if it is enabled. A transition is enabled if each of its input places has at least* $w(p, t)$ *tokens in it, where* $w(p, t)$ *is the weight of the arc from p to t. A transition fires by removing* $w(p, t)$ *tokens from each of its input places and then depositing* $w(t, p)$ *tokens into each of its output places, where* $w(t, p)$ *is the weight of the arc from t to p. If one of the input places of a transition contains no tokens, then that transition is not enabled.*

Firing sequence of transitions: Refer the example shown in figure 1. $\mu = (1, 2, 0, 0, 1)$.

Step 1 Initial marking is $(1, 2, 0, 0, 1)$. t_1 is the only transition that is enabled.

Step 2 When t_1 fires, it removes the only token at p_1 and deposits a token in each of its output places, i.e., p_2, p_3, p_5 : the marking now becomes $(0, 3, 1, 0, 2)$. Now both t_2 and t_3 are enabled.

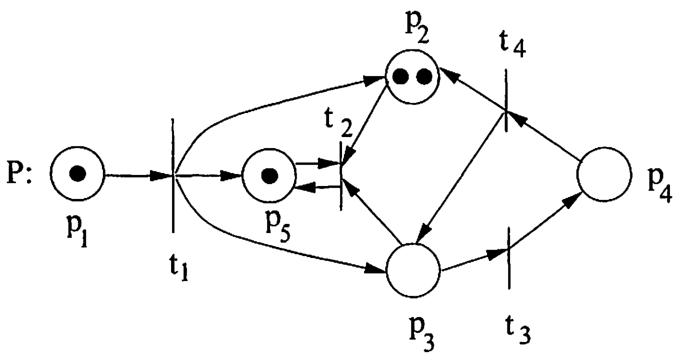


Figure 1:

Step 3a Suppose t_2 fires: Firing of t_2 removes a token from each of its input places, namely, p_2, p_3 and p_5 and deposits one token in p_5. (In effect the number of tokens in p_5 is unaltered). Marking now is $(0, 2, 0, 0, 2)$. No other transition is enabled-hence no execution can take place.

Step 3b Suppose t_3 fires: Firing of t_3 removes the only token at p_3 and adds one in p_4. The marking becomes $(0, 3, 0, 1, 2)$. t_4 is the only transition that is enabled.

Step 4 Suppose t_4 fires: the only token at p_4 is removed and one each is deposited at p_2 and p_3. The marking is $(0, 4, 1, 0, 2)$.

Once again, t_2 and t_3 are enabled - when t_3 fires it removes the token from p_3 and adds one in p_4 which makes t_4 enabled again - this process continues until t_2 fires.

Hence the sequence of firing transitions may be either t_1t_2 or $t_1t_3t_4t_2$ or $t_1t_3t_4t_3t_4t_2 \ldots$ i.e., $(t_1(t_3t_4)^*t_2)$.

Definition 2.5 *A labeled Petri net $\gamma = (C, \sigma, \mu, F)$, where $C = (P, T, I, O)$ is a Petri net Structure; $\sigma : T \to \Sigma$ is a morphism defined by $\sigma(t_i) = a_i \in \Sigma$, where Σ is an alphabet (σ can be extended in a natural way on T^*); μ is the initial marking of C and $F \subseteq P$ is a set of final places. A Petri net Language of a Labeled Petri net is defined as $L(\gamma) = \{\sigma(\beta) \in \Sigma^* / \beta \in T^*$ and $\delta(\mu, \beta) \in F\}$ where $\delta : N^n \times T^* \to N^n$ is a function where N is the set of natural numbers including 0.*

Definition 2.6 *A labeled Petri net $\gamma = (C, \sigma, \mu, F)$ with language $L(\gamma)$ in standard form satisfies the following properties:*

1. *The initial marking μ consists of exactly one token in start place p_s and zero tokens elsewhere; $p_s \notin O(t_j)$ for any $t_j \in T$.*

2. *There exists a place $p_f \in P$ such that*
    (a) $F = \{p_f\}$, *if* $\lambda \notin L(\gamma)$ *or* $F = \{p_s, p_f\}$, *if* $\lambda \in L(\gamma)$.
    (b) $p_f \notin I(t_j)$ *for all* $t_j \in T$.
    (c) $\delta(\mu', t_j)$ *is undefined for all* $t_j \in T$ *and* $\mu' \in R(C, \mu)$ *which have a token in* p_f.

