Parsing Algorithms for Regulated Grammars

Sherzod Turaev, Alexander Krassovitskiy, Mohamed Othman, Mohd Hasan Selamat

***Abstract*—** Petri nets, introduced by Carl Adam Petri [12] in 1962, provide a powerful mathematical formalism for describing and analyzing the flow of information and control in concurrent systems. Petri nets can successfully be used as control mechanisms for grammars, i.e., the generative devices of formal languages. In recent papers [4], [5], [9], [16] *Petri net controlled grammars* have been introduced and investigated. It was shown that various regulated grammars such as random context, matrix, vector, valence grammars, etc., resulted from enriching context-free grammars with additional mechanisms can be unified into the Petri net formalism, i.e., a grammar and its control can be represented by a Petri net. This unification allows approaching the membership (parsing) problem in formal language theory in the new point of view: instead of a usual derivation tree, one can use *a Petri net derivation tree* in which the control mechanism is also considered as a part of the tree. In this paper, we show that the parsing problem for regulated grammars can be solved by means of Petri net derivation trees constructed using the net unfolding. Moreover, we present a parsing algorithm for the deterministic restriction of Petri net controlled grammars based on the well-known Earley parsing algorithm.

***Keywords*—** Formal languages, Regulated grammars, Petri nets, Petri net controlled grammars, Parsing algorithms

I. Introduction

It is well-known fact that context-free grammars are not able to cover all phenomena of natural and programming languages, and also with respect to other applications of sequential grammars they cannot describe all aspects. On the other hand, context-sensitive grammars are powerful enough but have bad features with respect to decidability problems which are undecidable or at least very hard. Moreover, such concepts as a derivation tree, which is an important tool for the analysis of context-free languages, cannot be transformed to context-sensitive grammars. Therefore it is a natural idea to introduce grammars which use context-free rules and have a device which controls the application of the rules. Since Abraham first defined matrix grammars, several grammars with restrictions such as programmed, random context, valence grammars, etc. have been introduced. The monograph [3] gives a summary of this approach.

Though regulated grammars preserve many context-free-like features, one cannot still construct derivation trees in which the control mechanisms used in the grammars are also expressed by elements of the derivation trees. We can similarly define derivation trees of regulated grammars as for those of context-free grammars enriching with some additional features describing the control mechanisms but it would not be a natural extension. Another solution for this problem is to concern the notion of a derivation tree itself from another point of view: instead of directed graphs, we can consider other graphs, for instance, bipartite directed graphs – *Petri nets* – which can take into consideration the grammar as well as its control.

Petri nets are graphical and mathematical modeling tools, which are widely applied to many concurrent, asynchronous, distributed, parallel, nondeterministic and stochastic systems. The papers [6], [13], [15] illustrate that Petri nets can be used in modeling phenomena appearing in different areas.

A context-free grammar and its derivation process can be described by a Petri net, called a *context-free Petri net* (a *cf Petri net* for short), where places correspond to the terminals and nonterminals, transitions are the counterpart of the productions, the tokens reflect the occurrences of symbols in the sentential form, and there is a one-to-one correspondence between the application of (sequences of) rules and the firing of (sequence of) transitions (see, [2], [4]). Therefore it is a natural idea to control the derivations in a context-free grammar by adding some features to the associated cf Petri net. In [4], [5], [9] it has been shown that by adding places and arcs which satisfy some structural requirements one can generate well-known families of languages as random context languages, valence languages, vector languages and matrix languages. Thus the control by Petri nets can be considered as a unifying approach to different types of control.

On the other hand, Petri nets can be transformed into *occurrence nets*, i.e., usually an infinite, tree-like structure whose nodes have the same labels as those of the places and transitions of the Petri net preserving the relationship of adjacency, using *unfolding technique* introduced in [10] and given in [7] in detail under the name of *branching processes*. Any *finite initial* part, i.e., *prefix* of the occurrence net of a cf Petri net can be considered as a derivation tree for the corresponding context-free grammar as it has the same structure as a usual derivation tree, here we can also accept the rule of reading “leaf”-places with tokens from the left to the right as in usual derivation trees. We can also generalize this idea for regulated grammars considering prefixes of the occurrences nets obtained from cf Petri nets with additional places.

We should mention that there are also other approaches in the construction of derivation trees for non-context-free languages, for instance, [11] illustrates that an extended version of pushdown automata allows constructing derivation trees for

Sherzod Turaev is with Faculty of Computer Science and Information Technology, University Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia (e-mail: sherzod@fsktm.upm.edu.my).

Alexander Krassovitskiy is with Research Group on Mathematical Linguistics, University Rovira i Virgili, 43002, Tarragona, Spain (e-mail: alexander.krassovitski@estudiants.urv.cat).

Mohamed Othman is with Faculty of Computer Science and Information Technology, University Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia (e-mail: mothman@fsktm.upm.edu.my).

Mohd Hasan Selamat is with Faculty of Computer Science and Information Technology, University Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia (e-mail: hasan@fsktm.upm.edu.my).

context-sensitive grammars.

The paper is organized as follows: Sections 2.1 and 2.2 cite some notions and notations from the theories of formal languages and Petri nets needed in the sequel. Basic definitions concerning occurrences nets and branching process are given in Sections 2.3 and 2.4. Section 2.5 defines context-free Petri nets as a Petri net counterpart of context-free grammars. In Section 2.6 we briefly cite the definitions of Petri net controlled grammars, and briefly cite the results concerning to their computational power. Parsing algorithms are introduced and analyzed in Section 3: Section 3.1 constructs a general parsing algorithm while Section 3.2 discusses its computational drawbacks and gives some examples of restriction to be imposed on cf Petri nets in order to get a feasible complexity of the parsing. Section 3.3 considers a particular deterministic class of extended cf Petri nets elaborating an Earley-based deterministic parsing algorithm, which has a high practical interest.

II. Preliminaries

The reader is assumed to be familiar with the basic notions of the theories of formal languages and Petri nets, for details refer to [3], [8], [14].

A. Context-free grammars

The set of positive (non-negative) integers is denoted by $\mathbb{N}$ ($\mathbb{N}_0$). We denote by Σ^* the free monoid generated by an alphabet Σ. A string over Σ is a sequence of symbols from the alphabet. The *empty* string is denoted by λ.

