Int. j. inf. tecnol. (August 2024) 16(6):3663–3676
https://doi.org/10.1007/s41870-024-01903-0

ORIGINAL RESEARCH

Partial Petri Net Languages and their Properties

A. Mahadeer[1] · R. Arulprakasam[1] · V. R. Dare[2]

Received: 22 January 2024 / Accepted: 23 April 2024 / Published online: 5 June 2024

Abstract The languages derived from Petri Net provides a efficient technique for verification, validation and synthesis for the system. Partial words are extensively used in the fields such as bioinformatics, pattern matching and text searching. In this paper, we introduce a Partial Petri Net and then define its associated languages. Further, we discuss the closure properties that hold over these derived languages.

Keywords Partial words · Partial languages · Petri Nets · Closure properties

Mathematics Subject Classification 68Q45 · 68Q70 · 68Q85

1 Introduction

The Petri Net, a formal model for the flow of information, was first presented in 1962 by Carl Adam Petri [1]. Petri Nets have found applications in the analysis of systems that exhibit concurrent, asynchronous, distributed, parallel, non deterministic and stochastic behaviors. Tokens, which are identical and represented as black dots, are used in Petri Nets to emulate the system's dynamic and simultaneous processes. A language can be associated with the execution of a Petri Net. By formulating a labeling function for transitions over a given alphabet, a language over the alphabet is generated from all firing sequence set that origin from a particular starting marking and lead to a final marking set. The firing rule is solely dependent on the presence of these tokens and the number available in the input places for the transition to fire [1]. Henry Baker investigated Petri Net and its languages in 1972 [2]. This was followed by Michael Hack on Petri Net languages in 1976 [3] and in the same year, James L. Peterson introduced Petri Net as an automaton and investigated its languages [4]. In 1978, Starke studied the free terminal language of a Petri Net [5]. Jantzen investigated the hierarchy of Petri Net languages in 1979 [6]. During the year 1981, Valk et al. discussed Petri Nets and regular languages [7], while Araki et al. discussed that flow languages for some restricted class of flow expressions are equivalent to Petri Net languages [8]. Vidal Naquet, in 1982, investigated deterministic languages of Petri Nets [9]. Parigot et al. investigated the Buichi-like theorem, which characterizes Petri Net languages in terms of second-order logical formulas [10]. In 1987, the language theory of Petri Nets was studied by Jantzen [11] and Pelz investigated closure properties of deterministic languages of Petri Nets [12]. Various classes of Petri Net languages have been identified, based on the selection of transition labeling (free, λ-free and with λ-transitions) and the choice of the final markings set. In the literature [1, 11], three options for final markings are commonly chosen.

Tiplea investigated the selective Petri Net languages in 1992 [13]. In 1996, Gaubert and Giua explored Petri Net languages with infinite sets of final markings [14] and further extended their research in 1999 to investigate the infinite subsets within these sets [15]. In 2005, Valk et al. studied

✉ R. Arulprakasam
r.aruljeeva@gmail.com

A. Mahadeer
ma1637@srmist.edu.in

V. R. Dare
rajkumardare@yahoo.com

[1] Department of Mathematics, College of Engineering and Technology, SRM Institute of Science and Technology, Kattankulathur 603203, Tamilnadu, India

[2] Department of Mathematics, Madras Christian College, Tambaram, Chennai 600 059, Tamilnadu, India

the rationality of Petri Net languages [16] and in 2006, minimal representations of Petri net languages were studied by Sreenivas R. S. [17]. Kunimochi Y. investigated the algebraic properties of Petri Net languages and codes in 2009 [18]. In 1974, Fischer and Paterson [19] introduced partial words, which are defined as strings that include “do not care” symbols. In 1999, Berstel and Boasson [20] inaugurated the investigation into “Combinatorics on partial words”. This was further studied by Blanchet-Sadri [21, 22]. Periodicity in partial words and primitive partial words was investigated by Blanchet-Sadri [23, 24]. Certain properties of partial words was studied in [25]. Blanchet-Sadri [26] initiated the study of partial languages by presenting the notion of pcodes, defined as sets of partial words that maintain the uniqueness of factorization. Sasikala et al. [27] investigated the study of regular partial languages and local partial languages. Dassow et al. [28] presented a relationship between regular languages and partial words. Blanchet-Sadri et al. [29] explored questions posed by Dassow et al. regarding the relationship between the sizes of these structures, while Dassow et al. [30] investigated the regular languages of partial words. Sasikala et al. [31] introduced the Partial Array Token Petri Net to generate partial array languages.

Moreover, Mary Ann et al. proposed a bio-model engineering framework using Petri Nets [32]. Recent advancements, such as the improved database concurrency control algorithm [33] and the wavelet neural network model for intrusion detection [34], have significantly contributed to diverse computing tools and applications [35]. The exploration of secure keyword search in cloud computing has also been notable [36]. Muhammad Rizwan Ali et al. proposed cloud computing modeling and analysis based on Petri Net [37]. A timed neural network for the shortest path problem was investigated [38]. Toeplitz matrices-based key exchange protocols for the Internet of Things have been proposed [39].

The signifcant contributions of this research are:

- To understand the behavior of a system, analyzing its execution is essential. This approach is more useful and efficient in comprehending the system’s behavior.
- We study partial languages generated from the Partial Petri Net.
- We investigate closure properties over partial languages to provide a precise technique for verification, validation and synthesis for the important class of uncertain distributed systems.
- In existing literature, partial array languages have been studied with the introduction of Partial Array Token Petri Net [31], focusing on partial array languages. No prior work has been found to directly address this study. Notably, our study is specifically focused on the partial languages derived from the Partial Petri Net.
- The Partial Petri Net model established here is useful to enhance the modeling and analysis of concurrent, distributed systems characterized by uncertainty or partial observability of process executions. This potential application of the Partial Petri Net opens up new possibilities for research and exploration in the field.

The paper is organized as follows. Section 2 presents the basic definitions used in this paper. The Partial Petri Net is defined and its execution, state space and labelings, including an algorithm, are discussed in Sect. 3. In Sect. 4, Partial Petri Net languages and closure properties of these languages are studied. Finally, Sect. 5 gives the conclusion along with future scope.

2 Preliminaries

In this section, we have given some basic definitions and notations of partial words, partial languages and Petri Net.

Let Σ be a finite set of symbols or letters that is not empty. A sequence of letters from Σ is referred to as a string or word over Σ. The empty word is represented by the symbol λ. All strings that are made up of letters from Σ, including λ, are represented by Σ^*. On the other hand, Σ^+ represents the set of all strings in Σ, but excludes λ. A language L is defined as any subset of Σ^*. A finite partial word (or partial word) is a sequence of symbols that has a number of unspecified symbols, denoted by $\diamond$, that are called “holes” or “do not care symbols”. Consider a partial word $u = u[1...n]$ over the alphabet Σ. u is a partial function that maps the set $\{1, 2, ..., n\}$ to Σ. If $u(i)$ is defined for any i where $1 \leq i < n$, then i is part of the domain of u, represented as $D(u)$. If not, i is included in the set of holes of u, represented as $H(u)$. To represent the positions of the holes in u, we define the partial word $u_\diamond$ (Eq. 1) to be the total function $u_\diamond : \{1, 2, ..., n\} \to \Sigma_\diamond$ that maps to the extended alphabet $\Sigma_\diamond = \Sigma \cup \{\diamond\}$ and $\diamond \notin \Sigma$, such that:

$$u_\diamond(i) = \begin{cases} u(i) & \text{if} \quad i \in D(u) \\ \diamond & \text{if} \quad i \in H(u). \end{cases} \tag{1}$$

The set of all finite partial words over $\Sigma_\diamond$ is denoted by ${\Sigma_\diamond}^*$. Within this, ${\Sigma_\diamond}^+$ denotes the subset containing all non-empty partial words over $\Sigma_\diamond$. A partial language $L_\diamond$ is then defined as any subset $L_\diamond \subseteq {\Sigma_\diamond}^*$, that is, a set of partial words over the extended alphabet $\Sigma_\diamond$.

A Petri Net structure, denoted as C, can be expressed as a quintuple: $C = (P, T, I, O, \mu_0)$. Here, P represents a set of finite places, and T represents a finite transitions set. The intersection of the sets of transitions and places is empty, indicated by $P \cap T = \emptyset$. The input function, denoted as $I : T \to P^\infty$, maps transitions to bags of places, while the output function, denoted as $O : T \to P^\infty$, performs a similar mapping. The initial marking, represented by $\mu_0 : P \to \{0, 1, 2, \dots\}$, defines the initial state of the places in the Petri Net. if $p_i \in I(t_j)$ then a place p_i serves as an input place of an transition t_j. Conversely if $p_i \in O(t_j)$, p_i acts as an output place. The inputs and outputs associated with a transition are bags of places. A bag is a concept that builds upon the notion of a set. Similar to a set, a bag constitutes a group of elements from a certain domain. However, in contrast to a set, the elements in a bag can recur multiple times. A function, denoted as $\#(\cdot, \cdot)$, is represented for components in a given domain and bags within that domain, this determines the frequency of each component in the bag. Specifically, $\#(x, \beta) = k \geq 0$ indicates that element x appears k times exactly in the bag β. Since set theory is a subset of bag theory when the range of the # function is $\{0, 1\}$, we utilize a significant amount of the symbols and fundamental concepts of sets when dealing with bags. A Labeled Petri Net, denoted as $A = (C, \Sigma, \sigma)$ where $C = (P, T, I, O, \mu_0)$, Σ is the input alphabet and a labeling function $\sigma : T \rightarrow \Sigma$. The labeling function extends to firing sequences, where if γ represents a firing sequence, $\sigma(\gamma)$ is termed a label sequence. The languages generated by Petri Nets consist of the sets of label sequences corresponding to all firing sequences or just all terminal firing sequences.