It is true that every Petri net is equivalent to a Petri net in standard form.

3 Coded Petri net and coded Petri net language

In this section, we define a new class of Petri nets called coded Petri nets. We define the class of languages generated as coded Petri net languages.

Definition 3.1 *A labeled extended Petri net* $\gamma = (C, K, \sigma, \mu, F)$ *where* $C = (P, T, I, O)$ *is a Petri net structure,* K *is a set of labels* $w \in \Sigma^+$ *and labeling function* $\sigma : T \to K$ *is defined by* $\sigma(t_i) = w \in K$. *A Petri net language of a labeled extended Petri net is defined as* $L(\gamma) = \{\sigma(\beta) \in \Sigma^+ : \beta \in K^*$ *and* $\delta(\mu, \beta) \in F\}$.

In the above definition, suppose K is taken as a code set, then the corresponding extended Petri net is called coded Petri net.

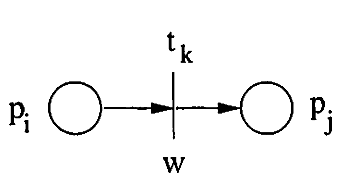
Definition 3.2 *A set* K *of words of* Σ^+ *forms a code over* Σ, *if every word of* Σ^+ *has at most one factorization using the words of* K.

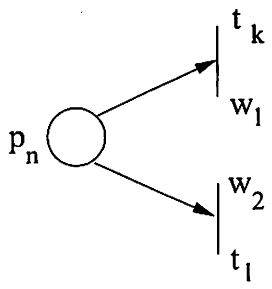
For example, $\{ab, ba\}$ is a code set over $\Sigma = \{a, b\}$, whereas $\{ab, a, b\}$ is not a code set over Σ. (The word $abab$ has factorizations as $(ab)(a)(b)$, $(a)(b)(a)(b)$, $(ab)(ab)$, $(a)(b)(ab)$).

Definition 3.3 *A coded Petri net (CPN) is a labeled extended Petri net, where* K *satisfies the code property and* σ *satisfies the following properties:*

1. *Uniqueness of code words: For any word* w *in* K, *there exists a unique pair of places* p_i *and* p_j *and transition* t_k *with the code word* w *such that* $\sigma(t_k) = w$; $I(t_k) = \{p_i\}$ *and* $O(t_k) = \{p_j\}$. *i.e., every transition has exactly one input place and one output place.*

2. *Uniqueness of first letters of code words for transitions: Suppose that there are two transitions* t_k *and* t_l *and a place* p_n *such that* $p_n \in I(t_k), p_n \in I(t_l), w_1 = \sigma(t_k)$ *and* $w_2 = \sigma(t_l)$ *where* w_1 *and* w_2 *are in* K, *then* w_1 *and* w_2 *differ in their first letters.*





If γ is a coded Petri net, then $L(\gamma)$ is defined as coded Petri net language.

The class of coded Petri net languages is denoted as L(CPNL).

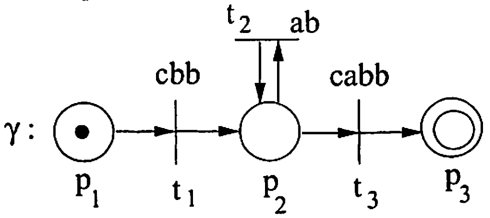
Definition 3.4 $\gamma = (C, K, \sigma, \mu, F)$ *is called a bifix coded Petri net if γ is a coded Petri net and the code set K forms a bifix code (both prefix code and suffix code) i.e., no word in K is a proper prefix or a proper suffix of another word in it. $L(\gamma)$ is known as bifix coded Petri net language.*

The class of bifix coded Petri net languages is denoted as L(BCPNL).

4 Properties of CPNL

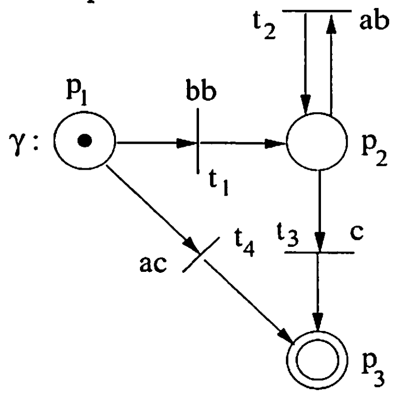
In this section we give some examples of CPNs and their languages CPNLs. We also prove a few algebraic properties of CPNLs and establish that a code word assigned to a transition of CPN is unique.