A *context-free grammar* is a quadruple $G = (V, \Sigma, S, R)$ where V and Σ are the sets of *nonterminal* and *terminal* symbols, respectively, $S \in V$ is the *start* symbol (the *axiom*), and $R \in V \times (V \cup \Sigma)^*$ is the set of *rules*. A rule of the form $A \to \lambda$ is called *an erasing rule*.

A *derivation relation* on $(V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ is denoted by $\Rightarrow$, its reflexive and transitive closure by $\Rightarrow^*$. The *language* generated by G is defined by

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}.$$

B. Petri Nets

A *Petri net* is a construct (P, T, F, ϕ) where P and T are disjoint finite sets of so called *places* and *transitions*, respectively, $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation* (the set of *directed arcs*) and $\phi : F \to \mathbb{N}$ is a *weight function*.

A Petri net can be represented by a bipartite directed graph with the node set $P \cup T$ where places are drawn as *circles*, transitions as *boxes* and arcs as *arrows*. The arrow representing an arc $(x, y) \in F$ is labeled with $\phi(x, y)$; if $\phi(x, y) = 1$, then the label is omitted.

A mapping $\mu : P \to \mathbb{N}_0$ is called a *marking*. For each place $p \in P$, $\mu(p)$ gives the number of *tokens* in p. Graphically, tokens are drawn as small solid *dots* inside circles.

$${}^\bullet x = \{y \mid (y, x) \in F\}$$

and

$$x^\bullet = \{y \mid (x, y) \in F\}$$

are called *pre-* and *post-sets* of $x \in P \cup T$, respectively.

A transition $t \in T$ is *enabled* by marking μ if and only if $\mu(p) \geq \phi(p, t)$ for all $p \in {}^\bullet t$. In this case t can *occur* (*fire*). Its occurrence transforms the marking μ into the marking μ' defined for each place $p \in P$ by

$$\mu'(p) = \mu(p) - \phi(p, t) + \phi(t, p).$$

We write $\mu \xrightarrow{t}$ to denote that t may fire in μ, and $\mu \xrightarrow{t} \mu'$ to indicate that the firing of t in μ leads to μ'.

A finite sequence $t_1 t_2 \cdots t_k$, $t_i \in T, 1 \leq i \leq k$, is called *an occurrence sequence* enabled at a marking μ and finished at a marking μ' if there are markings $\mu_1, \mu_2, \ldots, \mu_{k-1}$ such that

$$\mu \xrightarrow{t_1} \mu_1 \xrightarrow{t_2} \ldots \xrightarrow{t_{k-1}} \mu_{k-1} \xrightarrow{t_k} \mu'.$$

The markings μ_i, $1 \leq i \leq k-1$, and μ' are called *reachable* from marking μ.

A *marked* Petri net is a construct $N = (P, T, F, \phi, \iota)$ where (P, T, F, ϕ) is a Petri net, ι is the *initial marking*.

C. Occurrence Nets

The *casual*, *conflict* and *concurrency* relations between nodes of a net (P, T, F) are defined as follows.

- two nodes x and y are in *causal relation*, denoted by $x < y$, if the net contains a path with at least one arc leading from x to y.
- x and y are in *conflict relation*, or just in *conflict*, denoted by $x\#y$, if the net contains two paths

  $$st_1 \ldots x \text{ and } st_2 \ldots y$$

  starting at the same place s, and such that $t_1 \neq t_2$.
- x and y are in *concurrency relation*, denoted by $x\ co\ y$, if neither $x < y$ nor $y < x$ nor $x\#y$.

An *occurrence net* is a net $O = (B, E, F')$ where B and E are finite sets of *conditions* (places) and *events* (transitions), respectively, and F' is the *flow relation* such that

- $|{}^\bullet b| \leq 1$ for every $b \in B$;
- O is acyclic, or equivalently, the causal relation is partial order;
- O is finitely preceded, i.e. for every $x \in B \cup E$, the set of elements $y \in B \cup E$ such that $y < x$ is finite;
- no element is in conflict with itself.

It is easy to see that any two nodes of an occurrence net are either in causal, conflict or concurrency relation. $Min(O)$ denotes the set of minimal elements of $B \cup E$ with respect to the causal relation. A set $B' \subseteq B$ is called *co-set* if its elements are pairwise in concurrency relation.

D. Branching Process

A *branching process* of a Petri net $N = (P, T, F)$ is a labeled occurrence net $\xi = (O, h) = (B, E, F', h)$ where the labeling function h satisfies the following properties:

- $h(B) \subseteq P$ and $h(E) \subseteq T$ (h preserves the nature of nodes);
- for every $e \in E$, the restriction of h to ${}^\bullet e$ is a bijection between ${}^\bullet e$ (in N) and ${}^\bullet h(e)$ (in ξ), and similarly for $e^\bullet$ and $h(e)^\bullet$ (h preserves the environment of transitions);
- the restriction of h to $Min(O)$ is bijection between $Min(O)$ and ι (ξ starts at ι);
- for every $e_1, e_2 \in E$, if ${}^\bullet e_1 = {}^\bullet e_2$ and $h(e_1) = h(e_2)$ then $e_1 = e_2$ (ξ does not duplicate the transitions of N).

Let $\xi' = (O', h')$ and $\xi = (O, h)$ be two branching processes of a net system. ξ' is a *prefix* of ξ if O' is a subnet of O satisfying

- $Min(O)$ belongs to O';
- if a condition b belongs to O', then its input event $e \in {}^\bullet b$ in O also belongs to O';
- if an event e belongs to O', then its input and output conditions ${}^\bullet e \cup e^\bullet$ in O also belongs to O';
- h' is the restriction of h to O'.

E. Context-free Petri Nets

The definition of the following type of Petri nets is based on the idea of using similarity between the firing of a transition and the application of a production rule in a derivation in which places are symbols (i.e., nonterminals and terminals) of the grammar, and tokens are separate occurrences of symbols.