3 Partial Petri Net

In this section, we define Partial Petri Net with an example and discuss its execution, state space and labelings, including an algorithm.

Definition 1 A Partial Petri Net, $PN_\diamond$, is a quintuple defined by

$$PN_\diamond = (P, T_\diamond, \Sigma_\diamond, S, F),$$

where

P is a finite set of places,
$T_\diamond = T_r \cup T_p$ and $T_r \cap T_p = \phi$, here
T_r is a finite set of regular transitions and T_p is a finite set of partial transitions,
$\Sigma_\diamond$ is the input alphabet,
$S \in P$ is the start place,
$F \subseteq P$ is the finite set of final places.

Every transition, denoted as τ_j from the set $T_\diamond$, is defined by an ordered triple, expressed as

$$\tau_j = (\alpha_j, \mathcal{I}_j, \mathcal{O}_j)$$

where

$\alpha_j \in \Sigma_\diamond$ representing the label for τ_j,
$\mathcal{I}_j \subseteq P$ denotes the input places of the transition,
$\mathcal{O}_j \subseteq P$ denotes the output places of the transition.

Example 1 Consider the $PN_\diamond = (P, T_\diamond, \Sigma_\diamond, S, F)$, where $P = \{p_1, p_2, p_3\}$, $T_\diamond = \{\tau_{r1}, \tau_{r2}, \tau_{r3}, \tau_p\}$, $\Sigma_\diamond = \{a, b, c\} \cup \{\diamond\}$, $S = \{p_1\}$ and $F = \{p_3\}$.

When dealing with Partial Petri Nets, it is necessary to mention the individual elements of the ordered triples that constitute transitions. To aid in the precise recognition of these elements within a transition, we present three mapping functions (Eq. 2): the label function denoted by (α), the input function represented by $(\mathcal{I})$ and the output function represented by $(\mathcal{O})$. For a transition $\tau_j = (\alpha_j, \mathcal{I}_j, \mathcal{O}_j)$, where $\tau_j \in T_\diamond$, the definitions of these functions are as follows:

$$\alpha(\tau_j) = \alpha_j, \mathcal{I}(\tau_j) = \mathcal{I}_j, \mathcal{O}(\tau_j) = \mathcal{O}_j \tag{2}$$

To map transition sequences to label sequences, we expand the label function by

$$\alpha(y) = \begin{cases} \epsilon & \text{if} \quad y = \epsilon \\ \alpha(\tau_j)\alpha(x) & \text{if} \quad y = \tau_j x, \tau_j \in T_\diamond, x \in {T_\diamond}^* \end{cases} \tag{3}$$

3.1 Execution rules

The Definition (1) focus on describing the structural characteristics of a $PN_\diamond$. As an abstract machine, the $PN_\diamond$ exhibits computational properties that govern its behaviour during firing. The firing process of a $PN_\diamond$ is influenced by the presence and positioning of tokens, which are represented as black dots and traverse according to the firing rules for $PN_\diamond$ Fig. 1). These rules can be summarized as follows:

- Initialization: Initiate the $PN_\diamond$ by placing a single token in the start state (Fig. 2).
- Final State Check: If the net reaches a terminal state, the execution may halt; otherwise, compute the firing transition set, V (Fig. 3).
- Transition Firing: If V is nonempty, initiate the firing of one transition from V and subsequently revert to step second step (Fig. 4). If V is empty, the firing comes to a halt (Fig. 5).

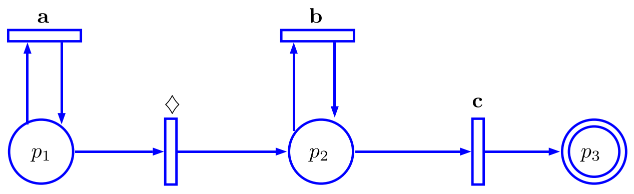


Fig. 1 An example of the $PN_\diamond$

When there are enough tokens in each of a transition's input places, the transition becomes enabled. Tokens are taken out of all of a transition's input places and deposited into all of its output places when the transition is fired. These definitions are clarified further by

Definition 2 A transition, $\tau_j \in T_\diamond$, is enabled if for every $p_n \in P$, the number of tokens in p_n is atleast $\#(p_n, \mathcal{I}(\tau_j))$.

Definition 3 An firable transition, τ_j, initiates by first eliminating the token $\#(p_n, \mathcal{I}(\tau_j))$ from every place $p_n \in P$ and proceeds by inserting the token $\#(p_n, \mathcal{O}(\tau_j))$ to each place $p_n \in P$.

3.2 The state space

Token distribution and number inside the net define the structure of a $PN_\diamond$. Alternatively, this can be written as the cardinality of tokens, permitting zero, at every place within the $PN_\diamond$, which is known as a marking. An n-vector of non-negative numbers represents the place of a $PN_\diamond$ and the token cardinality in each place is always a non-negative integer. A change in the place of $PN_\diamond$ is indicated by the firing of a transition. If there is a sequence of firings that can change the initial state-which corresponds to one token at the beginning and no tokens everywhere else into the target state, then that state is said to be reachable. We define M as the reachable state space, which is also commonly known as the marking of a $PN_\diamond$.

If we represent the set of non-negative integers as N, then $M \subseteq N^n$. Every element in M is structured as an $n-$vector, with its $\mathfrak{K}^{th}$ element indicating the tokens cardinality at place p_n where $1 \leq \mathfrak{K} \leq n$). The symbol S is used to denote the initial state and the vector $(1, 0, 0, \dots)$, whereas F represents the final state set and the vector $(0, 0, \dots, 1)$.

The subsequent-state function, represented as δ, is a function mapping from $N^n \rightarrow T_\diamond$ into N^n. For a m and a $\tau_j \in T_\diamond$, the subsequent-state function, denoted as $\delta(m, \tau_j)$, is defined iff for all n,$(1 \leq n \leq k)$,

$$m_n \geq \#(p_n, I(\tau_j)) \tag{4}$$

Therefore, $\tau_j \in T_\diamond$ is considered firable in a state m iff $\delta(m, \tau_j)$ is defined. If $\delta(m, \tau_j)$ is defined, the resulting state vector from firing τ_j is determined. The n^{th} element of the subsequent state is

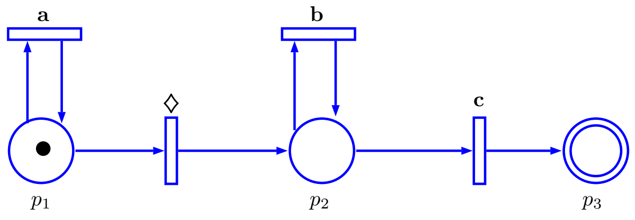


Fig. 2 Initially, place p_1 contains one token and the set of tokens is denoted as $V = \tau_{r1}, \tau_p$

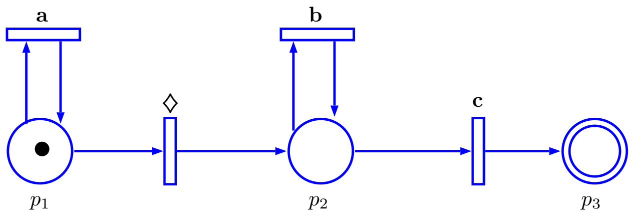


Fig. 3 After firing τ_{r1}, then $V = \{\tau_{r1}, \tau_p\}$

$$\delta(m, \tau_j)_n = m_n - \#(p_n, \mathcal{I}(\tau_j)) + \#(p_n, \mathcal{O}(\tau_j)) \tag{5}$$

If $\delta(m, \tau_j)$ is defined and Eq. (4), with $\#(p_n, \mathcal{I}(\tau_j)) \geq 0$, it follows that $\delta(m, \tau_j) \geq 0$, ensuring $\delta(m, \tau_j) \in N^n$.

The definition of $\delta(m, \tau_j)$ can be reformulated as a system of vector replacement [40]. For every $\tau_j \in T_\diamond$, we define u_j and v_j, where $(u_j)_n = -(p_n, \mathcal{I}(\tau_j))$ and $(v_j)_n = -\#(p_n, \mathcal{I}(\tau_j)) + \#(p_n, \mathcal{O}(\tau_j))$. $\delta(m, \tau_j)$ is defined if $m + u_j \geq 0$, and if $\delta(m, \tau_j)$ is defined, then $\delta(m, \tau_j) = m + v_j$. The reachable state space of the $PN_\diamond$ is analogous to the reachability set of a vector replacement system.

We expand the subsequent-state function from a single transition to the transition sequence, similar to the label function (Eq. 3). If y represents a transition sequence, denoted as $y \in {T_\diamond}^*$, then

$$\delta(m, y) = \begin{cases} m & \text{if} \quad y = \varepsilon \\ \delta(\delta(m, \tau_j), x) & \text{if} \quad y = \tau_j x, \tau_j \in T_\diamond, x \in {T_\diamond}^* \end{cases} \tag{6}$$

Certainly, the subsequent-state functions mentioned in the Eq. (5) are defined for their corresponding inputs if and only if Eq. (6) is defined.