Example 4.1



$F = \{p_3\}; \mu = (1, 0, 0)$;
$K = \{ab, cbb, cabb\}$ is a bifix code set.
γ is in standard form.
$L(\gamma) = \{(cbb)(ab)^*(cabb)\}$ is a BCPNL.

Example 4.2



$F = \{p_3\}; \mu = (1,0,0)$;
$K = \{ab, ac, bb, c\}$ is not a bifix code set.

γ is in standard form.
$L(\gamma) = \{(bb)(ab)^*(c)\} \cup \{ac\}$ is a CPNL.

It is true that in a coded Petri net, any code word $\alpha \in K$ occurs only once as a label of a transition. This is due to the conditions in definition 3.3. Theorem 4.1 establishes this result.

Theorem 4.1 *Let in a coded Petri net $\gamma = (C, K, \sigma, \mu, F)$ where $C = (P, T, I, O)$, for two places $p, p' \in P$ and $\alpha \in K^+$, $\delta(p, \alpha) = q$ and $\delta(p', \alpha) = q'$ for $q, q' \in P \Rightarrow p = p'$ and $q = q'$.*

Proof: (By induction)
Let $\alpha = u_1 u_2 \dots u_n$, $u_i \in K$, $i = 1, 2, \dots, n$.
Let $n = 1$, $\alpha = u_1 \in K$;
$\delta(p, \alpha) = \delta(p, u_1) = q$ and
$\delta(p', \alpha) = \delta(p', u_1) = q' \Rightarrow p = p'$ and $q = q'$. (By uniqueness of code words)
Let $n = k$; let the statement hold good for k. i.e., if $\delta(p, \alpha) = q$ and $\delta(p', \alpha) = q'$, where

$$\alpha = u_1 u_2 \dots u_k,\ \text{each } u_i \in K, 1 \le i \le k,\ \text{then } p = p' \text{ and } q = q' \qquad (1)$$

$$\beta = \alpha u_{k+1},\ \text{where } u_{k+1} \in K,\ \delta(p, \beta) = r \text{ and } \delta(p', \beta) = r'.$$

$$\delta(p, \alpha u_{k+1}) = r \Rightarrow \delta(\delta(p, \alpha), u_{k+1}) = r \Rightarrow \delta(q, u_{k+1}) = r \qquad (2)$$
$$\delta(p', \alpha u_{k+1}) = r' \Rightarrow \delta(\delta(p', \alpha), u_{k+1}) = r' \Rightarrow \delta(q', u_{k+1}) = r' \qquad (3)$$

(1), (2) and (3) $\Rightarrow p = p'$ and $r = r'$. Hence the theorem is true for $k + 1$ and the induction is complete. □

Theorem 4.2 *The class L(CPNL) is not closed under the algebraic operations of concatenation, union, reversal, homomorphism on the alphabet, and complement of the language.*

Sl. No.	*Operation* *Example*
1	**Concatenation** $K = \{ab, cbaa, cbb, ccb\}$; $L_1 = \{(ccb)(ab)\}$ $L_2 = \{(cbb)(ab)(cbaa)\}$. $L_1L_2 = \{(ccb)(ab)(cbb)(ab)(cbaa)\}$ *is not a CPNL, since* $ab \in K$ *occurs twice in* L_1L_2 *violating condition (1) of Definition 3.3.*
2	**Union** $L_1 = \{aa\}$; $L_2 = \{aaa\}$; $L_1 \cup L_2 = \{aa, aaa\}$ *is not a CPNL. since condition (2) of Definition 3.3 is violated.*
3	**Reversal** *If* $L = \{(ac)(bac)^*(cb)\}$, *then* $L^R = \{(bc)(cab)^*(ca)\}$, *which is not a CPNL (violates condition (2) of Definition 3.3).*
4	**Homomorphism** $L = \{(ab)^m c(ba)^n : m, n \geq 0\}$. *Define* h *on* Σ *by* $h(a) = a$, $h(b) = b$ *and* $h(c) = \lambda$. *Then* $h(L) = \{(ab)^m(ba)^n : m, n \geq 0\}$ *is not a CPNL since condition (1) of Definition 3.3 is violated.*
6	**λ-free homomorphism** $L = \{a^m b^2 : m \geq 0\}$. *Define* h *on* Σ *by* $h(a) = h(b) = ab$. *Then* $h(L) = \{(ab)^m : m \geq 2\}$, *is not a CPNL since both the conditions of Definition 3.3 are violated.*
5	**Complement** $L = \{\lambda, a\}$; *then* $L^c = \{a^n : n \geq 2\}$, *is not a CPNL since both the conditions of Definition 3.3 are violated.*