Definition 1: A *context-free Petri net* (in short, a *cf Petri net*) with respect to a context-free grammar $G = (V, \Sigma, S, R)$ is a construct $N = (P, T, F, \phi, \beta, \gamma, \iota)$ where

- (P, T, F, ϕ) is a Petri net;
- labeling functions $\beta : P \to V \cup \Sigma$ and $\gamma : T \to R$ are bijections;
- there is an arc from place p to transition t if and only if $\gamma(t) = A \to \alpha$ and $\beta(p) = A$. The weight of the arc (p, t) is 1;
- there is an arc from transition t to place p if and only if $\gamma(t) = A \to \alpha$ and $\beta(p) = x$ where $|\alpha|_x > 0$. The weight of the arc (t, p) is $|\alpha|_x$;
- the initial marking ι is defined by $\iota(\beta^{-1}(S)) = 1$ and $\iota(p) = 0$ for all $p \in P - \{\beta^{-1}(S)\}$.

Example 2: Figure 1 illustrates cf Petri net N_1 corresponding to the grammar G_1 with the rules:

$$R_1 = \{r_1 : S \to aSb, r_2 : S \to ab\}$$

(the other components of the grammar can be seen from these rules).

In [4], [5] it was shown that there is a one-to-one correspondence between the application of (sequence of) rules and the firing of (sequence of) transitions. Thus, it is a very natural and very easy idea to control the derivations in a context-free grammar by adding some features (i.e., places, transitions and arcs) to the associated Petri net. In [16], various Petri net control mechanisms and associated grammars were defined by equipping cf Petri nets with new places and arcs. For instance, it was proven that by adding some places and arcs which satisfy special requirements, precisely, the new places with transitions of a cf Petri net form chains and cycles, one can generate families of vector, matrix and semi-matrix languages.

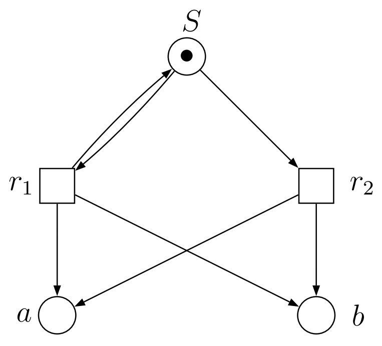

Fig. 1. a cf Petri net N.

Thus the control by a Petri net makes possible to unify a grammar and its control into a Petri net. This approach allows considering the membership problem for grammars in the new point of view: instead of a usual derivation tree, we can use *a Petri net derivation tree* in which the control mechanism is also considered as a part of the tree.

F. Petri Net Controlled Grammars

Since a context-free grammar and its derivation process can also be described by a Petri net, where places correspond to nonterminals, transitions are the counterpart of the production rules, and the tokens reflect the occurrences of symbols in the sentential form, and there is a one-to-one correspondence between the application of (sequence of) rules and the firing of (sequence of) transitions, it is a very natural and very easy idea to control the derivations in a context-free grammar by adding some features to the associated Petri net. In this section we construct Petri net control mechanisms from context-free Petri nets by adding new places and arcs, and define the corresponding grammars, called *Petri net controlled grammars*. It was shown [16] that by adding some places and arcs which satisfy special requirements, precisely, the new places with transitions of a context-free Petri net form chains and cycles, one can generate families of vector, matrix and semi-matrix languages.

Let $\mathcal{P} = \{\rho_1, \rho_2, \ldots, \rho_n\}$ be a set of disjoint chains where each chain $\rho_i = (P_{\rho_i}, T_{\rho_i}, F_{\rho_i}) \in \mathcal{P}$, $1 \le i \le n$, is defined as

$$\rho = t_{i,1} p_{i,1} t_{i,2} p_{i,2} \cdots p_{i,k_i-1} t_{i,k_i}$$

with the sets of places, transitions and arcs, respectively,

$$\begin{aligned} P_\rho &= \{p_{i,1}, p_{i,2}, \ldots, p_{i,k_i-1}\}, \\ T_\rho &= \{t_{i,1}, t_{i,2}, \ldots, t_{i,k_i}\}, \\ F_\rho &= \{(t_{i,j}, p_{i,j}) \mid 1 \le j \le k_i - 1\} \\ &\quad \cup \{(p_{i,j}, t_{i,j+1}) \mid 1 \le i \le k_i - 1\}. \end{aligned}$$

Remark 3: If a chain $\rho \in \mathcal{P}$ consists of a single transitions, i.e., $\rho = t$, then the sets of places and arcs of ρ is considered to be empty, i.e., $P_\rho = F_\rho = \emptyset$.

Definition 4: Let $G = (V, \Sigma, S, R)$ be a context-free grammar with its corresponding context-free Petri net

$$N = (P, T, F, \phi, \beta, \gamma, \iota).$$

Let $T_1, T_2, \dots, T_n$ be a partition of T and

$$\mathcal{P} = \{\rho_1, \rho_2, \dots, \rho_n\}$$

be the set of disjoint chains such that $T_{\rho_i} = T_i$, $1 \le i \le n$, and

$$\bigcup_{\rho \in \mathcal{P}} P_\rho \cap P = \emptyset.$$

An *z-Petri net* is a system

$$N_z = (P \cup Q, T, F \cup E, \varphi, \zeta, \gamma, \mu_0, \tau)$$

where

$$Q = \bigcup_{\rho \in \mathcal{P}} P_\rho \text{ and } E = \bigcup_{\rho \in \mathcal{P}} F_\rho;$$

- the weight function φ is defined by $\varphi(x, y) = \phi(x, y)$ if $(x, y) \in F$ and $\varphi(x, y) = 1$ if $(x, y) \in E$;
- the labeling function $\zeta : P \cup Q \to V \cup \{\lambda\}$ is defined by $\zeta(p) = \beta(p)$ if $p \in P$ and $\zeta(p) = \lambda$ if $p \in Q$;
- the initial marking μ_0 is defined by $\mu_0(p) = \iota(p)$ if $p \in P$ and $\mu_0(p) = 0$ if $p \in Q$;
- τ is the final marking where $\tau(p) = 0$ for all $p \in P \cup Q$.