We can now define the smallest subset of N^n defined by the reachable state space, M.

- $S \in M$
- if $m \in M$, and Eq. (6) is defined for $y \in {T_\diamond}^*$, then Eq. (6) $\in M$

We constrain the subsequent-state function to the reachable set M, since we are only focus in reachable states. As a result, $\delta : M \times T_\diamond \rightarrow M$, and with the possible exception of the initial state, this mapping is surjective.

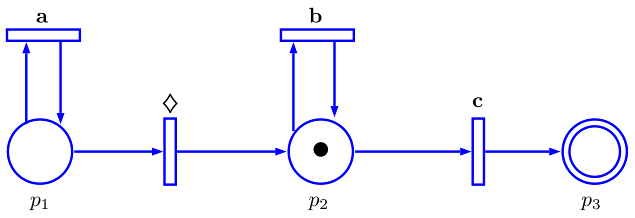


Fig. 4 After firing τ_p, then $V = \{\tau_{r2}, \tau_{r3}\}$

3.3 Labelings

A distinct labeling of the $PN_\diamond$ means that each transition possesses a unique label [that is, if $\alpha(\tau_i) = \alpha(\tau_j)$, then $\tau_i = \tau_j$]. On the other hand, the λ−free Partial Petri Net Language class permits transitions to have the same labels, but it does not allow transitions to be empty, (i.e)., for every $\tau_j \in T_\diamond : \alpha(\tau_j) \neq \lambda$. Additionally, a more inclusive labelling function has been considered, permitting null-labelled transitions, denoted as $\alpha(\tau_j) = \lambda$. These transitions are not present in the language and their existence in the $PN_\diamond$ execution is not recorded. Furthermore, the deterministic labeling has an additional property. For each marking and each label, only one transition with that label is eligible for firing. (i.e)., for every m_i and for every $\tau_j, \tau_j^{'} \in T_\diamond : (\alpha(\tau_j) = \alpha(\tau_j^{'})$ and $m_i(\tau_j)$ and $m_i(\tau_j^{'})) \Longrightarrow \tau_j = \tau_j^{'}$. These various types of labelings, including distinct (d), λ−free, λ−transitions (λ) and deterministic (def), yield four distinct languages types.

Algorithm 1 Partial Petri Net Execution and Language Study

```
Input: Place a single token in the initial place of $PN_\diamond$
Output: After reaching the terminal state, study the partial languages associated with $PN_\diamond$.
procedure PartialPetriNet($PN_\diamond$)
    Initialization:
        $M \leftarrow \{S\}$
        $V \leftarrow$ {Compute initial firing transitions}
    while $M$ is not empty do
        if $V$ is empty then
            return "No more firable transitions. Execution halted."
        end if
        for each $m$ in $M$ do
            if $m$ is in $F$ then
                return "Final state reached. Execution halted."
            end if
            for each $\tau$ in $V$ do
                if $\tau$ is enabled in $m$ then
                    Compute $\delta(m, \tau)$
                    if $\delta(m, \tau)$ is defined then
                        Add $\delta(m, \tau)$ to $M$
                        Remove $m$ from $M$
                    end if
                end if
            end for
        end for
    end while
    for each $\tau$ in $T$ do
        if $\tau$ has a distinct label then

        else if $\tau$ has a $\lambda$-free label then

        else if $\tau$ has a $\lambda$-transition label then

        else if $\tau$ has a deterministic label then

        end if
    end for
    for each $m$ in $M$ do

    end for
end procedure
```

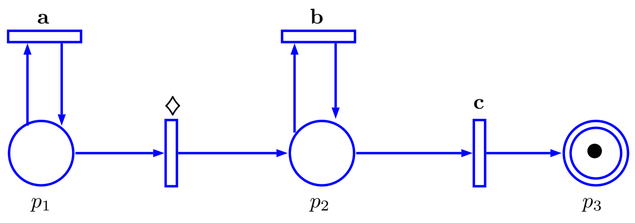


Fig. 5 After firing τ_{r3}, then $V = \{\phi\}$

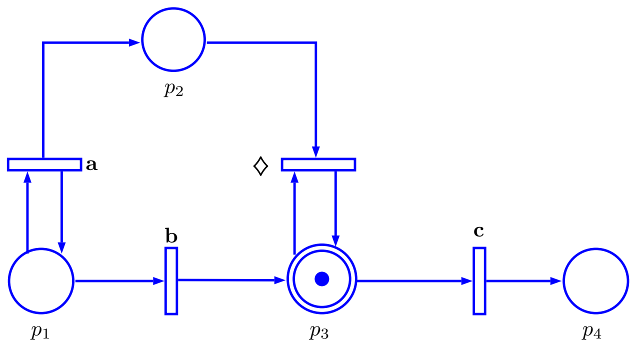


Fig. 6 An example $PN_\diamond$ to illustrate the different partial languages

Table 1 Four different PPNLs

Language Type	Language
$\mathcal{L}_\diamond$–type	$L_\diamond = \{a^n b \diamond^n \mid n \geq 0\}$
$\mathcal{G}_\diamond$–type	$L_\diamond = \{a^m b \diamond^n \mid m \geq n \geq 0\}$
$\mathcal{P}_\diamond$–type	$L_\diamond = \{a^m b \diamond^n c \mid m \geq n \geq 0\}$
$\mathcal{R}_\diamond$–type	$L_\diamond = \{a^n \mid m \geq 0\}$ $\cup \{a^m b \diamond^n \mid m \geq n \geq 0\} \cup \{a^m b \diamond^n c \mid m \geq n \geq 0\}$

4 Partial Petri Net Languages and its closure properties

This section defines Partial Petri Net Language classes and studies their closure properties.

Definition 4 A partial language $L_\diamond \in \Sigma_\diamond{}^*$ is said to be an $\mathcal{L}_\diamond$-type Partial Petri Net Language (PPNL) if there exists a $PN_\diamond$ such that $L_\diamond = \{\alpha(y) \in \Sigma_\diamond{}^* \mid y \in T_\diamond{}^* \ \&\ \delta(m, y) \in F\}$.

Definition 5 A partial language $L_\diamond \in \Sigma_\diamond{}^*$ is said to be an $\mathcal{G}_\diamond$-type PPNL if there exists a $PN_\diamond$ such that $L_\diamond = \{\alpha(y) \in \Sigma_\diamond{}^* \mid y \in T_\diamond{}^*, \exists\, m_f \in F{:}\ \delta(m, y) \geq m_f\}$.

Definition 6 A partial language $L_\diamond \in \Sigma_\diamond^*$ is said to be an $\mathcal{P}_\diamond$-type PPNL if there exists a $PN_\diamond$ such that $L_\diamond = \{\alpha(y) \in \Sigma_\diamond{}^* \mid y \in T_\diamond{}^* \ \&\ \exists\ \delta(m, y)$ but $\forall\ \tau_j \in T_\diamond,\ \nexists\ \delta(\delta(m, y), \tau_j)\}$.

Table 2 Different Classes of PPNLs

Language/label	distinct	deterministic	λ–free	λ–transitions
$\mathcal{L}_\diamond$–type	$\mathcal{L}_\diamond{}^d$	$\mathcal{L}_\diamond{}^{def}$	$\mathcal{L}_\diamond$	$\mathcal{L}_\diamond{}^\lambda$
$\mathcal{G}_\diamond$–type	$\mathcal{G}_\diamond{}^d$	$\mathcal{G}_\diamond{}^{def}$	$\mathcal{G}_\diamond$	$\mathcal{G}_\diamond{}^\lambda$
$\mathcal{P}_\diamond$–type	$\mathcal{P}_\diamond{}^d$	$\mathcal{P}_\diamond{}^{def}$	$\mathcal{P}_\diamond$	$\mathcal{P}_\diamond{}^\lambda$
$\mathcal{R}_\diamond$–type	$\mathcal{R}_\diamond{}^d$	$\mathcal{R}_\diamond{}^{def}$	$\mathcal{R}_\diamond$	$\mathcal{R}_\diamond{}^\lambda$

Definition 7 A partial language $L_\diamond \in \Sigma_\diamond^*$ is said to be an $\mathcal{R}_\diamond$-type PPNL if there exists a $PN_\diamond$ such that $L_\diamond = \{\alpha(y) \in \Sigma_\diamond{}^* \mid y \in T_\diamond{}^* \ \&\ \exists\, \delta(m, y)\}$.

Example 2 Consider the $PN_\diamond = (P, T_\diamond, \Sigma_\diamond, S, F)$, where $P = \{p_1, p_2, p_3, p_4\}$, $T_\diamond = \{\tau_{r1}, \tau_{r2}, \tau_{r3}, \tau_p\}$, $\Sigma_\diamond = \{a, b, c\} \cup \{\diamond\}$, $S = \{p_1\}$ and $F = \{p_3\}$.

The languages generated by the $PN_\diamond$ in Fig. (6) are shown in Table (1). In addition to the four types of partial languages (definitions 4, 5, 6 and 7) defined by differences in the final state set, further classifications arise from variations through labelings (Table 2).

