On one context-free language for producer/consumer Petri net with the unbounded buffer

Vadym Mukhin
Mathematical methods of system analysis department
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine
v.mukhin@kpi.ua

Vitalii Statkevych
Mathematical methods of system analysis department
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine
mstatckevich@yahoo.com

***Abstract*—We consider the producer/consumer Petri net and the context-free formal language that it generates. Nonregularity of the language is proved, the corresponding pushdown automaton and context-free grammar are obtained. The connection with the Dyck formal language is pointed out.**

Keywords—Petri nets; formal languages; automata; computer science; programming; software design

I. INTRODUCTION

Petri nets, introduced by C. A. Petri, are used in modeling, analysis and design of various systems [1, 2]. Petri nets can model nondeterministic and asynchronous behaviour of the systems, parallelism and concurrency features and conflicts. A variety of synchronization problems are known, among them are the mutual exclusion problem, the producer/consumer problem, the dining philosophers problem, the readers/writers problem, et al. The dining philosophers problem and the producer/consumer problem were proposed by E. Dijkstra (1968).

A particular Petri net can be associated with a certain language. In [1] twelve classes of Petri net languages are introduced, known relations among them are given and the connections with Chomsky hierarchy are stated (the theory of formal languages is described in [3–5]). In particular every regular language is a Petri net language, every Petri net language is a context-sensitive language. There exist context-free languages which are not Petri net languages (for example even length palindromes $\{ww^R : w \in \{a,b\}^*\}$), as well as there exist Petri net languages which are not context-free languages (for example $\{a^n b^n c^n : n \geq 1\}$) [1].

If a particular Petri net is a bounded net, the reachability graph of the net can be transformed into a finite automaton, i.e. one can associate the graph vertices with the automaton states, the graph edges with the automaton transitions and the initial marking with the initial state [6]. Thus the bounded net generates a regular language.

The paper [7] introduces the method of computing a regular expression for the language of a safe Petri net. The technique is based on reductions of Petri nets and omits the constructing the complete reachability graph. The method is illustrated using the dining philosophers problem as an example.

However, the producer/consumer Petri net with the unbounded buffer is not a safe Petri net, thus the method proposed in [7] cannot be applied in our case. Further we show that the corresponding language is not a regular language, but a context-free language (see also [8]).

Context-free languages are essential in syntax analysis and compilation process, therefore are related to programming languages. Many programming languages use constructions that are typical for context-free languages [3]. For example, let $f : A \times A \to A$ be a binary operation on an arbitrary set A, one can match the expression $f(f(x_1, f(x_2, f(x_3, x_4))), f(x_5, x_6))$ with the word $((((()))()())$ of balanced parentheses, while the language of balanced parentheses (the Dyck language [3, 5]) is generated by the context-free grammar.

Different connections between Petri nets and the theory of formal languages and grammars were investigated in [9–13].

In this paper we consider the L-type Petri net language for the producer/consumer problem with the unbounded buffer. This Petri net language is context-free and can be generated by context-free grammar, and classical programming languages include several constructions which can be analyzed by context-free grammars.

The paper is organized as follows. Basic notations and constructions are given in Section II. We prove that the considered language is nonregular and present the corresponding pushdown automaton and the context-free grammar in Section III. Connection with the Dyck language is shown in Section IV. Practical application is given in Section V. This paper uses the results obtained in [8, 14].

II. PRELIMINARIES

In the producer/consumer problem producer process A creates objects (tokens), which are put in the buffer (place p_5), consumer process B removes objects (tokens) one at a time from the buffer p_5 and consumes them (see Fig. 1).

978-1-7281-6870-8/20/$31.00 ©2020 IEEE

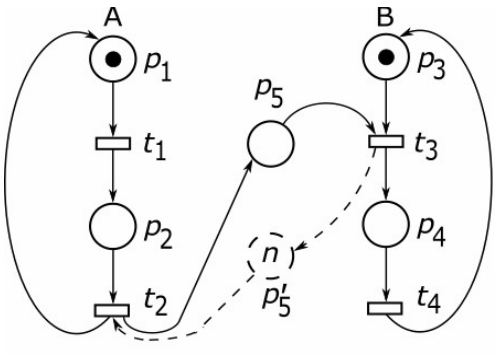

Fig. 1. Producer/consumer Petri net

This problem has several variants, among them are: a) the problem with the unbounded buffer; b) the problem with the bounded buffer; c) the multiple-producer/multiple-consumer problem, et al.