Example 5: Figure 2 illustrates z-Petri net N_z with respect to the context-free grammar

$$G_4 = (\{S, A, B\}, \{a, b\}, S, R)$$

where R consists of

$$\begin{array}{lll} r_0 : S \to AB, & & \\ r_1 : A \to \lambda, & r_2 : B \to \lambda, & r_3 : A \to aA, \\ r_4 : B \to aB, & r_5 : A \to bA, & r_6 : B \to bB. \end{array}$$

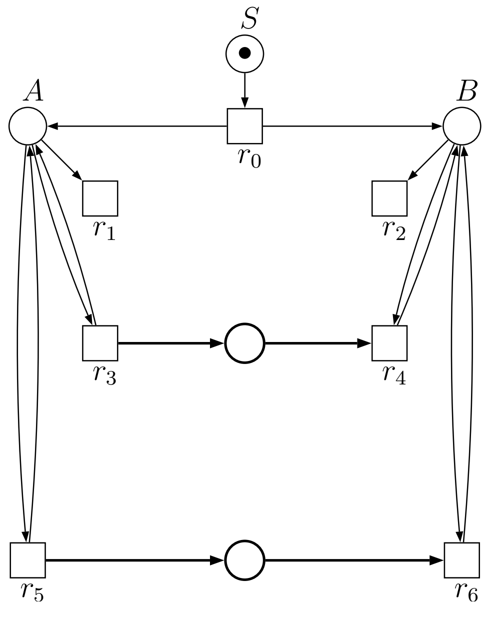

Fig. 2. A z-Petri net N_z

Let $\mathcal{P} = \{\rho_1, \rho_2, \dots, \rho_n\}$ be a set of disjoint cycles where each cycle $\rho_i = (P_{\rho_i}, T_{\rho_i}, F_{\rho_i}) \in \mathcal{P}$, $1 \le i \le n$, is defined as

$$\rho = p_{i,1} t_{i,1} p_{i,2} t_{i,2} \cdots p_{i,k_i} t_{i,k_i} p_{i,1}$$

with the sets of places, transitions and arcs, respectively,

$$\begin{aligned} P_{\rho_i} &= \{p_{i,1}, p_{i,2}, \dots, p_{i,k_i}\}, \\ T_{\rho_i} &= \{t_{i,1}, t_{i,2}, \dots, t_{i,k_i}\}, \\ F_{\rho_i} &= \{(t_{i,j}, p_{i,j}) \mid 1 \le j \le k_i\} \\ &\quad \cup \{(t_{i,j}, p_{i,j+1}) \mid 1 \le i \le k_i - 1\} \\ &\quad \cup \{(t_{i,k_i}, p_{i,1})\}. \end{aligned}$$

Definition 6: Let $G = (V, \Sigma, S, R)$ be a context-free grammar with its corresponding context-free Petri net

$$N = (P, T, F, \phi, \beta, \gamma, \iota).$$

Let $T_1, T_2, \dots, T_n$ be a partition of T and

$$\mathcal{P} = \{\rho_1, \rho_2, \dots, \rho_n\}$$

be the set of disjoint cycles such that $T_{\rho_i} = T_i$, $1 \le i \le n$, and

$$\bigcup_{\rho \in \mathcal{P}} P_\rho \cap P = \emptyset.$$

A *c-Petri net* is a system

$$N_c = (P \cup Q, T, F \cup E, \varphi, \zeta, \gamma, \mu_0, \tau)$$

where

$$Q = \bigcup_{\rho \in \mathcal{P}} P_\rho \text{ and } E = \bigcup_{\rho \in \mathcal{P}} F_\rho;$$

- the weight function φ is defined by $\varphi(x, y) = \phi(x, y)$ if $(x, y) \in F$ and $\varphi(x, y) = 1$ if $(x, y) \in E$;
- the labeling function $\zeta : P \cup Q \to V \cup \{\lambda\}$ is defined by $\zeta(p) = \beta(p)$ if $p \in P$ and $\zeta(p) = \lambda$ if $p \in Q$;
- the initial marking μ_0 is defined by $\mu_0(p) = \iota(p)$ if $p \in P$, and $\mu_0(p_{i,1}) = 1$, $\mu_0(p_{i,j}) = 0$ where $p_{i,j} \in P_i$, $1 \le i \le n$, $2 \le j \le k_i$;
- τ is the final marking where $\tau(p) = 0$ if $p \in P$, and $\tau(p_{i,1}) = 1$, $\tau(p_{i,j}) = 0$ where $p_{i,j} \in P_i$, $1 \le i \le n$, $2 \le j \le k_i$.

Example 7: Figure 3 illustrates a c-Petri net N_c with respect to the context-free grammar given in Example 5.

Let $\mathcal{P} = \{\rho_1, \rho_2, \dots, \rho_n\}$ be a set of cycles such that

$$P_{\rho_1} \cap P_{\rho_2} \cap \cdots P_{\rho_n} = \{p_0\}$$

where each cycle $\rho_i = (P_{\rho_i}, T_{\rho_i}, F_{\rho_i}) \in \mathcal{P}$, $1 \le i \le n$, is defined as

$$\rho = p_0 t_{i,1} p_{i,1} t_{i,2} \cdots p_{i,k_i - 1} t_{i,k_i} p_0$$

with the sets of places, transitions and arcs, respectively,

$$\begin{aligned} P_{\rho_i} &= \{p_0, p_{i,1}, p_{i,2}, \dots, p_{i,k_i - 1}\}, \\ T_{\rho_i} &= \{t_{i,1}, t_{i,2}, \dots, t_{i,k_i}\}, \\ F_{\rho_i} &= \{(p_{i,j}, t_{i,j+1}) \mid 1 \le j \le k_i - 1\} \\ &\quad \cup \{(t_{i,j}, p_{i,j}) \mid 1 \le i \le k_i - 1\} \\ &\quad \cup \{(p_0, t_{i,1})\} \cup \{(t_{i,k_i}, p_0)\}. \end{aligned}$$