Although the definitions vary, the classes of PPNLs exhibit a close relationship. Specifically, the distinct labelings set is contained within the set of deterministic labelings, which in turn, is contained within the λ–free labelings set and this set is also contained within the λ–labelings set (Eqs. 7, 8, 9 and 10).

$$\mathcal{L}_\diamond^d \subseteq \mathcal{L}_\diamond^{def} \subseteq \mathcal{L}_\diamond \subseteq \mathcal{L}_\diamond^\lambda \tag{7}$$

$$\mathcal{G}_\diamond{}^d \subseteq \mathcal{G}_\diamond{}^{def} \subseteq \mathcal{G}_\diamond \subseteq \mathcal{G}_\diamond{}^\lambda \tag{8}$$

$$\mathcal{P}_\diamond{}^d \subseteq \mathcal{P}_\diamond{}^{def} \subseteq \mathcal{P}_\diamond \subseteq \mathcal{P}_\diamond{}^\lambda \tag{9}$$

$$\mathcal{R}_\diamond{}^d \subseteq \mathcal{R}_\diamond{}^{def} \subseteq \mathcal{R}_\diamond \subseteq \mathcal{R}_\diamond{}^\lambda \tag{10}$$

Our investigation is confined to a specific set of standard form Partial Petri Nets, even though our interest extends to the entire class of $\mathcal{L}_\diamond$–type PPNLs. This limitation narrows the class of PPNLs without making the proofs and constructs more difficult. Every $PN_\diamond$ language can be generated by multiple Partial Petri Nets, however we only work with nets which have specific characteristics. We show that a standard form of $PN_\diamond$ exists, which generates a language of $\mathcal{L}_\diamond$–type for every PPNL, proving that this is not reducing the language set. First, we define the $PN_\diamond$ in standard form.

Definition 8 A Partial Petri Net, $PN_\diamond = (P, T_\diamond, \Sigma_\diamond, S, F)$ is in standard form if

1. $\mathcal{I}(\tau_j) \neq \phi$ and $\mathcal{O}(\tau_j) \neq \phi \; \forall \; \tau_j \in T_\diamond$.
2. $S \notin \mathcal{O}(\tau_j) \; \forall \; \tau_j \in T_\diamond$
3. $\exists \; p_f \in P$ such that
   - $F = p_f$ (if $\lambda \notin L_\diamond(PN_\diamond)$) or $F = \{p_f, S\}$ (if $\lambda \in L_\diamond(PN_\diamond)$),
   - $p_f \notin \mathcal{I}(\tau_j) \; \forall \; \tau_j \in T_\diamond$
   - For any $\tau_j \in T_\diamond$ and $m \in M$ with a token in p_f (i.e., $m_f > 0$)), $\delta(m, \tau_j)$ is undefined.

The standard form for Partial Petri Nets requires transitions to possess both non-empty input bags and output bags. Additionally, this standard formulation necessitates the existence of two special places: a separate final place that connects to no transition inputs, as well as a single start place that does not defined as the output for any transitions.

When executed, the $PN_\diamond$ in standard form initiates with a solitary token placed in the starting place. This token gets removed upon the firing of the first transition, resulting in an empty start place thereafter. A token reaching the designated final place can halt the $PN_\diamond$, as no transitions accept input from the final place. As a result, the token remains in the final place. To demonstrate standard-form Partial Petri Nets possess equivalent capability as the generalized form, we provide the subsequent theorem.

Theorem 1 *For any $PN_\diamond$, there exists an equivalent $PN_\diamond$ in standard form.*

Proof Consider a $PN_\diamond = (P, T_\diamond, \Sigma_\diamond, S, F)$. We illustrate the construction of an equivalent $PN_\diamond{}' = (P', T_\diamond{}', \Sigma_\diamond, S', F')$ in standard form (Fig. 7). First, we introduce three additional places that are distinct from P: p_r, S', and p_f. The starting place is S', the terminal place is p_f and the run place is p_r. Tokens in p_r are required for the firing of any transition in $T_\diamond$. One token will be present in S' for the initial marking of $PN_\diamond{}'$ and one token will be present in p_f [for $\lambda \in L_\diamond(PN_\diamond)$] or S' [if $\lambda \notin L_\diamond(PN_\diamond)$].

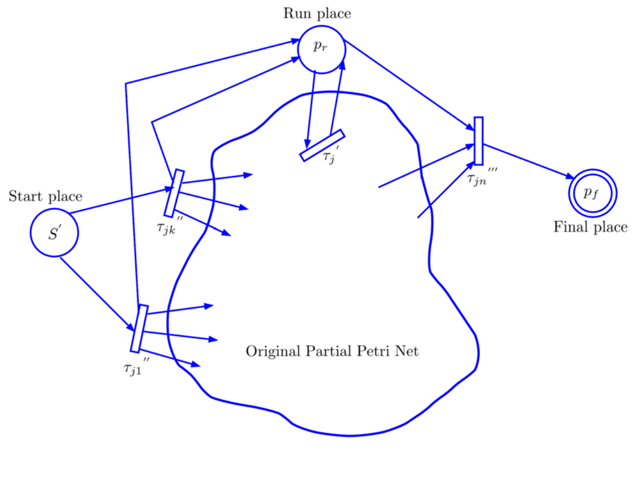


Fig. 7 The built structure of a $PN_\diamond$ in its standard form

At this point, we have to make sure that all of the transition sequences in $PN_\diamond$ that move from the starting marking to the terminal marking are also in $PN_\diamond{}'$ respectively. In order to do this, we examine three different kinds of strings in $L_\diamond(PN_\diamond)$. First, F' is defined in a way that appropriately recognizes the empty string λ. By examining if the starting marking is the terminal marking $m \in F$, we can determine whether $\lambda \in L_\diamond(PN_\diamond)$.

Second, we consider a particular transition from S' to p_f in $PN_\diamond{}'$ for all strings of length 1 in $L_\diamond(PN_\diamond)$, as follows: Define $\tau_\sigma \in T_\diamond{}'$ for $\sigma \in \Sigma$ with $\sigma \in L_\diamond(PN_\diamond)$, where $\mathcal{I}(\tau_\sigma) = \{S'\}$ and $\mathcal{O}(t_\sigma) = \{p_f\}$. Label σ is associated with τ_σ. By examining each transition $\tau_j \in T_\diamond$, with $\alpha(\tau_j) = \sigma$, we can determine that $\sigma \in L_\diamond(PN_\diamond)$. Also, we can determine that $\delta(m, \tau_j) \in F$. Lastly, take into consideration all strings longer than 1. These strings forms a sequence in $T_\diamond$, denoted by $\tau_{j_1}, \tau_{j_2}, \ldots, \tau_{j_n}$. We then define a sequence incorporating new transitions a and b: $a\tau_{j1}{}', \ldots, \tau_{jn}{}'b$. To generate the initial marking m of $PN_\diamond$ and a token in p_r, the transition a would require a token from S'. With the exception of p_r, which functions as both an input and an output, each $\tau_j{}' \in T_\diamond{}'$ is identical to $\tau_j \in T_\diamond$. This allows us to remove the token in p_r, hence disabling all transitions in $T_\diamond{}'$. At last, the b transition would output a token to p_f and eliminate the token from p_r along with a terminal marking of $PN_\diamond$. A sequence $a\tau_{j1}{}', \ldots, \tau_{jn}{}'b$ would be the only way for the token in the initial to the terminal places in $PN_\diamond{}'$ under this construction. This sequence aligns with $\tau_{j_1}, \tau_{j_2}, \ldots, \tau_{j_n}$ leading from m to a terminal marking in $PN_\diamond$.

Incorporating the additional symbols for transitions a and b would make the sequence overly long, since those symbols would only apply to $PN_\diamond{}'$ and not the $PN_\diamond$. A null labeling for a and b would be one way to solve this, but null labelings are not allowed in $\mathcal{L}_\diamond$–type languages. To resolve this, we need to merge transitions b and $\tau_{j1}{}'$ into a single transition denoted $\tau_{j1}{}'''$, as well as merge transitions a and $\tau_{j1}{}'$ into $\tau_{j1}{}''$. Therefore, we define the combined transitions in $T_\diamond{}'$, corresponding to any $\tau_j \in T_\diamond$:

- Define $\tau_j{}' \in T_\diamond{}'$, where $\mathcal{O}(\tau_j{}') = \mathcal{O}(\tau_j) \cup \{p_r\}$ and $\mathcal{I}(\tau_j{}') = \mathcal{I}(\tau_j) \cup \{p_r\}$.
- Define $\tau_j{}''$ with $\mathcal{I}(\tau_j{}'') = \{S'\}$ and $\mathcal{O}(\tau_j{}'') = m - \mathcal{I}(\tau_j) + \mathcal{O}(\tau_j) \cup \{p_r\}$ if $\mathcal{I}(\tau_j) \subseteq m$.
- For every m' in F representing a terminal marking that could result from τ_j firing as the terminal

transition, we define τ_j''' with $\mathcal{O}(\tau_j''') = \{p_f\}$ and $\mathcal{I}(\tau_j''') = m' - \mathcal{O}(\tau_j) + \mathcal{I}(\tau_j) \cup \{p_r\}$.

The labeling α' (Eq. 11) is now defined by

$$\alpha'(\tau_j') = \alpha''(\tau_j'') = \alpha'''(\tau_j''') = \alpha(\tau_j) \tag{11}$$

Any string σ that is in $L_\diamond(PN_\diamond)$ is generated, by $\tau_{j_1}, \tau_{j_2}, \dots, \tau_{j_n}$ such that $\sigma = \alpha(\tau_{j_1}, \tau_{j_2}, \dots, \tau_{j_n})$. By formulation

$$\sigma = \alpha(\tau_{j1}'', \tau_{j2}', \dots, \tau_{jn-1}', \tau_{jn}''') \tag{12}$$

and so Eq. (12) $\in L_\diamond(PN_\diamond)$. Therefore, given that $L_\diamond(PN_\diamond) = L_\diamond(PN_\diamond')$, $PN_\diamond$ and $PN_\diamond'$ are equivalent. □

Figure (8) shows a simple $PN_\diamond$ which is not in standard form. Applying the proof's construction to this $PN_\diamond$ results in the standard form, shown in Fig. (9).