The problem with the bounded buffer deals with buffer of size n with additional place p_5', which has initially n tokens. The multiple-producer/multiple-consumer problem deals with s producers and t consumers, so that places p_1 and p_3 have initially s and t tokens respectively. We denote the initial marking of the net as μ_0.

We consider the L-type Petri net language for the problem with the unbounded buffer. Let F be a finite set of final markings. Let w be the sequence of transitions so that the marking μ, obtained as a result of firing the sequence of transitions w, is the final marking, i.e. $\mu \in F$. Let $\sigma : T \to A$ be a labeling function. Then the L-type Petri net language is equal to the set of strings $\sigma(w)$ [1].

In our case let $A = \{a_1, a_2, a_3, a_4\}$ be an alphabet. We associate symbols a_1, a_2, a_3, a_4 with the transitions t_1, t_2, t_3, t_4 respectively so that the labeling function $\sigma : T \to A$ has the form $\sigma(t_i) = a_i$, $i = 1,\ldots,4$. Thus the concerned net is a free-labeled net. The only final marking is equal to the initial marking, i.e. $F = \{\mu_0\} = \{(1,0,1,0,0)\}$. We denote the L-type Petri net language of the concerned net as L.

A pushdown automaton (see [3–5]) is the 7-tuple $P = (Q, T, \Gamma, \Delta, I, Z_0, F)$, where Q is a finite set of states, T is a finite set of input symbols, Γ is a finite stack alphabet, $\Delta \subset Q \times (T \cup \{\varepsilon\}) \times \Gamma \times Q \times \Gamma^*$ is the transition relation, $I \subset Q$ is the set of start states, $Z_0 \in \Gamma$ is the initial stack symbol and $F \subset Q$ is the set of accepting states (here ε denotes an empty string).

A formal grammar [3–5] is the 4-tuple $G = (V, T, P, S)$, where V is a finite set of variables (also called nonterminal symbols N), T is a finite set of terminal symbols, $V \cap T = \varnothing$, P is a finite set of production rules and $S \in V$ is the start symbol.

III. The Pushdown Automaton and the Context-free Grammar for the Producer/Consumer Petri Net Language

Here we state the pushdown automaton that accepts the language L and the context-free grammar that generates it.

Theorem 1. The L-type Petri net language L for the producer/consumer problem is accepted by the pushdown automaton $P = (Q, T, \Gamma, \Delta, \{q_0\}, Z_0, \{q_0\})$, where the set of states $Q = \{q_0, q_1, q_2, q_3\}$, the set of input symbols $T = \{a_1, a_2, a_3, a_4\}$, the stack alphabet $\Gamma = \{A, Z_0\}$ and the transition relation Δ contains 15 transitions

$\delta_0 = (q_0, \varepsilon, Z_0, q_0, \varepsilon)$, $\delta_1 = (q_0, a_1, Z_0, q_1, Z_0)$,
$\delta_2 = (q_0, a_1, A, q_1, A)$, $\delta_3 = (q_0, a_3, A, q_2, \varepsilon)$,
$\delta_4 = (q_1, a_2, Z_0, q_0, AZ_0)$, $\delta_5 = (q_1, a_2, A, q_0, AA)$,
$\delta_6 = (q_1, a_3, A, q_3, \varepsilon)$, $\delta_7 = (q_2, a_1, Z_0, q_3, Z_0)$,
$\delta_8 = (q_2, a_1, A, q_3, A)$, $\delta_9 = (q_2, a_4, Z_0, q_0, Z_0)$,
$\delta_{10} = (q_2, a_4, A, q_0, A)$, $\delta_{11} = (q_3, a_2, Z_0, q_2, AZ_0)$,
$\delta_{12} = (q_3, a_2, A, q_2, AA)$, $\delta_{13} = (q_3, a_4, Z_0, q_1, Z_0)$,
$\delta_{14} = (q_3, a_4, A, q_1, A)$.