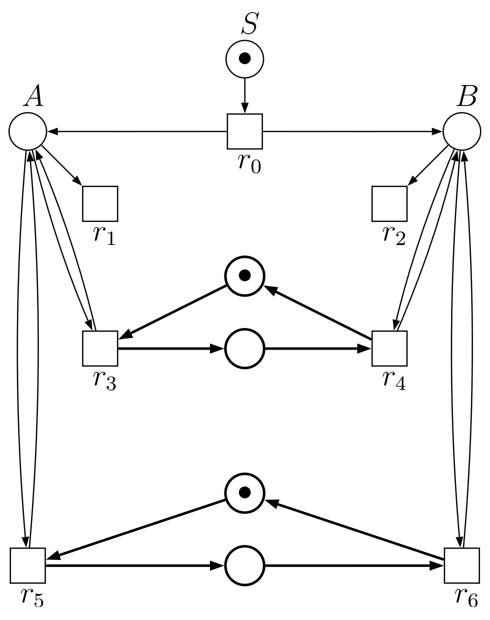

Fig. 3. A c-Petri net N_c

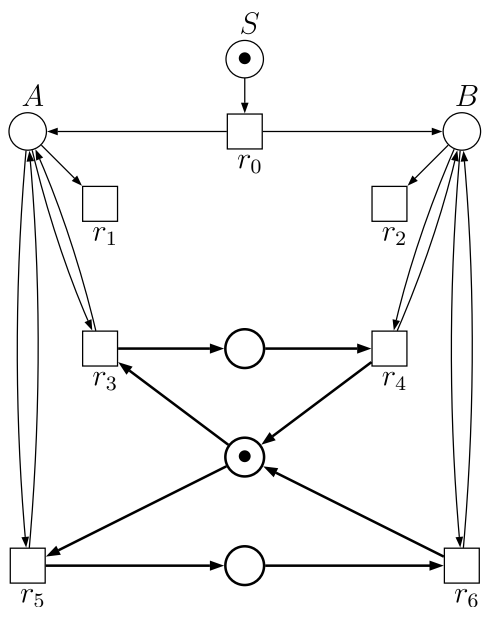

Fig. 4. An s-Petri net N_s

Definition 8: Let $G = (V, \Sigma, S, R)$ be a context-free grammar with its corresponding context-free Petri net

$$N = (P, T, F, \phi, \beta, \gamma, \iota).$$

Let $T_1, T_2, \dots, T_n$ be a partition of T. Let

$$\mathcal{P} = \{\rho_1, \rho_2, \dots, \rho_n\}$$

be the set of cycles such that $T_{\rho_i} = T_i$, $1 \le i \le n$,

$$P_1 \cap P_2 \cap \cdots \cap P_n = \{p_0\}$$

and

$$\bigcup_{\rho \in \mathcal{P}} P_\rho \cap P = \emptyset.$$

An *s-Petri net* is a system

$$N_s = (P \cup Q, T, F \cup E, \varphi, \zeta, \gamma, \mu_0, \tau)$$

where

$$Q = \bigcup_{\rho \in \mathcal{P}} P_\rho, \text{ and } E = \bigcup_{\rho \in \mathcal{P}} F_\rho;$$

- the weight function φ is defined by $\varphi(x, y) = \phi(x, y)$ if $(x, y) \in F$ and $\varphi(x, y) = 1$ if $(x, y) \in E$;
- the labeling function $\zeta : P \cup Q \to V \cup \{\lambda\}$ is defined by $\zeta(p) = \beta(p)$ if $p \in P$ and $\zeta(p) = \lambda$ if $p \in Q$;
- μ_0 is the initial marking where $\mu_0(p_0) = 1$ and $\mu_0(p) = \iota(p)$ if $p \in (P \cup Q) - \{p_0\}$;
- τ is the final marking where $\tau(p_0) = 1$ and $\tau(p) = 0$ if $p \in (P \cup Q) - \{p_0\}$.

Example 9: Figure 4 illustrates a s-Petri net N_s with respect to the context-free grammar given in Example 5.

We define grammars controlled by z (c, s)-Petri nets introduced above.

Definition 10: (i) An *x-Petri net controlled grammar* is a quintuple $G = (V, \Sigma, S, R, N_x)$ where V, Σ, S, and R are defined as for a context-free grammar and

$$N_x = (P \cup Q, T, F \cup E, \varphi, \zeta, \gamma, \mu_0, \tau)$$

is a x-Petri net with respect to the context-free grammar (V, Σ, S, R) where $x \in \{z, c, s\}$.

(ii) The language generated by a x-Petri net controlled grammar G, denoted by $L(G)$, consists of all strings $w \in \Sigma^*$ such that there is a derivation $S \xRightarrow{r_1 r_2 \cdots r_k} w \in \Sigma^*$ and a successful occurrence sequence of transitions $\nu = t_1 t_2 \cdots t_k$ of N_x such that $r_1 r_2 \cdots r_k = \gamma(t_1 t_2 \cdots t_k)$.

We denote the families of languages generated by x-Petri net controlled grammars (with erasing rules) by $\mathbf{PN}_x$, $(\mathbf{PN}_x^\lambda)$ where $x \in \{z, c, s\}$. We denote by $\mathbf{MAT}^{[\lambda]}$, $\mathbf{sMAT}^{[\lambda]}$, $\mathbf{VEC}^{[\lambda]}$ the families of matrix, semi-matrix and vector languages, respectively.

Theorem 11 ([16]):

$$\begin{aligned}(1)\ \mathbf{PN}_s = \mathbf{MAT} \subseteq \mathbf{PN}_z = \mathbf{VEC} \\ \subseteq \mathbf{PN}_z^\lambda = \mathbf{PN}_c^\lambda = \mathbf{PN}_s^\lambda, \\ (2)\ \mathbf{MAT} \subseteq \mathbf{PN}_c = \mathbf{sMAT} \subseteq \mathbf{MAT}^\lambda.\end{aligned}$$

III. Parsing Algorithms

In this section we formalize the notion of the parsing for context-free Petri nets, and present parsing algorithms, particularly for the deterministic restriction of Petri net controlled grammars based on the well-known Earley parsing algorithm. We also discuss the complexity problems of the introduced algorithms.