We will delve into the closure properties of PPNL. Given two such languages, $L_{\diamond 1}$ and $L_{\diamond 2}$, we know that the $PN_\diamond$ in standard form generates each of these languages. We therefore consider two Partial Petri Nets in standard form. $PN_{\diamond 1} = (P_1, T_{\diamond 1}, \Sigma_\diamond, S_1, F_1)andPN_{\diamond 2} = (P_2, T_{\diamond 2}, \Sigma_\diamond, S_2, F_2)$ with $L_{\diamond 1} = L_\diamond(PN_{\diamond 1})$ and $L_{\diamond 2} = L_\diamond(PN_{\diamond 2})$. Given that both are in standard form, the start places of $PN_{\diamond 1}$ and $PN_{\diamond 2}$ are $S_1 \in P_1$ and $S_2 \in P_2$, respectively. Further, $F_2 = \{S_2, p_{f2}\}$ or $\{p_{f2}\}$ and $F_1 = \{S_1, p_{f1}\}$ or $\{p_{f1}\}$.

Using the given Partial Petri Nets, we demonstrate the construction of another $PN_\diamond' = (P', T_\diamond', \Sigma_\diamond, S', F')$ with a language $L_\diamond(PN_\diamond')$, representing the combination of $L_{\diamond 1}$ and $L_{\diamond 2}$.

Theorem 2 *Let $L_{\diamond 1}$ and $L_{\diamond 2}$ be two PPNLs, then $L_{\diamond 1}L_{\diamond 2}$ is also a PPNL.*

Proof Consider constructing a new $PN_\diamond'$ where the final place of $PN_{\diamond 1}$, denoted p_{f1}, is made equivalent to the start place of $PN_{\diamond 2}$, denoted S_2. This has the effect that the transition depositing a token in p_{f1} also signals the start of execution in $PN_{\diamond 2}$. Due to this construction, any string formed by concatenating elements from $L_{\diamond 1}$ and $L_{\diamond 2}$ will have a valid path in $PN_\diamond'$ from the initial place S_1 through the overlap place $p_{f1} = S_2$ to the final place p_{f2}. Hence, the concatenated string must be an element of the language $L_\diamond(PN_\diamond')$. A similar argument can show that any string generated from $PN_\diamond'$ must consist of a string from $L_{\diamond 1}$ followed by one from $L_{\diamond 2}$. To properly account for the possibility of empty strings, the

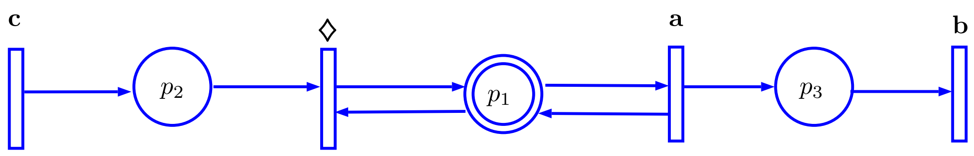


Fig. 8 A $PN_\diamond$ which is not in standard form

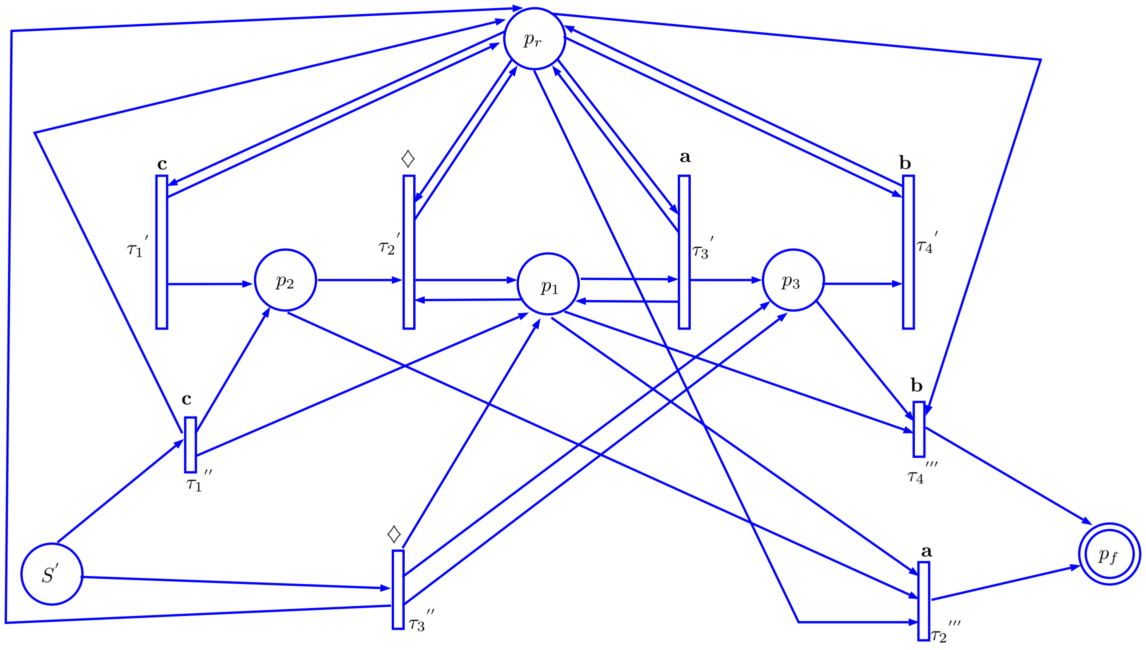
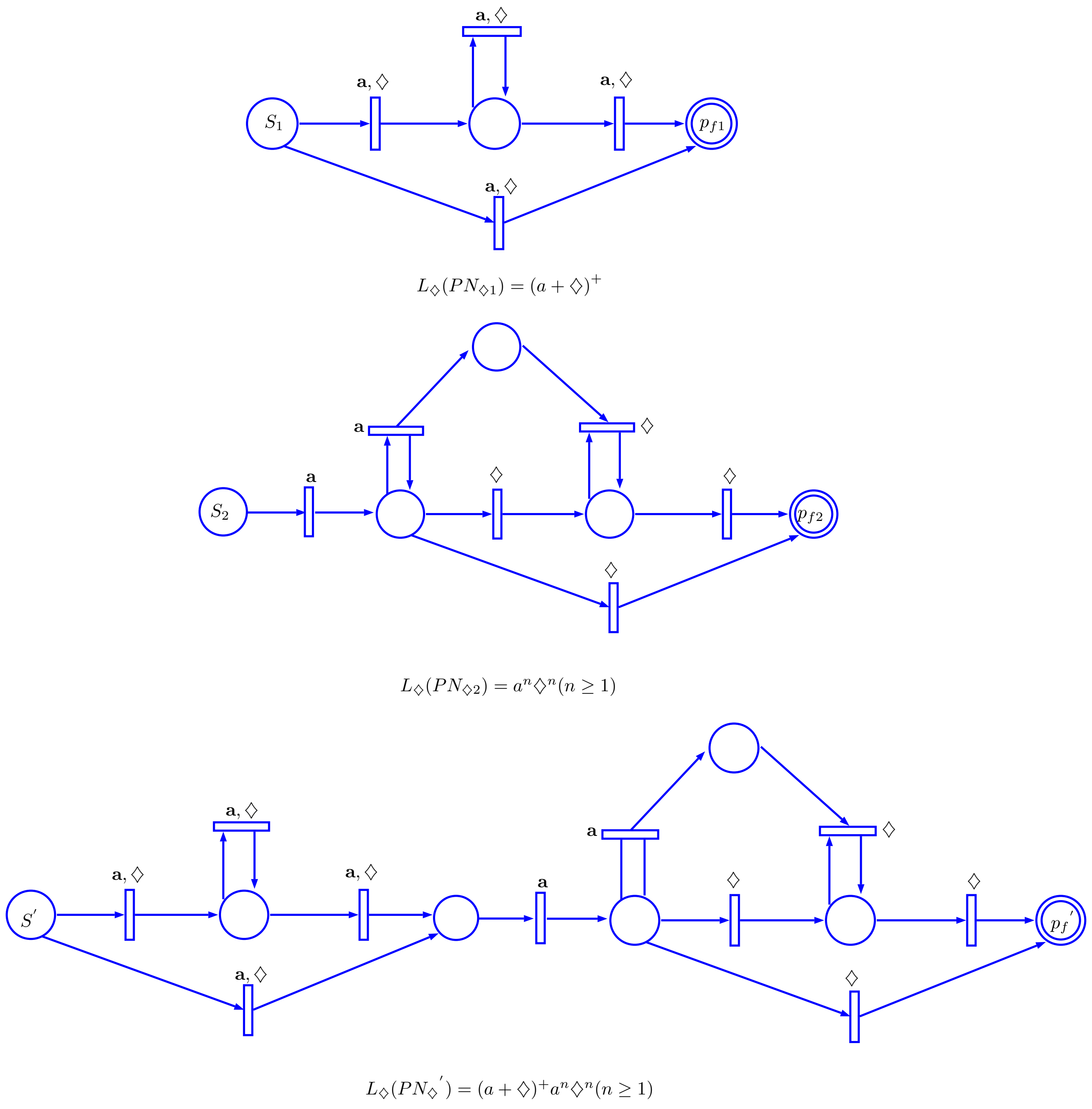