The top and the bottom of the stack $\Gamma = Z_1 \ldots Z_n$ are considered to be the leftmost symbol Z_1 and the rightmost symbol $Z_n = Z_0$ respectively. The proof of the theorem can be easily obtained from the construction given in [8]. Thus the language L is the context-free language. Note that the given pushdown automaton P is the deterministic pushdown automaton.

Theorem 2. The language L is nonregular.

Proof. We prove the theorem by contradiction, thus we suppose that the language L is nonregular. Then according to the pumping lemma for regular languages [3–5] there exists a natural number $p \geq 1$ such that for every string $w \in L$ of length $|w| \geq n$ one can write $w = xyz$, where $y \neq \varepsilon$, $|xy| \leq p$ and $xy^i z \in L$ for all $i \geq 0$. We consider the string $w = (a_1 a_2)^p (a_3 a_4)^p \in L$ of length $|w| = 4p$. Due to the inequality $|xy| \leq p$ we obtain that the nonempty string y contains only symbols a_1 and a_2. We also obtain $xz \in L$ in case of $i = 0$. In order to generate the string $xz \in L$ less than p tokens are put in the buffer (place p_5) and precisely p tokens are removed from the place p_5. The obtained contradiction proves that the language L is nonregular.

We use the technique described in [3–5] in order to convert the pushdown automaton P to the context-free grammar, that generates the language L. We consider the formal grammar $G = (V, T, P, S)$ with the set of variables $V = \{S\} \cup \{[q_i Z q_j] : q_i \in Q, q_j \in Q, Z \in \Gamma\}$, the set

of terminal symbols $T = \{a_1, a_2, a_3, a_4\}$, the start symbol S and the set of production rules P. The set P is constructed in the following way:

- the set P contains $S \to [q_0 Z_0 q_i]$ for all $q_i \in Q$;
- for every transition $\delta = (q_i, a, Z, q_j, Z_1 \dots Z_k) \in \Delta$, $k \geq 1$ from the transition relation the set P contains $[q_i Z s_k] \to a[q_j Z_1 s_1][s_1 Z_2 s_2] \dots [s_{k-1} Z_k s_k]$ for every sequence of states $s_1, \dots, s_k \in Q$;
- for every transition $\delta = (q_i, a, Z, q_j, \varepsilon) \in \Delta$ the set P contains $[q_i Z q_j] \to a$.

Thus P contains the following production rules:

- $S \to [q_0 Z_0 q_0] \mid [q_0 Z_0 q_1] \mid [q_0 Z_0 q_2] \mid [q_0 Z_0 q_3]$,
- $[q_0 Z_0 q_0] \to \varepsilon$ for δ_0,
- $[q_0 Z_0 q_i] \to a_1 [q_1 Z_0 q_i]$ for δ_1,
- $[q_0 A q_i] \to a_1 [q_1 A q_i]$ for δ_2,
- $[q_0 A q_2] \to a_3$ for δ_3,
- $[q_1 Z_0 q_i] \to a_2 [q_0 A q_j][q_j Z_0 q_i]$ for δ_4,
- $[q_1 A q_i] \to a_2 [q_0 A q_j][q_j A q_i]$ for δ_5,
- $[q_1 A q_3] \to a_3$ for δ_6,
- $[q_2 Z_0 q_i] \to a_1 [q_3 Z_0 q_i]$ for δ_7,
- $[q_2 A q_i] \to a_1 [q_3 Z_0 q_i]$ for δ_8,
- $[q_2 Z_0 q_i] \to a_4 [q_0 Z_0 q_i]$ for δ_9,
- $[q_2 A q_i] \to a_4 [q_0 A q_i]$ for δ_{10},
- $[q_3 Z_0 q_i] \to a_2 [q_2 A q_j][q_j Z_0 q_i]$ for δ_{11},
- $[q_3 A q_i] \to a_2 [q_2 A q_j][q_j A q_i]$ for δ_{12},
- $[q_3 Z_0 q_i] \to a_4 [q_1 Z_0 q_i]$ for δ_{13},
- $[q_3 A q_i] \to a_4 [q_1 A q_i]$ for δ_{14},