5 Learning of finite coded Petri net language

The concept of identification in the limit formulated by E.M. Gold [4] has been of basic importance in theoretical studies of inductive inference. He has established that the class of languages containing finite languages and at least one infinite language cannot be identified in the limit from positive sample. Hence, the class of regular languages is not identifiable in the limit from positive sample. As a consequence, learning subclasses of regular languages with positive data has gained importance [1, 3, 6, 11, 12]. Angluin [1] has established that k - reversible languages which form a subclass of regular languages is identifiable in the limit from positive data.

Algorithms for learning code regular languages are presented [3] by J.D. Emerald in the framework of identification in the limit. E. Makinen has proved [6] that given a finite sample of positive data, the problem of finding a Szilard language of a linear grammar compatible with the sample can be solved.

N. Tanida and T. Yokomori [11] have introduced a subclass of DFA's called strictly deterministic automata (SDA's) and shown that the class is learnable in the limit from positive data requiring polynomial time for updating conjectures.

Here, we propose an algorithm analogous to the one developed in [5, 12], for learning finite coded Petri net language which is a sub class of regular languages. It consists of two phases. In the first phase, each example is presented to the algorithm until the language of the target system is identified in the form of a finite state acceptor. This phase is related to inductive inference of regular languages [1, 4, 6]. In the second phase, the dependency relation is extracted from the language and the structure of a Petri net is guessed. This algorithm is for a class of live and safe Petri nets and its running time is bounded by a polynomial function of given inputs.

Definition 5.1 *For a Petri net* $C = (P, T, I, O)$, ${}^\bullet p = \{t_i : O(t_i) = p\}$; $p^\bullet = \{t_i : I(t_i) = p\}$, *for every place* p *in* P.

Definition 5.2 *A marked graph is a Petri net such that* $|{}^\bullet p| = |p^\bullet| = 1$, *for all* p *in* P; *i.e., every place is an input for exactly one transition and an output for exactly one transition.*

Definition 5.3 *A marked graph has a loop at a place* p *if* ${}^\bullet p \cap p^\bullet \neq \phi$.

Definition 5.4 *A five tuple* $\Theta = (S, \Sigma, \delta, S_o, G)$ *is called an acceptor / finite state automaton where* S *is a finite set of states,* Σ *is an alphabet,* $\delta : S \times \Sigma \longrightarrow 2^s$ *is a state transition function,* S_o *in* S *is the initial state and* $G \subseteq S$ *is the set of final states.*

Definition 5.5 *Let* $\Sigma = \{a_i : i = 1, 2, \ldots, r\}$ *be an alphabet. Then the set of sequences including the empty sequence over* Σ *is denoted by* Σ^*. $\#_i(z)$ *is the number of occurrences of* a_i *in* $z \in \Sigma^+$. $\psi : \Sigma^* \longrightarrow N^r$, *defined by* $\psi(z) = (\#_1(z), \#_2(z), \ldots, \#_r(z))$ *is called Parikh mapping.*

Definition 5.6 *Let* $L \subseteq \Sigma^*$. *Then* $Pref(L) = \{\sigma \in \Sigma^* :$ *there exists* $\alpha \in \Sigma^*$ *such that* $\sigma\alpha \in L\}$.