Fig. 9 A standard form of $PN_\diamond$



$\mathbf{a}, \diamondsuit$
S_1
p_{f1}
$L_{\diamondsuit}(PN_{\diamondsuit 1}) = (a + \diamondsuit)^+$
$\mathbf{a}$
$\diamondsuit$
S_2
p_{f2}
$L_{\diamondsuit}(PN_{\diamondsuit 2}) = a^n \diamondsuit^n (n \geq 1)$
S'
p_f'
$L_{\diamondsuit}(PN_{\diamondsuit}{}') = (a + \diamondsuit)^+ a^n \diamondsuit^n (n \geq 1)$

Fig. 10 An example of concatenating two Partial Petri Net Languages

formal definition of $PN_{\diamond}{}'$ requires defining it as the union of $PN_{\diamond 1}$ and $PN_{\diamond 2}$ along with some additional transitions. Specifically, for any transition τ_j in $T_{\diamond 2}$ with input place S_2, introduce a corresponding transition τ_j' in $PN_{\diamond}{}'$ with input place p_{f1} and output places $O_2(\tau_j)$. Also, if S_1 is a final place in $PN_{\diamond 1}$ add another transition τ_j'' with input place S_1 and output places from $O_2(\tau_j)$. The labeling functions α' for the new transitions $\tau' j$ and $\tau'' j$ mirror those from $PN_{\diamond 2}$. The final place set F' for $PN_{\diamond}{}'$ is defined as the union of F_1 and F_2, unless S_2' is not a final place, in which case $F' = F_2$. The construction is illustrated in Fig. (10). □

Theorem 3 *Let* $L_{\diamond 1}$ *and* $L_{\diamond 2}$ *be two PPNLs, then* $L_{\diamond 1} \cup L_{\diamond 2}$ *is also a PPNL.*

Proof The approach for proving this theorem follows a similar construction that was shown previously. Specifically, we introduce a new $PN_{\diamond}{}'$ defined such that its language satisfies $L_{\diamond}(PN_{\diamond}{}') = L_{\diamond 1} \cup L_{\diamond 2}$. This is accomplished via $PN_{\diamond}{}'$ containing a single new start place formed by combining the start places of $PN_{\diamond 1}$ and $PN_{\diamond 2}$. An initial transition in $PN_{\diamond}{}'$ removes the token from this unified start place. Subsequently, depending on whether this initial transition lies in $T_{\diamond 1}$ or $T_{\diamond 2}$, the underlying net $PN_{\diamond 1}$ or $PN_{\diamond 2}$ proceeds to execute as it normally would, just as defined previously. we introduce a new start place S' along with additional transitions $\tau_{j1}{}'$ and $\tau_{j2}{}'$ defined as follows. For each transition τ_{j1} in $T_{\diamond 1}$ that has the start place S_1 as an input place, create a corresponding transition $\tau_{j1}{}'$ with input place S' and output places identical to those of τ_{j1} in $PN_{\diamond 1}$, (i.e.) $\mathcal{O}_1(\tau_{j1})$. Similarly, for each transition τ_{j2} in $T_{\diamond 2}$ with start place S_2 as input, add transition $\tau_{j2}{}'$ with input S' and outputs $\mathcal{O}_2(\tau_{j2})$. The labeling α' map to α_1 and α_2 accordingly. The starting marking assigns a single token in S' and the terminal marking set F' is defined as the union of F_1 and F_2. Further, if $S_1 \in F_1$ or $S_2 \in F_2$, then $S' \in F'$. The construction is demonstrated in Fig. (11). □

Theorem 4 *The intersection of two PPNLs, $L_{\diamond 1}$ and $L_{\diamond 2}$, is also a PPNL.*

Proof Constructing a $PN_{\diamond}{}'$ that generates the intersection of two PPNLs poses certain difficulties. As sequences are being generated, a transition that fires in one $PN_{\diamond}$, necessitates an equivalent labeled transition to be enabled in the other and vice versa. When multiple transitions carrying the same label exist across the pairs of nets, all possible pairings between these matching transitions require consideration. For each such pair, we introduce a new transition which activates iff both transitions in the pair can fire in their respective original Partial Petri Nets. The input and output places for this new transition are defined as the sums of the input and output places of the associated pair of old transitions. Formally, if $\tau_j \in T_{\diamond 1}$ and $\tau_k \in T_{\diamond 2}$ have $\alpha(\tau_j) = \alpha(\tau_k)$, the corresponding $\tau_{j,k} \in T_{\diamond}{}'$ such that $\mathcal{I}'(\tau_{j,k}) = \mathcal{I}_1(\tau_j) + \mathcal{I}_2(\tau_k)$ and $\mathcal{O}'(\tau_{j,k}) = \mathcal{O}_1(\tau_j) + \mathcal{O}_2(\tau_k)$. Any such $\tau_{j,k}$ with combined input places forming the set S_1, S_2 are replaced with a transition $\tau_{j,k}{}'$ with the unified start place S' as its sole input. Through this merging process, the two Partial Petri Nets are effectively joined into one unified $PN_{\diamond}{}'$ capable of generating their language intersection. Figure (12) illustrates this construction. □

Theorem 5 *The concurrent of two PPNLs, $L_{\diamond 1}$ and $L_{\diamond 2}$, is also a PPNL.*

Proof Constructing $PN_{\diamond}{}'$ to generates the concurrent property of languages $L_{\diamond 1}$ and $L_{\diamond 2}$ can be formulated as follows, given Partial Petri Nets that generate $L_{\diamond 1}$ and $L_{\diamond 2}$. First, initialize the marking by inserting tokens in the start places of $PN_{\diamond 1}$ and $PN_{\diamond 2}$. Then, define the set of terminal markings of the combined net to be any marking such that the restriction of that marking to places P_1 belongs to final marking set F_1 of $PN_{\diamond 1}$ concurrently with its restriction to places P_2 belonging to final marking set F_2 of $PN_{\diamond 2}$. This construction is illustrated in Fig. (13) □

Theorem 6 *Let $L_{\diamond}$ be a PPNL, then its reverse, $L_{\diamond}{}^R$ is also a PPNL.*

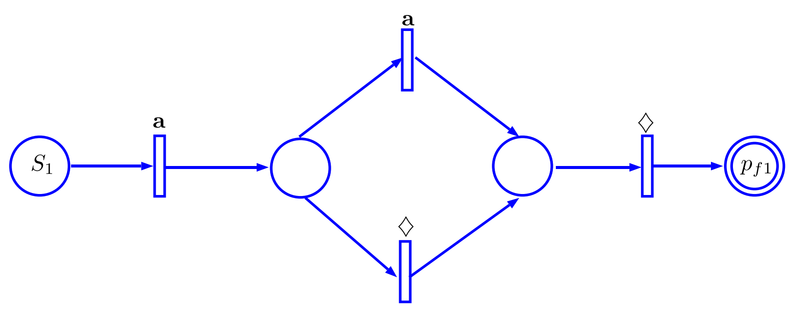
Proof The procedure for obtaining the reverse language is simple. Consider the $PN_{\diamond}$ and construct the reverse $PN_{\diamond}{}'$ by swapping the initial and final markings and similarly transposing the places for each transition. This reversal of arcs and markings causes $PN_{\diamond}{}'$ to accept exactly the reverse strings of those accepted by the $PN_{\diamond}$. Thus we have $L_{\diamond}(PN_{\diamond}{}') = L_{\diamond}(PN_{\diamond})^R$, showing that operating $PN_{\diamond}$ in reverse yields its reverse language. Consequently, this construction inverts the order of all strings generated by $PN_{\diamond}$. □

Remark 1 Closure under complementation may not hold for PPNLs.

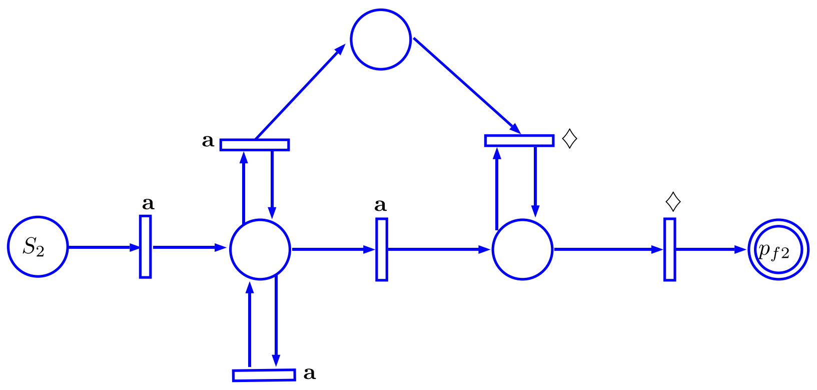
5 Conclusion and future scope

In this paper, the Partial Petri Net is defined. By modifying final state markings reached through execution sequences, we generated associated PPNLs. Further, by varying the labeling of transitions, we categorized these language classes. We demonstrated closure properties like union, intersection, concatenation, concurrency and reversal hold for these languages. While these initial results developing the formal theory of Partial Petri Nets and their languages are significant, further exploration remains.