where $0 \leq i \leq 3$ and $0 \leq j \leq 3$. Afterwards, we remove useless variables and corresponding production rules. Thus we obtain the context-free grammar G, that generates the language L, in simpler notation.

IV. Connection with the Dyck Language

The producer/consumer problem has a connection with the Dyck language. The Dyck language is a context-free language over the alphabet $\{a_1, \dots, a_n, a'_1, \dots, a'_n\}$ with $2n$ symbols, that is generated by the context-free grammar $S \to \varepsilon \mid a_1 S a'_1 S \mid \dots \mid a_n S a'_n S$. More informally the Dyck language contains the strings of balanced parentheses of n different types, where a_i stands for the left parenthesis and a'_i stands for the right one [3, 5].

One can choose another labeling function $\sigma_1(t_2) = a_2$, $\sigma_1(t_3) = a_3$, $\sigma_1(t_1) = \sigma_1(t_4) = \varepsilon$ so that t_1 and t_4 are null labeled transitions. Then the L^λ-type Petri net language is equal to the Dyck language with one type of parentheses, the symbols a_2 and a_3 stand for the left and right parenthesis respectively.

For the labeling function σ introduced above the grammar G has the leftmost derivations $[q_0 Z_0 q_0] \Rightarrow \varepsilon$,

$$[q_0 Z_0 q_0] \Rightarrow a_1 [q_1 Z_0 q_0] \Rightarrow a_1 a_2 [q_0 A q_2][q_2 Z_0 q_0] \Rightarrow$$
$$\Rightarrow a_1 a_2 a_3 [q_2 Z_0 q_0] \Rightarrow a_1 a_2 a_3 a_4 [q_0 Z_0 q_0] \Rightarrow a_1 a_2 a_3 a_4,$$
$$[q_0 Z_0 q_0] \Rightarrow a_1 [q_1 Z_0 q_0] \Rightarrow a_1 a_2 [q_0 A q_2][q_2 Z_0 q_0] \Rightarrow$$
$$\Rightarrow \dots \Rightarrow (a_1 a_2)^n [q_0 A q_2][q_2 A q_2]^{n-1} [q_2 Z_0 q_0] \Rightarrow \dots \Rightarrow$$
$$\Rightarrow (a_1 a_2)^n (a_3 a_4)^{n-1} [q_0 A q_2][q_2 Z_0 q_0] \Rightarrow$$
$$\Rightarrow (a_1 a_2)^n (a_3 a_4)^{n-1} a_3 [q_2 Z_0 q_0] \Rightarrow$$
$$\Rightarrow (a_1 a_2)^n (a_3 a_4)^{n-1} a_3 a_4 [q_0 Z_0 q_0] \Rightarrow (a_1 a_2)^n (a_3 a_4)^n,$$
$$[q_0 Z_0 q_0] \Rightarrow \dots \Rightarrow a_1 a_2 a_3 a_4 [q_0 Z_0 q_0] \Rightarrow \dots \Rightarrow$$
$$\Rightarrow (a_1 a_2 a_3 a_4)^2 [q_0 Z_0 q_0] \Rightarrow (a_1 a_2 a_3 a_4)^2.$$

These derivations correspond to the Dyck language with one type of parentheses, the pairs of symbols $a_1 a_2$ and $a_3 a_4$ stand for the left and right parenthesis respectively.