A. Generalized Parsing

Let $N = (P, T, F, \phi, \iota, \beta, \gamma)$ be the context-free Petri net associated with a context-free grammar $G = (V, \Sigma, S, R)$. Let

$w = x_1x_2 \cdots x_n$, $x_i \in \Sigma$, $1 \leq i \leq n$, be an input string over Σ. The parsing problem is to determine whether w is derivable by N. If yes, then provide an occurrence net (a derivation net) yielding w in a final marking.

The next algorithm for cf Petri nets tracks down a *sentential form* starting from the input string w until the axiom S of the grammar G has been reached. Here a sentential form is defined as a string over terminal and nonterminal symbols of the underlying context-free grammar to specify the ordered set of places. These places are initially marked with symbols of the input string w. As the corresponding marking is simply a vector of places, we keep it as a string over terminal and nonterminal alphabets.

Algorithm 1. General nondeterministic parsing algorithm.
Input: N is a context-free Petri net and $w = x_1x_2 \cdots x_n \in \Sigma^*$.
Output: Occurrence net ON of w, if w can be yielded by the cf Petri net N.

```
// Initialization
Let $\beta : P \to V \cup \Sigma$ be a net labeling function:
$\beta : (N.Places) \to (G.Term \cup G.NonTerm)$.
Let $SF$ be a sentential form over $G.Term \cup G.NonTerm$.
Let a function $\psi : N.Places \times N.Trans \to N.Places \times N.Trans$ return a copy of a place or a transition with the same label.
$ON.Places \leftarrow \{p_i | \ \beta(p_i) = x_i, i \in [n]\}$
$ON.Trans \leftarrow \emptyset$
$ON.Flow \leftarrow \emptyset$
repeat
  // Nondeterministic generation of occurrence net
  $select\ r \in N.Trans$
  Let $R \subseteq ON.Places$ s.t. $r^\bullet = R$ and $^\bullet R = \emptyset$ and preserve order in $R$
  // i.e., it finds a set of places $R$ yielded by transition $r$
  $r' \leftarrow \psi(r)$    // copy transition
  $PIn \leftarrow \psi(^\bullet r)$ // copy incoming places
  $ON.Trans \leftarrow ON.Trans \cup r'$
  $ON.Places \leftarrow ON.Places \cup \ PIn$
  $ON.Flow \leftarrow ON.Flow \cup (r', R) \cup \{(p', r') | p' \in PIn\}$
  //updating flow relation
  $SF \leftarrow (SF - \beta(^\bullet r)) \cup \beta(PIn')$ // symbolic replacement of rewritten symbols in $SF$ (in proper positions)
  if $ON.Places$ are NOT ordered properly within $SF$ then
    return FAILURE
  end if
until $\beta^{-1}(S) \in ON$ or $R$ can not be computed
// do output $ON$
if $\beta^{-1}(S) \in OC$ then
  return $ON$
else
  return FAILURE
end if
```

Figure 5 shows the branching process of the cf Petri net N_1 illustrated in Fig. 1.

Our algorithm constructs an occurrence net in the down-up way. In the algorithm we use the selection procedure which chooses the transition to be added into occurrence net. Due to the procedure per each cycle takes arbitrary transition $r \in N.Trans$, the algorithm is nondeterministic. In fact, without any restriction on the cf Petri net N, this algorithm may even not terminate. On the other side, for the net N, if grammar G has terminal and/or λ-productions, the algorithm always terminates. But the algorithm is still not effective for this type of a Petri net, because the performance time for the algorithm, if it being transformed into a deterministic computation device, is exponential regarding to both the length of the input string and the size of N.

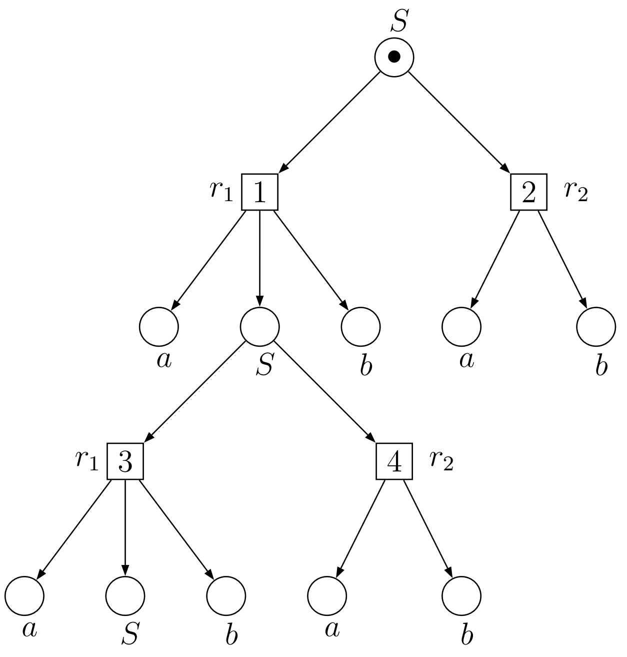

Fig. 5. The unfolding of the cf Petri net N_1.

By the algorithm the construction of the reversed net is defined inexplicitly. The problem is how to handle the recursive growth of the number of possible occurrence sequences of the Petri net (the net branching). If derivation is blocked with an incompatible word (the places of ON are not properly ordered according to SF) then it rolls back and restores previous sentential form.

B. Complexity vs. Restrictions

In this section we analyze the complexity problem of parsing algorithms for cf Petri nets with some restrictions. Our aim is to define subclasses of cf Petri nets such that the parsing problem can be solved efficiently.

In the previous section we have discussed that several language classes can be resulted by extending cf Petri nets with additional features. For instance, the extension with chains and cycles allows generating all vector and matrix languages (for details, see [16]). Since the membership (parsing) problem for matrix languages is NP-hard, this problem for the language families generated by grammars controlled by extended cf Petri nets are also NP-hard. On the other hand, restrictions imposed on matrix grammars, for example, the right-hand side of each rule of a grammar has the unique prefixes (suffixes), makes possible to construct the effective parsing algorithms for some subclasses of matrix languages. Thus, we also need to define such subclasses of (extended) cf Petri nets that the parsing problem is solved efficiently.