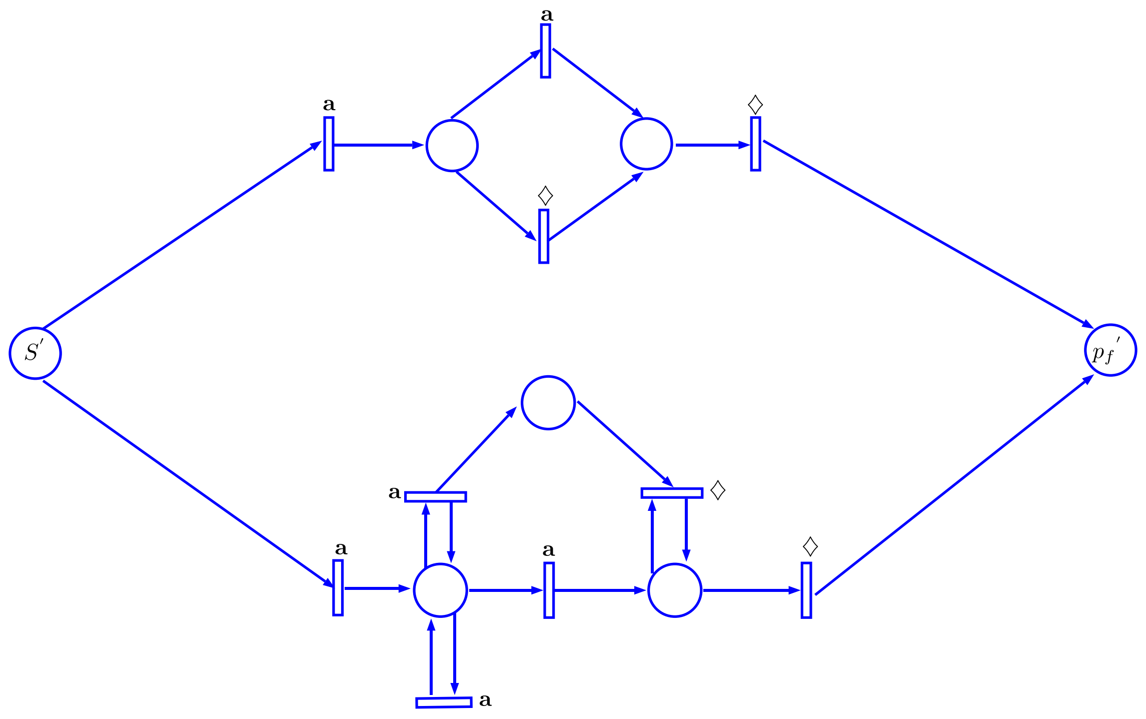
Future work includes a more extensive categorization and analysis of additional classes of PPNL beyond the $\mathcal{L}_{\diamond}-$type studied thus far. Relationships between properties of these classes of PPNL and those of regular and local partial languages also warrant research. The incorporation of place-based labeling represents an open possibility that may yield valuable new insights into the process of behavioral languages and results. There remains a rich opportunity



$$L_{\diamondsuit}(PN_{\diamondsuit 1}) = a(a + \diamondsuit)\diamondsuit$$



$$L_{\diamondsuit}(PN_{\diamondsuit 2}) = a^m \diamondsuit^n (m > n > 1)$$



$$L_{\diamondsuit}(PN_{\diamondsuit}^{'}) = a(a + \diamondsuit)\diamondsuit + a^m \diamondsuit^n (m > n > 1)$$

Fig. 11 An example of union of two Partial Petri Net Languages

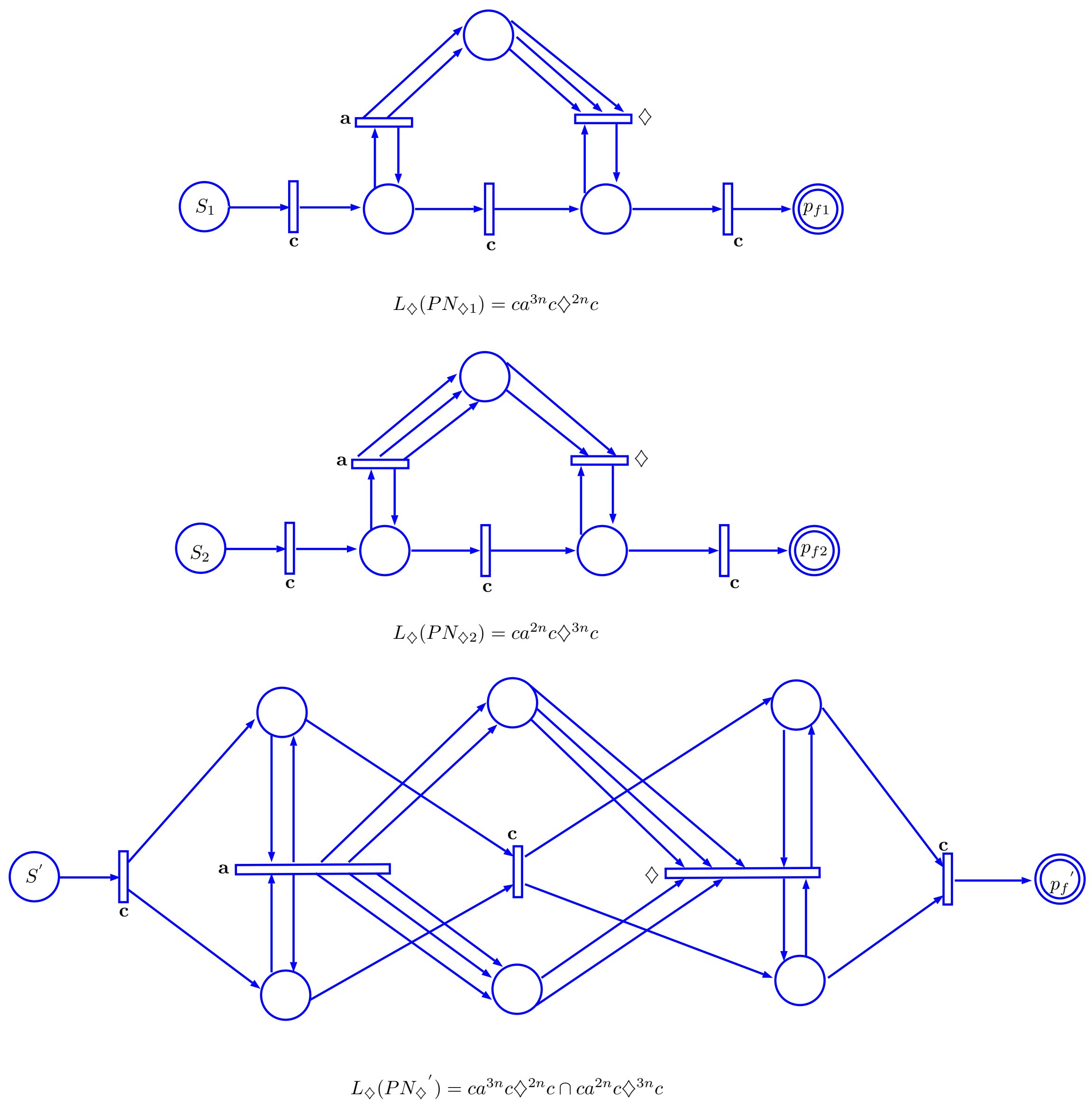
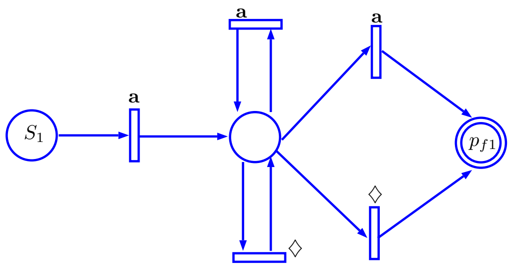
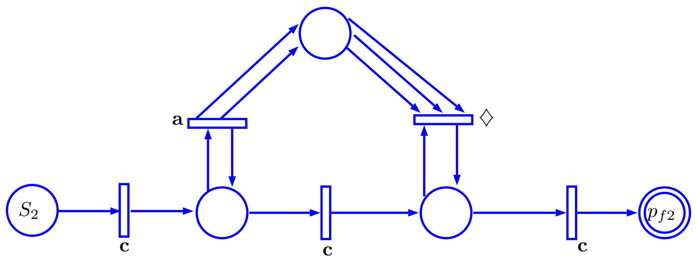

a
◇
S_1
c
c
c
p_{f1}
$L_\Diamond(PN_{\Diamond 1}) = ca^{3n}c\Diamond^{2n}c$
a
◇
S_2
c
c
c
p_{f2}
$L_\Diamond(PN_{\Diamond 2}) = ca^{2n}c\Diamond^{3n}c$
$S^{'}$
c
a
c
◇
c
$p_f^{'}$
$L_\Diamond(PN_\Diamond^{'}) = ca^{3n}c\Diamond^{2n}c \cap ca^{2n}c\Diamond^{3n}c$