There also exist other leftmost derivations such as

$$[q_1 Z_0 q_1] \Rightarrow a_2 [q_0 A q_2][q_2 Z_0 q_1] \Rightarrow a_2 a_3 [q_2 Z_0 q_1] \Rightarrow$$
$$\Rightarrow a_2 a_3 a_1 [q_3 Z_0 q_1] \Rightarrow a_2 a_3 a_1 a_4 [q_1 Z_0 q_1],$$
$$[q_1 A q_1] \Rightarrow a_2 [q_0 A q_2][q_2 A q_1] \Rightarrow a_2 a_3 [q_2 A q_1] \Rightarrow$$
$$\Rightarrow a_2 a_3 a_1 [q_3 A q_1] \Rightarrow a_2 a_3 a_1 a_4 [q_1 A q_1],$$
$$[q_1 Z_0 q_1] \Rightarrow a_2 [q_0 A q_2][q_2 Z_0 q_1] \Rightarrow a_2 a_3 [q_2 Z_0 q_1] \Rightarrow$$
$$\Rightarrow a_2 a_3 a_4 [q_0 Z_0 q_1] \Rightarrow a_2 a_3 a_4 a_1 [q_1 Z_0 q_1],$$
$$[q_1 A q_1] \Rightarrow a_2 [q_0 A q_2][q_2 A q_1] \Rightarrow a_2 a_3 [q_2 A q_1] \Rightarrow$$
$$\Rightarrow a_2 a_3 a_4 [q_0 A q_1] \Rightarrow a_2 a_3 a_4 a_1 [q_1 A q_1]$$

and similar to them, which can be viewed as the analogs of derivations $A \overset{*}{\Rightarrow} a_2 a'_2 a_1 a'_1 A$ and $A \overset{*}{\Rightarrow} a_2 a'_2 a'_1 a_1 A$ (here A denotes a variable) for the Dyck language with two distinct types of parentheses.

We recall that the well-known Chomsky–Schützenberger representation theorem states that a formal language is context-free if and only if it is represented as a homomorphic image of the intersection of the certain Dyck language and certain regular language. The derivations obtained above due to their simplicity illustrate the fact that in the particular case of the producer/consumer Petri net the context-free language L is "in close proximity" to the Dyck language.

Remark. As stated above, the multiple-producer/ multiple-consumer problem deals with s producers and t consumers, so that places p_1 and p_3 have initially s and t tokens respectively. Thus the initial marking of the corresponding Petri net is equal to $\mu_0 = (s,0,t,0,0)$. The technique used in Theorem 1 can be extended to this case, thus the pushdown automaton, that accepts the considered Petri net language, can be obtained.

V. Practical Application of the Suggested Mechanisms

We can use the constructions similar to the proposed pushdown automata and the proposed context-free grammar in order to implement the LL-parser (the description of LL-parsing and the definition of $LL(k)$-grammar can be found in [3, 15–17]). The LL-parser can use the procedure of string splitting. Namely, the input string is split into several substrings, the reducing of the string is repeated recursively until parsing the entire input string. Syntax constructions are supposed to begin with an opening brace and end with the matching closing one.

This function can be implemented as follows [15]:

1) the function determines whether the string begins with the opening brace and ends with the matching closing one;

2) both conditions met, the braces are deleted and the string is passed to the function that parses the list of key-value pairs.

Thus two different functions have to be implemented, i.e. the function, that parses the given string, and the function, that parses the list of key-value pairs.

The classical LL-parser uses the function to handle nonterminal symbols, performing different actions for each symbol [15]. In our case, braces are used in addition to classical delimiters. Moreover, we use not only the function, that splits the string by the delimiter (comma between key-value pairs and colon between key and value), but also the function, that splits the string by the matching closing brace in case the value is an object itself [15, 16].

Thus top-down parsing can be performed by splitting the string [15]. So strings of the given format can be parsed. Also it provides the opportunity to use and parse the syntax constructions of the specific format. Moreover, one can develop the domain-specific language (DSL) for the particular domain and the interpreter using the LL-parser.

So the implementation of LL-parser can be used for different problems connected with formal representation of syntax constructions and development of new constructions for programming languages.