Definition 5.7 *For every positive sample* Q *over* Σ,
$PT(Q) = (Pref(Q), \Sigma, \delta, \lambda, Q)$ *is called prefix tree acceptor for* Q *where* $\delta(s, t) = st$, *whenever* $s, st \in Pref(Q)$ *and* λ *is the empty string.*

Definition 5.8 *Two states s_1 and s_2 of $PT(Q)$ are equal if and only if $|\psi(s_1)-\psi(s_2)|$ is either the zero vector or the vector of the form $(1, 1, \dots, 1)$.*

Definition 5.9 *For $L \subseteq \Sigma^*$, dependency relation D is defined as $D = \{(x, y) \in \Sigma \times \Sigma$: there exists $\sigma \in \Sigma^*$ such that $\sigma xy \in Pref(L)$ and $\sigma y \notin Pref(L)\}$.*

Construction of a marked graph for a given finite positive sample

Throughout this discussion, we consider only live and self-loop free marked graphs whose initial and final markings are the same, viz. m_0=(1, 0, ..., 0). We employ an algorithm similar to the one developed in [5, 12] to construct a marked graph, reading a language containing the given positive sample.

This algorithm develops a finite state automaton, which generates a trace language. The algorithm then extracts a dependency relation of the trace language and guesses the structure of a marked graph from dependency; language of this marked graph contains the given positive sample Q. The running time of the algorithm is bounded by a polynomial function of given inputs.

Step 1 In $PT(Q)$, merge all equal states by definition 5.8 and obtain a prefix tree acceptor $\Theta = (Pref(Q), \Sigma, \delta, \lambda, Q)$, in which all states are different.

Step 2

(a) $C_i = \{y : \delta(s_i, y) \neq \phi\}$

(b) For each state $s' \in \delta(s, x)$, let $Y_1 = \{y : (s, y) \neq \phi\}$ and $Y_2 = \{y : \delta(s', y) \neq \phi\}$. For each $y \in Y_2$, if $y \notin Y_1$ and if $x, y \notin C_i$ for any i, then $(x, y) \in D$ (by definition 5.9).

Step 3 For every pair $(x, y) \in D$, make a place p such that $^\bullet p = x$ and $p^\bullet = y$.

In step 1, stripped acceptor obtained from the sample is produced; step 2 evolves the dependency relation and finally step 3 helps us to construct the marked graph.

Example 5.1 *Consider the code set $K = \{ab, bca, cab, abcc, ac\}$.*
Let $Q = \{(ab)(bca)(cab)(abcc)(ac), (ab)(cab)(bca)(abcc)(ac),$
$(ab)(cab)(abcc)(bca)(ac), (ab)(abcc)(cab)(bca)(ac), (ab)(abcc)(bca)(cab)(ac)\}$.

For convenience, we number the code words as follows: ab as 1, bca as 2, cab as 3, abcc as 4 and ac as 5. Then $Q = \{12345, 13245, 13425, 14325, 14235\}$. $Pref\{Q\} = \{\lambda, 1, 12, 13, 14, 123, 132, 134, 143, 142, 1234, 1324, 1342, 1432, 1423, 12345, \dots\}$.

The distinct states	δ *is defined as below*	$C_i, i = 1, 2, \dots, 8$
$s_o = (0,0,0,0,0)$	$\delta(s_o, 1) = s_1$	$C_o = \{1\}$
$s_1 = (1,0,0,0,0)$	$\delta(s_1, 2) = s_2$ $\delta(s_1, 3) = s_3$ $\delta(s_1, 4) = s_4$	$C_1 = \{2,3,4\}$
$s_2 = (1,1,0,0,0)$	$\delta(s_2, 3) = s_5$ $\delta(s_2, 4) = s_7$	$C_2 = \{3,4\}$
$s_3 = (1,0,1,0,0)$	$\delta(s_3, 2) = s_5$ $\delta(s_3, 4) = s_6$	$C_3 = \{2,4\}$
$s_4 = (1,0,0,1,0)$	$\delta(s_4, 3) = s_6$ $\delta(s_4, 2) = s_7$	$C_4 = \{2,3\}$
$s_5 = (1,1,1,0,0)$	$\delta(s_5, 4) = s_8$	$C_5 = \{4\}$
$s_6 = (1,0,1,1,0)$	$\delta(s_6, 2) = s_8$	$C_6 = \{2\}$
$s_7 = (1,1,0,1,0)$	$\delta(s_7, 3) = s_8$	$C_7 = \{3\}$
$s_8 = (1,1,1,1,0)$	$\delta(s_8, 5) = s_o$	$C_8 = \{5\}$