Fig. 12 An example of intersection of two Partial Petri Net Languages

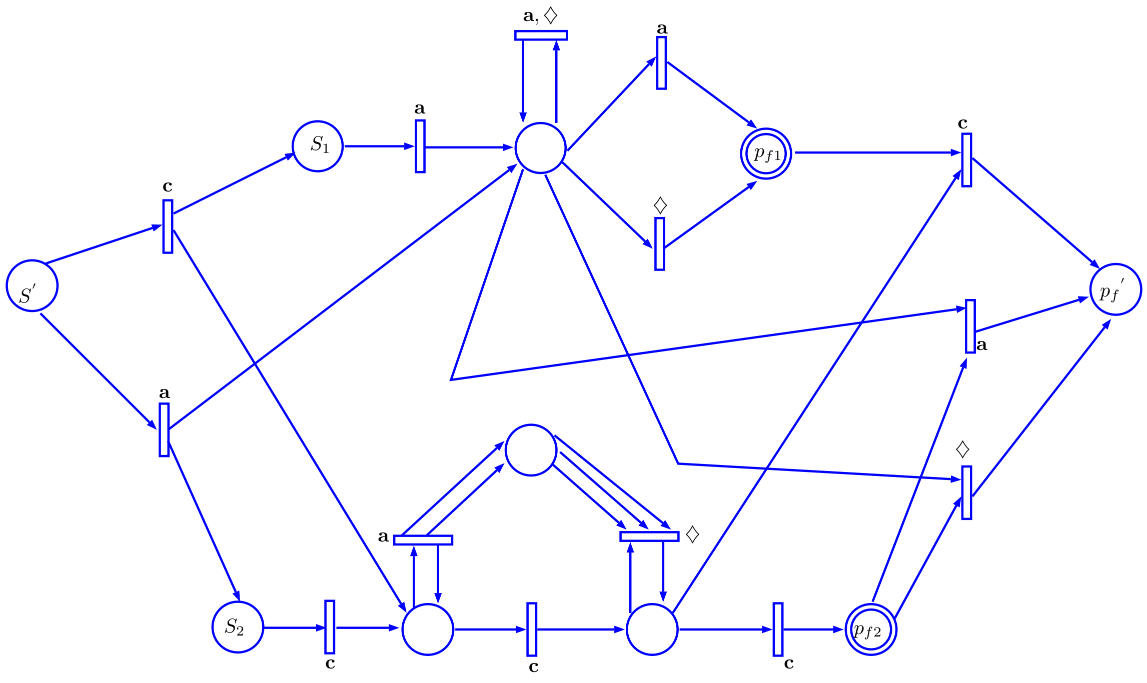
to enhance the understanding of concurrent systems using Partial Petri Nets. Moreover, future work could integrate Partial Petri Nets with bio-inspired techniques like neural networks and evolutionary algorithms for learning and optimization. Additional opportunities include improving concurrency control in databases extending intrusion detection systems formally verifying security and privacy in cloud platforms, optimizing concurrent paths with learning methods, and improving the efficiency of cloud task scheduling



$L_{\diamondsuit}(PN_{\diamondsuit 1}) = a(a+\diamondsuit)^{+}$



$L_{\diamondsuit}(PN_{\diamondsuit 2}) = ca^{3n}c\diamondsuit^{2n}c$



$L_{\diamondsuit}(PN_{\diamondsuit}{}') = a(a+\diamondsuit)^{+}||ca^{3n}c\diamondsuit^{2n}c$

Fig. 13 An example of concurrent of two Partial Petri Net Languages

algorithms. Furthermore, Partial Petri Nets could be applied to analyse emerging lightweight security protocols tailored for Internet of Things and edge computing architectures. By pursuing both theoretical advancements and practical implementations, the understanding and impact of Partial Petri Nets can continue to expand.

Data availability No datasets were generated or analysed during this study.

Declarations

Conflict of interest The authors have no Conflict of interest to declare that are relevant to the content of this article.

References

1. Peterson JL (1981) Petri net theory and the modeling of systems. Prentice Hall PTR
2. Baker H (1972) Petri nets and languages. CSG memo 68, Project MAC, Massachusetts Institute of Technology
3. Hack M (1976) Petri net languages. CSG memo 124, Project MAC, MIT
4. Peterson JL (1976) Computation sequence sets. J Comput Syst Sci 13(1):1–24
5. Starke PH (1978) Free petri net languages. In: International Symposium on Mathematical Foundations of Computer Science, pp. 506–515. Springer, Berlin, Heidelberg
6. Jantzen M (1979) On the hierarchy of petri net languages. RAIRO. Informatique théorique 13(1):19–30
7. Valk R, Vidal-Naquet G (1981) Petri nets and regular languages. J Comput Syst Sci 23(3):299–325
8. Araki T, Kagimasa T, Tokura N (1981) Relations of flow languages to petri net languages. Theoretical Comput Sci 15(1):51–75
9. Vidal-Naquet G (1982) Deterministic languages of petri nets. In: Girault C, Reisig W (eds) Application and Theory of Petri Nets. Springer, Berlin, Heidelberg, pp 198–202
10. Parigot M, Pelz E (1985) A logical approach of petri net languages. Theoretical Comput Sci 39:155–169
11. Jantzen M (1987) Language theory of petri nets. In: Petri Nets: Central Models and Their Properties: Advances in Petri Nets 1986, Part I Proceedings of an Advanced Course Bad Honnef, 8.–19. September 1986, pp. 397–412. Springer, Berlin, Heidelberg
12. Pelz E (1987) Closure properties of deterministic petri nets. In: Annual Symposium on Theoretical Aspects of Computer Science, pp. 371–382. Springer, Berlin, Heidelberg
13. Tiplea F-L (1992) Selective petri net languages. Int J Comput Math 43(1–2):61–80
14. Gaubert S, Giua A (1996) Petri net languages with infinite sets of final markings. Proc WODES 96:326–331
15. Gaubert S, Giua A (1999) Petri net languages and infinite subsets of N^m. J Comput Syst Sci 59(3):373–391
16. Valk R, Vidal G (2005) On the rationality of petri net languages. Theoretical Computer Science: 3rd Gl Conference Darmstadt. March 28–30, 1977. Springer, Berlin, Heidelberg, pp 319–328
17. Sreenivas RS (2006) On minimal representations of petri net languages. IEEE Trans Automatic Control 51(5):799–804
18. Kunimochi Y (2009) Algebraic properties of petri net languages and codes. University of Debrecen, Faculty of Informatics
19. Fischer MJ, Paterson MS (1974) String-matching and other products. Technical report, MIT, USA
20. Berstel J, Boasson L (1999) Partial words and a theorem of fine and wilf. Theoretical Comput Sci 218(1):135–141
21. Blanchet-Sadri F, Hegstrom RA (2002) Partial words and a theorem of fine and wilf revisited. Theoretical Comput Sci 270(1–2):401–419
22. Blanchet-Sadri F (2003) A periodicity result of partial words with one hole. Comput Math Appl 46(5–6):813–820
23. Blanchet-Sadri F (2004) Periodicity on partial words. Comput Math Appl 47(1):71–82
24. Blanchet-Sadri F (2005) Primitive partial words. Discrete Appl Math 148(3):195–213
25. Kumari RK, Jeyanthi L, Janaki K, Arulprakasam R, Madhusoodhanan P (2023) Properties of cover and seed of partial words. IAENG Int J Appl Math 53(3):361–366
26. Blanchet-Sadri F (2004) Codes, orderings, and partial words. Theoretical Comput Sci 329(1–3):177–202
27. Sasikala K, Dare VR, Thomas DG (2007) Learning of partial languages. Eng Lett 14(2):72–80
28. Dassow J, Manea F, Mercaş R (2012) Connecting partial words and regular languages. How the World Computes: Turing Centenary Conference and 8th Conference on Computability in Europe, CiE 2012, Cambridge, UK, June 18–23, 2012. Proceedings 8. Springer, Berlin, Heidelberg, pp 151–161
29. Balkanski E, Blanchet-Sadri F, Kilgore M, Wyatt BJ (2013) Partial word dfas. In: International Conference on Implementation and Application of Automata, pp. 36–47. Springer, Berlin, Heidelberg
30. Dassow J, Manea F, Mercaş R (2014) Regular languages of partial words. Inform Sci 268:290–304
31. Sasikala K, Sweety F, Kalyani T, Thomas DG (2020) Partial array token petri net and p system. In: International Conference on Membrane Computing, pp. 135–152. Springer, Berlin, Heidelberg
32. Blätke MA, Rohr C, Heiner M, Marwan W (2014) A petri-net-based framework for biomodel engineering. Large-Scale Networks in Engineering and Life Sciences, 317–366
33. Mohamed M, Badawy M (2019) EL-Sayed, A: An improved algorithm for database concurrency control. Int J Inf Tecnol 11(1):21–30
34. Hamid Y, Shah FA, Sugumaran M (2019) Wavelet neural network model for network intrusion detection system. Int J Inf Tecnol 11(2):251–263
35. Gedeon T (2017) Bio-inspired computing tools and applications: position paper. Int J Inf Tecnol 9(1):7–17
36. Tariq H, Agarwal P (2020) Secure keyword search using dual encryption in cloud computing. Int J Inf Tecnol 12(4):1063–1072
37. Ali MR, Ahmad F, Chaudary MH, Khan ZA, Alqahtani MA, Alqurni JS, Ullah Z, Khan WU (2021) Petri net based modeling and analysis for improved resource utilization in cloud computing. PeerJ Comput Sci 7:351
38. Krishnan R, Murugan A (2021) Timed neural network using object-based model of neurons for shortest path problem. Int J Inf Tecnol 13(5):2037–2042
39. A P, R P (2024) Toeplitz matrices based key exchange protocol for the internet of things. Int. j. inf. tecnol. **16**(1), 293–300
40. Keller RM (1972) Vector replacement systems: a formalism for modeling asynchronous systems. Technical report, Department of Electrical Engineering Computer Sciences Laboratory Princeton