VI. Conclusions

In this paper the L-type Petri net language for the producer/consumer problem was considered. The language was proved to be nonregular by using the pumping lemma for regular languages. The pushdown automaton with the initial stack symbol, that accepts the language, was stated (we recall that another version of the pushdown automaton without the initial stack symbol was presented in [8]). The context-free grammar, that generates the considered language, was constructed. The connection with the Dyck language was illustrated, several significant corresponding derivations were given.

Thus we have shown the possibility to transform the producer/consumer Petri net into the pushdown automata, therefore to prove that the producer/consumer Petri net language is the context-free language. The obtained results will be used in further investigations of Petri net languages.

Also the proposed mechanisms can be applied in LL-parser implementation in order to deal with syntax constructions.

References

[1] J. L. Peterson, Petri net theory and the modeling of systems, Prentice Hall, 1981.

[2] T. Murata, “Petri nets: properties, analysis and applications”, Proceedings of the IEEE, vol. 77, no. 4, April 1989.

[3] A. Aho, J. D. Ullman, The theory of parsing, translation and computing. Vol. 1: Parsing, Prentice Hall, 1972.

[4] J. E. Hopcroft, R. Motvani, J. D. Ullman, Introduction to automata theory, languages and computation, 3rd ed., Addison-Wesley, 2006.

[5] A. Je. Pentus, M. R. Pentus, Theory of formal languages, Moscow State Univ., 2004 [in Russian].

[6] R. Valk, G. Vidal-Naquet, “Petri nets and regular languages”, Journal of Computer and System Sciences, vol. 23, pp. 299–325, 1981.

[7] A. Gronewold, H. Fleischhack, “Computing Petri net languages by reductions”, Fundamentals of computation theory: 10th International conference; proceedings, FCT’95, Drezden, Germany, August 22–25, 1995, Springer, pp. 253–262.

[8] V. Statkevych, “Regular expressions for producer/consumer Petri net languages with bounded buffer of size 1 and 2 [in Russian]”, System Analysis and Information Technology: 19th International conference, Kyiv, Ukraine, May 22–25, 2017, NTUU “KPI”, p. 122.

[9] M. Jantzen, G. Zetzsche, “Labeled step sequences in Petri nets”, Applications and Theory of Petri Nets: 29th International conference, Xi'an, China (June 23–27, 2008), Proceedings, pp. 270–287.

[10] J. Dassow, S. Turaev, “Petri net controlled grammars: the case of special Petri nets”, Journal of Universal Computer Science, vol. 15, no. 14, pp. 2808–2835, 2009.

[11] J. Dassow, G. Mavlankulov, M. Othman, S. Turaev, M.H. Selamat and R. Stiebe, “Grammars controlled by Petri nets”. In book: Petri nets: Manufacturing and Computer Science, pp. 337–358, 2012.

[12] I. Spectorsky, “Application of Petri nets to the analysis of context-free grammars [in Russian]”, System Research & Information Technologies, vol. 4, pp. 129–133, 2011.

[13] V. Statkevich, “Connection between Petri nets and Polish notation [in Russian]”, System Research & Information Technologies, vol. 2, pp. 7–13, 2016.

[14] V. Statkevych, “On regular expressions for producer/consumer Petri net languages with bounded buffer”, Nonlinear analysis and applications: 4th International scientific conference on memory of corresponding member of National Academy of Science of Ukraine V. S. Mel’nik, Kyiv, Ukraine, 4–6 April, 2018, NTUU “KPI”, p. 68.

[15] K. Lvov, “About LL parsing: An approach to parsing through the concept of string cutting [in Russian]”, https://habr.com/ru/post/412905

[16] R. Edelmann, J. Hamza, V. Kunčak, “LL(1) Parsing with derivatives and zippers. Efficient, functional, and formally verified approach to parsing”, arXiv:1911.12737v1 [cs.FL], 28 Nov. 2019, https://arxiv.org/pdf/1911.12737.pdf

[17] M. D. Adams, C. Hollenbeck, and M. Might, “On the complexity and performance of parsing with derivatives”, In Proc. of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI ’16). ACM, New York, NY, USA, pp. 224–236, 2016. https://doi.org/10.1145/2908080.2908128