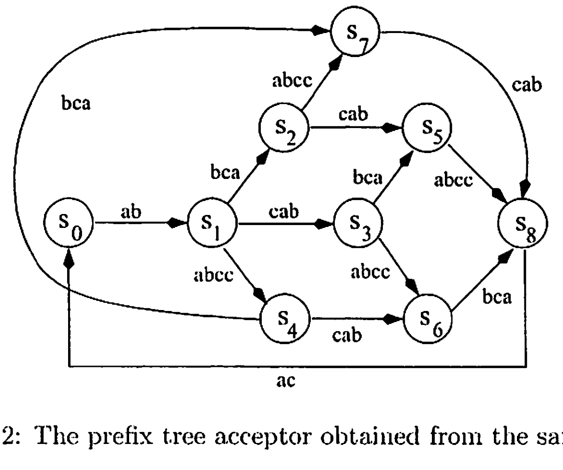


Figure 2: The prefix tree acceptor obtained from the sample Q

$D = \{(1,2), (1,3), (1,4), (2,5), (3,5), (4,5), (5,1)\}$. *The corresponding marked graph is shown in Figure 3.*
The language read by this marked graph is
$\{(ab)(bca)(cab)(abcc)(ac), (ab)(bca)(abcc)(cab)(ac),$
$(ab)(cab)(bca)(abcc)(ac), (ab)(cab)(abcc)(bca)(ac),$
$(ab)(abcc)(cab)(bca)(ac), (ab)(abcc)(bca)(cab)(ac)\} \supset Q$.

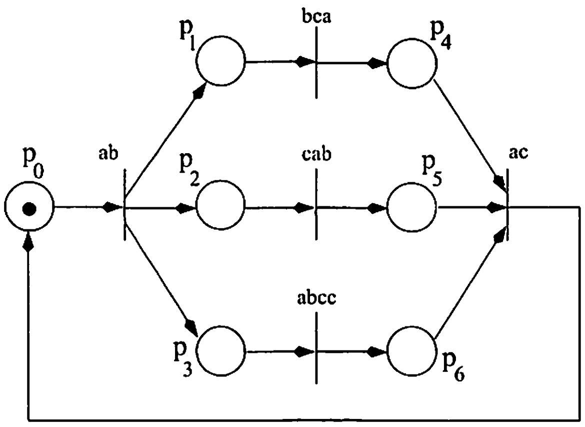


Figure 3: The marked graph

References

[1] D. Angluin, Inference of reversible languages, J. ACM 29, 741-765, 1982.

[2] J. Berstel and B. Perrin, Theory of codes, Academic Press, NY, London, 1985.

[3] J. D. Emerald, K. G. Subramanian, D. G. Thomas, Learning code regular and code linear languages, Proc. of ICGI - 96, Lecture Notes in Artificial Intelligence, Vol. 1147, Springer, Berlin, 211-221, 1995.

[4] E. M. Gold, Language identification in the limit, Information and Control, Vol. 10, 447-474, 1967.

[5] K. Hiraishi, Construction of a class of safe Petri nets by presenting firing sequences, Application and theory of Petri nets, Lecture Notes in Computer Science, Vol. 616, 224-262, 1992.

[6] E. Makinen, The grammatical inference problem for the Szilard language of linear grammars, Information Processing Letters, Vol. 36, 203-206, 1990.

[7] T. Murata, Petri nets, Properties, Analysis and Application, Proceedings of the IEEE, Vol. 77, No. 4, 541-480, 1989.

[8] J. L. Peterson, Computation sequence sets, Journal of Computer System Science, Vol. 13, No. 1, 1-24, 1976.

[9] J. L. Peterson, Petri nets, ACM Computing Surveys, Vol. 9, No. 3, 223-252, 1997.

[10] J. L. Peterson, Petri net theory and modeling of Systems, Prentice Hall Inc., Englewood Cliffs, NJ, 1981.

[11] N. Tanida, T. Yokomori, Polynomial time identification of strictly regular languages in the limit, IEICE Trans. Inform. Syst., E 75. D, 125-132, 1992.

[12] K. Thirusangu, K. Rangarajan, A note on the construction of marked graphs, Information Processing Letters, Vol. 55, 211-215, 1995.