Petri nets and random-context grammars

Vladimír Marek*
marek@dcse.fee.vutbr.cz

Milan Češka*
ceska@dcse.fee.vutbr.cz

Abstract: This paper studies relation between Petri nets, which is one of the basic models used in modelling of discrete-event systems and concurrent systems, and random-context grammars, which is a computational model used in theoretical computer science. A mapping from random-context grammars to Petri nets preserving computational sequences (languages) is presented and some of its properties are shown.

Key Words: Context-free grammar, random-context grammar, Petri net, formal language, Szilard language

1 Introduction

The concept of formal languages, one of mainstreams of the computer science, can be applied also in system modelling. This is introduced in [3].

A formal language L is defined as any subset of words (strings) over some alphabet. Let us assume that we are interested in sequences of events which generate outputs for given inputs rather than in the correspondence of these inputs and outputs. In this case we can define language L(S) as a set of all sequences of events which correspond to all possible sequences of inputs. We call L(S) the language of computational sequences; it is defined over the alphabet consisting of names of events.

Theory of formal models deals with decision and modelling power of models, and similarly, the formal language theory describes classes of languages and studies their richness (associated with the modelling power of corresponding model) and decidability of problems (aassociated with the decision power).

Here we try to find similarities between Petri nets, as a representant of theory of formal models, and random-context grammars, a special concept extending context-free grammars which is very widely used not only in formal languages theory but also in compilers, text editors, etc. Those similarities can support research on new model and compiler design.

2 Preliminaries

$\mathbf{N}$ denotes a set of natural numbers (zero included) and $\mathbf{P}(M)$ denotes a powerset, e.g. set of all subsets of set M. Let Σ be a finite set of symbols (e.g. finite alphabet). Then Σ^* denotes the set of all words $x_{i1}x_{i2}...x_{in}$ ($x_{ik} \in \Sigma$) over Σ including the empty word ε. A formal language over Σ is set $L \subseteq \Sigma^*$, *aplh(x)* denotes a set of all symbols present in x ($x \in \Sigma^*$, $aplh(x) \subseteq \Sigma$).

* Department of Computer Science and Engineering, Božetěchova 2, CZ-612 66 Brno,
http://www.fee.vutbr.cz/UIVT/

A context-free grammar (CFG) is a quadruple $G = (N_G, T_G, P_G, S)$ where N_G is a finite set of *nonterminal symbols*, T_G a finite set of *terminal symbols*, $N_G \cap T_G = \emptyset$, P_G is a finite set of *production rules* of the form $r: A \to w$; r is the label (name) of the rule, $A \in N_G$, $w \in (N_G \cup T_G)^*$; $S \in N_G$ is called a *start symbol*. $\Rightarrow$ is a direct derivation relation, $\Rightarrow \subseteq ((N_G \cup T_G)^* \times (N_G \cup T_G)^*$; $xAy \Rightarrow xwy$ $[r]$ if $r: A \to w \in P_G$ $\forall x, y \in (N_G \cup T_G)^*$; if the rule is not important, we simply write $xAy \Rightarrow xwy$. $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$ and is defined for some sequence of production rules $q \in P_G^*$: $x \Rightarrow^* y$ $[q]$ if $x = y$ (then q is the empty sequence, $q = \varepsilon$) or $\exists z \in (N_G \cup T_G)^*$, $\exists r \in P_G$ and $\exists q_{zy} \in P_G^*$ such that $x \Rightarrow z$ $[r]$ and $z \Rightarrow^* y$ $[q_{zy}]$ and $q = rq_{zy}$. If the sequence of rules is not important, we simply write $x \Rightarrow^* y$.

Language generated by G, $L(G) = \{w: S \Rightarrow^* w$ and $w \in T_G^*\}$, is called a context-free language (CFL).

The language of the production rules sequences that lead to the derivation of a terminal word is called a Szilard language of the grammar, $Sz(G) = \{q: S \Rightarrow^* w$ $[q]$ and $w \in T_G^*\}$.

2.1 Random-context grammars

A *random-context* grammar, shortly RCG, (sometimes also called a *permitting grammar*) is a pair $H = (G, R)$, where $G = (N_G, T_G, P_G, S)$ is a context-free grammar and $R \subseteq P_G \times N_G$ is a finite relation from set of production rules P_G to set of nonterminals N_G.

For $x, y \in (N_G \cup T_G)^*$, $r \in P_G$:

$x \Rightarrow y$ $[r]$ in H, or more shortly $x \Rightarrow y$ in H,

if

1) $x \Rightarrow y$ $[r]$ in G, and
2) $R(r) \subseteq alph(x)$

If $R = \emptyset$, H is exactly a context-free grammar since the second condition is always satisfied. It can be seen that the only difference between a CFG and RCG is in the definition of $\Rightarrow$.