Further, we consider two restrictions imposed on extended cf Petri nets such that, first, any Petri net derivation can only have left-to-right control chains, which excludes permutations of derived (terminal) symbols; second, in any Petri net derivation, the leftmost nonterminal symbol to be rewritten by the chain controls. These restrictions determinate the processing of an extended cf Petri net applied to syntactical form. Meanwhile, the choice of chain controls remain (at each derivation step) nondeterministic. Unfortunately, the word acceptance problem for this type of "deterministic" extended cf Petri nets is still NP-complete, which can be proved by reduction from 3-SAT problem.

Another way to restrict the complexity is realized by simplifying controls in such a way that it excludes high nondeterminism while generating occurrence net. For instance, if procedure *select* in the algorithm nondeterministically chooses a "wrong" computation, then it must abort very quickly. We call this notion an *effective branching*. Alternatively, it is advisable to have a decisive procedure which can select "good" branches to be first proceeded. Unfortunately, it may strongly depend on concrete instance of the modeled problem.

C. Deterministic Extended CF Petri Nets

We define the class of *deterministic* extended cf Petri net controlled (ECFPNC) grammars for which Petri net controls can be unfolded to occurrences nets unambiguously. We call an extended cf Petri net controlled G is *k-deterministic* if the application of Petri net controls is determined by k lookahead symbols of the input string being rear right-to-left. The family of (k-)deterministic ECFPNC languages consists of all languages generated by (k-)deterministic ECFPNC grammars.

We should mention that since deterministic restriction is applied only to Petri net controls, context free rules do not suffer from this restriction. Hence the family of context-free languages is included in the family of (k-)deterministic ECFPNC languages.

The derivation (unfolding process) of such grammars is constructed as follows. Suppose first l symbols of a string $x_1x_2\cdots x_lx_{l+1}\cdots x_n$ have been already mapped into occurrence net (ON), i.e., the ON contains nonterminal and control places and l terminal places marked with $x_1, x_2, \ldots, x_l$. Then a new place marked with x_{l+1} (together with the corresponding nonterminal and control places, arcs) can be added to the ON by observing $x_{l+1}x_{l+2}\cdots x_{l+k}$ look-ahead symbols. The corresponding family of languages can be considered as a family of *k-step predictable (windowed) languages*. By definition, the family of deterministic ECFPNC's can be handled uniquely, for example, by the following algorithm.

Algorithm 2.

```
// Initialization
**Let** $ON$ be a variable of occurrence net to be generated
$ON \leftarrow initiate\_ON(x_1x_2\cdots x_k)$
**for all** $l \in [n-k]$
**if** $update\_ON(x_lx_{l+1}x_{l+2}\cdots x_{l+k}) \neq \emptyset$ **then**
    $ON \leftarrow ON \uplus update\_ON(ON, x_lx_{l+1}x_{l+2}\cdots x_{l+k})$
**else**
    **return** FAILURE
**end if**
// Do output $ON$
**return** $ON$
```

Both functions

$$initiate\_ON : \Sigma^k \to PN$$

and

$$update\_ON : PN \times \Sigma^k \to PN$$

provide the encapsulation for invocations to the occurrence net. Binary operator $\uplus$ is used to denote joining of two nets. Unfortunately, because the definition is too general it has a lack of a clear complexity issues and an elementary net application. To handle these problems, we can follow two possible ways (at least). The first way is to extend the definition of a complexity class restriction, i.e., a portion of ON including a single terminal place (with corresponding nonterminal places and transitions) can be added within a number of computational steps not exceeding $C \cdot n \cdot k$, where k is the number of look-ahead symbols, n is the size of ECFPNC. Such complexity class, intuitively, is for the algorithms that can proceed explicitly a singular net review for each input symbol to be read. The second way to deal with deterministic ECFPNC's is to restrict net controls by requiring that the nondeterministic choice of ECFPNC must be eliminated. It specifies a mapping

$$AC : \Sigma^k \times V \to (T \times P) \cup (P \times T)$$

in explicit form[1]. It can be realized by requiring that control net elements can be used only if special markers in input string are being accessed while processing k next input symbols.

In the next algorithm, we use Early-like technique to construct an occurrence net corresponding to input. During the process markings (token distribution) preserve the net controls dynamically, i.e., transitions fire the net on-fly. Hence, lexical order of symbols of the context productions is essential, we preserve this order unexpectedly using lists to store net elements. We define function $SentForm : ON \to (\Sigma \cup V)^*$ to read "sentential" form of the occurrence net. $SentForm$ collects leaf labels in the "natural" order of reading the tree-like occurrence net. It skips empty leaf places and concatenates labels of nonempty leaf places in the left-to right way. We denote $\bar{\cup}$ to be union operation over places and transitions which preserves the tree ordering.

[1]This mapping defines Petri net control via single controlling place. It is valid only for unordered vector grammars. For other types of regulated grammars the mapping of Petri net controls can be done analogously