Similarly as in CFG, $\Rightarrow^*$ is the transitive and reflexive closure of $\Rightarrow$, and the language generated by a RCG H is defined as follows

$$L(H) = \{w: S \Rightarrow^* w \wedge w \in T_G^*\}$$

Such language is called a random-context language (RCL).

Similarly, Szilard language is defined as $Sz(H) = \{r: S \Rightarrow^* w$ $[r]$ and $w \in T_G^*\}$.

<u>*Note 1:*</u> If $r: A \to x \in P_G$, $R(r) = Q$, for simplification we write $(r: A \to x, Q) \in P_G$ where $Q \subseteq N_G$ is called a *permitting context*.

<u>*Note 2:*</u> If $(r: A \to x, Q) \in P_G$ and $A \in Q$ ... such production rule works the same way as $(r: A \to x, Q \backslash \{A\})$ since A must be present in the current word for r to be applicable for the left side of r is A. For purposes of this paper this possible ambiguity is not suitable, we rather prefer the form where $A \notin Q$ [A itself is not a member of the permitting context].

Example

$N_G = \{S, A, B, C, X, Y, Z\}$

$T_G = \{a, b, c\}$

$P_G = \{ (r_1: S \to ABC, \emptyset),$

$(r_2: A \to aX, \{B\}),$

$(r_3: B \to bY, \{C\})$,
$(r_4: C \to cZ, \{X\})$,
$(r_5: X \to A, \{Y\})$,
$(r_6: Y \to B, \{Z\})$,
$(r_7: Z \to C, \{A\})$,
$(r_8: A \to a, \{B\})$,
$(r_9: B \to b, \{C\})$,
$(r_{10}: C \to c, \emptyset)\}$

Some terminal word derivations:

a) $S \Rightarrow ABC\ [r_1] \Rightarrow aBC\ [r_8] \Rightarrow abC\ [r_9] \Rightarrow \mathbf{abc}\ [r_{10}]$

$abc \in L(H)$, $r_1 r_8 r_9 r_{10} \in Sz(H)$

b) $S \Rightarrow ABC\ [r_1] \Rightarrow aXBC\ [r_2] \Rightarrow aXbYC\ [r_3] \Rightarrow aXbYcZ\ [r_4] \Rightarrow$
$\Rightarrow aAbYcZ\ [r_5] \Rightarrow aAbBcZ\ [r_6] \Rightarrow aAbBcC\ [r_7] \Rightarrow$
$\Rightarrow aabBcC\ [r_8] \Rightarrow aabbcC\ [r_9] \Rightarrow \mathbf{aabbcc}\ [r_{10}]$

$aabbcc \in L(H)$, $r_1 r_2 r_3 r_4 r_5 r_6 r_7 r_8 r_9 r_{10} \in Sz(H)$

The language generated by this random-context grammar, $L(H) = \{a^i b^i c^i : \ i \geq 1\}$, is not context-free, e.g. cannot be generated by any context-free grammar. Hence we see that the computational power of random-context grammars is greater than that of context-free grammars.

Let $\mathbf{L}$(*grammar_type*) denote a class of languages generated by the grammar type. Thus $\mathbf{L}(CF)$ is a class of context-free languages (generated by CFG's) which match the class 2 of the Chomsky hierarchy, $\mathbf{L}(RC)$ a class of random-context languages (generated by RCG's) and $\mathbf{L}(CS)$ a class of context-sensitive languages (generated by context-sensitive grammars, CSG's) which match the class 1 of the Chomsky hierarchy.

It follows the definition that RCG's is a generalization of CFG's. Hence, $\mathbf{L}(CF)$ is a proper subclass of the class of $\mathbf{L}(RC)$. It is well known that $\mathbf{L}(RC)$ is a proper subclass of $\mathbf{L}(CS)$. This is given without a proof.

$$\mathbf{L}(CF) = \mathbf{L}(2) \subset \mathbf{L}(RC) \subset \mathbf{L}(CS) = \mathbf{L}(1)$$

2.2 Parikh mapping

Let V be a finite alphabet, $|V|$ the number of its elements, and w a word, $w \in V^*$. For some fixed order of symbols of V, there exists a mapping $\psi: V^* \to \mathbf{N}^{|V|}$ such that $\psi(w) = (n_1, n_2, ..., n_{|V|})$, where n_i is the number of occurences of the i-th symbol of alphabet V in word w. Mapping ψ is called a *Parikh mapping*. To access single values of the n-tuple we will use the following notation: $\psi(w)_X$ is the number of occurences of symbol X in word w.

Example

$V = \{a, b, c\}$
$\psi(\varepsilon) = (0, 0, 0)$
$\psi(cac) = (1, 0, 2)$
$\psi(abba) = (2, 2, 0)$

$\psi(aabb) = (2, 2, 0)$
$\psi(bacca) = (2, 1, 2)$
etc.