Algorithm 3. An Early-like deterministic ECFPNC parsing.
Input: $N \in ECFPN$, $w = x_1 x_2 \cdots x_n \in \Sigma^*$,
$AC : \Sigma^k \times V \to T \times P \times T$.
Output: Corresponding occurrence net ON.

```
// Initialization
Let $\beta$ be a net labeling function
$\beta : (N.Places) \to (G.Term \cup G.NonTerm)$
Let a function $\psi : N.Places \times N.Trans \to N.Places \times N.Trans$ returns a copy of place or a transition with the same label.
$PTemp \leftarrow \psi(p)$ s.t., $p \in N.Places$ and $\beta(p) = S$
$ON \leftarrow (\{PTemp\}, \emptyset, \emptyset)$;
$Mark \leftarrow (p)$ //a dynamical list corresponding to the current net marking
$Controls \leftarrow []$ // a queue to proceed net controls has type $T \times P \cup P \times T$
$i \leftarrow 1$
$Config \leftarrow \lambda$ // a stack structure to store a string of labels of the net configurations
// $Config_0$ refers to the head symbol
while there is no $p \in ON.Places$ s.t. $\beta(p) \in G.NonTerm$ do
  $Available\_Controls \leftarrow AC(x_{i+1}, x_{i+2}, \ldots, x_{i+k})$
  $PsTemp \leftarrow \{\psi(PTemp^\bullet)\}$
  $TsTemp \leftarrow \{PsTemp^\bullet\}$
  $ON.Trans \leftarrow ON.Trans \bar{\cup} PsTemp$
  $ON.Places \leftarrow ON.Places \bar{\cup} TsTemp$
  $ON.Flow \leftarrow ON.Flow \cup \{(p,t) \in PsTemp \times TsTemp | N.Flow(p,t)\} \cup$
      $\{(t,p) \in TsTemp \times PsTemp | N.Flow(t,p)\} \cup$
      $\{(p',t) \in P \times TsTemp |\ p' \in ON.Places$ is marked and $(p',t) \in Controls\}$
  $Controls \leftarrow Controls \cup \{(t,p') \in TsTemp \times PsTemp\} \cap Available\_Controls$
  Let $TTemp \in TsTemp \cap Available\_Controls$ and $TTemp$ is enabled
  $Config \overset{put}{\leftarrow} \beta(TTemp)$ // adds the current transition to the stack
  $Fire\_Transition(TTemp, ON)$
  Let $pr\_SentForm(ON)$ computes the maximal prefix of $SentForm(ON)$ over terminals $\Sigma$
  while $pr\_SentForm(ON)$ is NOT a prefix of $w$ do
    $TTemp \overset{extract}{\leftarrow} Config_0$
    $Fire\_Transition^{-1}(TTemp, ON)$ // to do the reversed firing of $TTemp$
  end while
  Let $PTemp$ be a place labeled with the next non-unfolded branch in $Config$
  if $Config = \lambda$ and there is no branches to unfold then
    return FAILURE // the algorithm is blocked
    exit
  end if
  $i \leftarrow |pr\_SentForm(ON)|$
end while
return $ON$
```

The realization of the algorithm is deterministic, if AC is deterministic. The algorithm always terminates, if there is no chain and λ-productions in G. The algorithms continues the computation while ON contains marked places labeled with nonterminals. It unfolds places labeled with nonterminals. The main cycle uses $Available\_Controls \leftarrow AC(x_1, x_2, \ldots, x_k)$ to explicitly identify the set of the net controls available by the context $x_1, x_2, \ldots, x_k$, $k < n$. The built occurrence net resolves conflicts of the nondeterministic choice of the grammar by selecting them in a sequel. Since the algorithm utilizes the Early-based parsing technique, the worst case complexity will differ only on a constant factor from that of Earley($O(|N|^2 \cdot n^3)$). The trick here is in predefined function AC being aware of available controls.

IV. Conclusion

In this paper we attempt to define a handful formalism working efficiently with languages slightly more powerful than context-free languages. The introduced deterministic extended cf Petri nets can be applied in the modeling of signal processing systems. With general assumptions we may say that its alphabet contains special systems intended to control the input processing. e.g., it may switch the controlling device or computation pipes. Hence, if an input string can be recognized by some ECFPNC by using the control of input symbols then the parsing algorithm can be applied efficiently. Our further goal is to realize the parsing algorithm and then to test on real examples.

There are still many interesting topics regarding ECFPNC parsing. One may try to follow divide and conquer strategy by splitting input string by independent portion or try to preprocess the grammar first.

Acknowledgements

The first author has been supported by University Putra Malaysia via RUGS 05-01-10-0896RU/F1 while the second and third authors have been supported by Fundamental Research Grant Scheme FRGS/1/11/SG/UPM/01/1.

References

[1] S. Abraham, Some qeustions of phrase-structure grammars, *Comput. Linguistics* 4, 1965, pp. 61–70.
[2] S. Crespi-Reghizzi and D. Mandrioli, Petri nets and commutative grammars, *Internal Report* 74–85, Laboratorio di Calcolatori, IEEPM 1974.
[3] J. Dassow and Gh. Păun, *Regulated rewriting in formal language theory,* Springer–Verlag, Berlin, 1989.
[4] J. Dassow and S. Turaev, Petri net controlled grammars with a bounded number of additional places, *Acta Cybernetica* 19, 2010, pp. 609–634.
[5] J. Dassow and S. Turaev, k-Petri net controlled grammars. In: C. Martín-Vide, F. Otto and H. Fernau, (eds.) *LNCS* 5196, 2008, pp. 209–220. Springer–Verlag, Berlin.
[6] M.A. Drighiciu, A. Petrisor, M. Popescu, A Petri Nets approach for hybrid systems modeling, *International Journal of Circuits, Systems and Signal Processing,* 2(3), 2009, pp. 55–64.
[7] J. Engelfriet, Branching processes of Petri nets, *Acta Informatica* 28, 1991, pp. 575–591.
[8] J.E. Hopcroft and J.D. Ullman, *Introduction to automata theory, languages, and computation,* Addison-Wesley Longman Publishing Co., Inc., 1990.
[9] V. Marek and M. Češka, Petri nets and random-context grammars, In: the 35th Spring Conference: Modelling and Simulation of Systems, MARQ Ostrava, Hardec nad Moravicí, 2001, pp.145–152.

[10] K.L. McMillan, A Technique of a State Space Search Based on Unfolding, *Formal Methods in System Design* 6(1), 1995, pp. 45–65.

[11] B. Nagy. *Leftmost derivation and shadow-pushdown automata for context-sensitive languages,* In: the 10th WSEAS International Conference on Computers, Vouliagmeni, Athens, Greece, 2006, pp. 962–967.

[12] C.A. Petri, *Kommunication mit Automaten,* PhD Thesis. University of Bonn, 1962.

[13] A.A. Pouyan, A.H.Beigi, M.Kadkhoda, *An agent-based model for virtual tourism using object Petri nets,* In: the 5th WSEAS International Conference on Circuits, Systems, Electronics, Control & Signal Processing, Dallas, Texas, USA, 2006, pp. 149–154.

[14] W. Reisig and G. Rozenberg G, (eds.) *Lectures on Petri nets I: Basic models,* LNCS 1491, Springer–Verlag, 1998.

[15] A.S. Staines, Supporting Requirements Engineering with Different Petri Net Classes, *International Journal of Computers,* 4(4), 2010, pp. 215–222.

[16] S. Turaev, *Petri net controlled grammars,* PhD Thesis. In: Tesis Doctorals en Xarxa. University Rovira i Virgili, 2010.