It is easy to see that Parikh mapping over alphabet V is not a bijection since it is surjective but not injective. In this paper we will use the Parikh mapping later, for mapping words over an alphabet to states of a Petri net.

2.3 Petri nets

A Petri net is a quadruple $PN = (P_{PN}, T_{PN}, F, B)$, where P_{PN} is a finite *set of places*, T_{PN} a finite *set of transitions* such that $P_{PN} \cap T_{PN} = \varnothing$; $F: P_{PN} \times T_{PN} \to \mathbf{N}$ is called a *forward incidence function* which specifies weights of arcs going from places to transitions, and $B: T_{PN} \times P_{PN} \to \mathbf{N}$ a *backward incidence function* which specifies weights of arcs going from transitions to places. In the graphical form we draw *places as circles*, *transitions as boxes* and *arcs* (only those weighted 1 and more) *as arrows*.

Let $\Delta: T_{PN} \times P_{PN} \to \mathbf{N}$ be defined as $\forall p \in P_{PN}, \forall t \in T_{PN}: \Delta(t, p) = B(t, p) - F(p, t)$.

A marking m of PN is a mapping $m: P_{PN} \to \mathbf{N}$, or, it assigns number of tokens to each place. Initial marking of a PN is usually denoted as m_0. Tokens are depicted as dots or numbers in places.

Let $PN = (P_{PN}, T_{PN}, F, B)$ be a Petri net and m its marking. We say that transition $t \in T_{PN}$ is *enabled* in marking m iff $\forall p \in P_{PN}: m(p) \geq F(p, t)$. A transition which is enabled can be fired. By *firing* transition t, the marking of the net changes to m', $\forall p \in P_{PN}: m'(p) = m(p) + \Delta(t, p)$, and we write $m\,[t\rangle\, m'$.

Firing can be extended from single transitions to their sequences, then for the empty sequence $m\,[\varepsilon\rangle\, m$. Hence $m\,[\rangle\, m''$, iff $m = m''$ (for the empty sequence) or there exist $t \in T_{PN}$, $s \in T_{PN}^*$, and marking m' such that $m\,[t\rangle\, m'$ and $m'\,[s\rangle\, m''$. Then we also write $m\,[ts\rangle\, m''$.

Given the initial marking m_0 and set Q_F of final markings we define the language of the Petri net, $L(PN) = \{s: s \in T_{PN}^* \text{ and } \exists m \in Q_F \text{ such that } m_0\,[s\rangle\, m\}$.

3 Algorithm construction

Here we will describe the most substantial part of this paper, an algorithm that transforms a RCG to a Petri net. For each derivation in RCG we can find a sequence of transitions in the Petri net such that at any time instant the current word is mapped to the marking of the Petri net by the Parikh mapping ψ (defined over the alphabet of nonterminals and terminals).

3.1 The algorithm specification

Input: a random-context grammar $H = (G, R)$, where $G = (N_G, T_G, P_G, S)$ and $R \subseteq P_G \times N_G$

Output: a Petri net $PN = (P_{PN}, T_{PN}, F, B)$ and its initial marking m_0.

Steps:

1) $P_{PN} = \{X': X \in (N_G \cup T_G)\}$,
2) $T_{PN} = \{r': r \in P_G\}$,
3) F and B are defined as follows

   $\forall r \in P_G$, assume r of the form $r: A \to w$, and let $Q = R(r)$:

$\forall X \in (N_G \cup T_G)$:

a) $F(X', r') = |\{X\} \cap (Q \cup \{A\})|$

b) $B(r', X') = \psi(w)_X + |\{X\} \cap (Q \setminus \{A\})|$

4) $m_0 = \psi(S)$; ψ is defined over $N_G \cup T_G$.

Steps 1 and 2 form mappings $h_1: (N_G \cup T_G) \to P_{PN}$ and $h_2: P_G \to T_{PN}$. Both h_1 and h_2 are bijections. Their union, $h = h_1 \cup h_2$ is also a bijection which to each symbol or production rule of grammar assigns its dashed version (a place or transition of the Petri net). Later we will use also a generalization H_2 of h_2; defined on sets, $H_2: \mathbf{P}(P_G) \to \mathbf{P}(T_{PN})$; $H_2(U) = \{h_2(r): r \in U\}$, where $U \subseteq P_G$, then $H_2(U) \subseteq T_{PN}$.

This algorithm can be used also for context-free grammars, since a CFG is just a special case of a RCG with $R(r) = \varnothing$ [e.g. $Q = \varnothing$] for all production rules r.

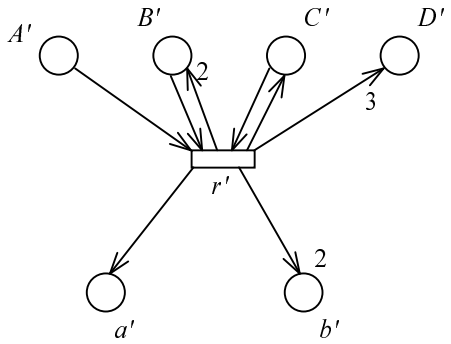
Informally:

1) *Places* of PN correspond with the *symbols* (terminal and nonterminal) of the grammar and are named the same way, with the dashes added.
2) *Transitions* of PN correspond with the *production rules* of the grammar and are named the same way, with the dashes added.
3) For each production rule ($r: A \to w, Q$) create [assume $A \notin Q$]:
   a) arcs weighted 1 from A' to r' and for each nonterminal $B \in Q$ from B' to r' [current word must contain A and all nonterminals of Q for r to be applicable].
   b) for each symbol (terminal or nonterminal) X in w an arc from r' to X' weighted by the number of occurrences of X in w [by applying r on the current word, the symbols of w are added into it], further an arc for each nonterminal $B \in Q$ from r' to B' [in a) we just check the presence of these nonterminals in the current word, so now we have to put them back].

   An arc which is created more times is weighted by the sum of the appropriate weights.
4) The initial marking is the marking where the number of tokens in place S' is 1, the number of tokens in the other places is 0.

3.2 Examples

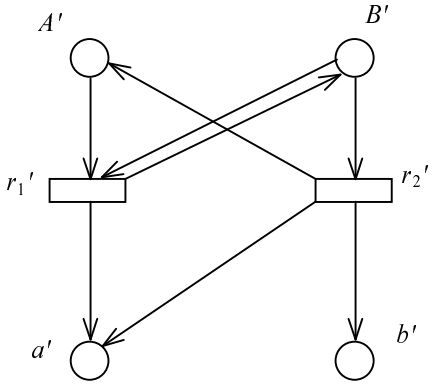
a) Single production rule ($r: A \to DabBbDD, \{B, C\}$):



or algebraically (nonzero entries are **bold**),

	A'	B'	C'	D'	a'	b'
$F(r') =$	(**1**,	**1**,	**1**,	0,	0,	0)
$B(r') =$	(0,	**2**,	**1**,	**3**,	**1**,	**2**)

b) Two production rules (r_1: $A \to a$, $\{B\}$), (r_2: $B \to abA$, $\varnothing$).



	A'	B'	a'	b'
$F(r_1') =$	(**1**,	**1**,	0,	0)
$F(r_2') =$	(0,	**1**,	0,	0)
$B(r_1') =$	(0,	**1**,	**1**,	0)
$B(r_2') =$	(**1**,	0,	**1**,	**1**)

This is not a complete grammar (e.g. with start symbol), we do not compute the initial marking m_0.

c) Complete grammar: the grammar taken from the example in section 2.1 .

$PN = (P_{PN}, T_{PN}, F, B)$

$P_{PN} = \{S', A', B', C', X', Y', Z', a', b', c'\}$

$T_{PN} = \{r_1', r_2', r_3', r_4', r_5', r_6', r_7', r_8', r_9', r_{10}'\}$

	S'	A'	B'	C'	X'	Y'	Z'	a'	b'	c'
$F(r_1') =$	(**1**,	0,	0,	0,	0,	0,	0,	0,	0,	0)
$F(r_2') =$	(0,	**1**,	**1**,	0,	0,	0,	0,	0,	0,	0)
$F(r_3') =$	(0,	0,	**1**,	**1**,	0,	0,	0,	0,	0,	0)
$F(r_4') =$	(0,	0,	0,	**1**,	**1**,	0,	0,	0,	0,	0)
$F(r_5') =$	(0,	0,	0,	0,	**1**,	**1**,	0,	0,	0,	0)
$F(r_6') =$	(0,	0,	0,	0,	0,	**1**,	**1**,	0,	0,	0)
$F(r_7') =$	(0,	**1**,	0,	0,	0,	0,	**1**,	0,	0,	0)
$F(r_8') =$	(0,	**1**,	**1**,	0,	0,	0,	0,	0,	0,	0)
$F(r_9') =$	(0,	0,	**1**,	**1**,	0,	0,	0,	0,	0,	0)
$F(r_{10}') =$	(0,	0,	0,	**1**,	0,	0,	0,	0,	0,	0)

$B(r_1') =$	(0,	**1**,	**1**,	**1**,	0,	0,	0,	0,	0,	0)
$B(r_2') =$	(0,	0,	**1**,	0,	**1**,	0,	0,	**1**,	0,	0)
$B(r_3') =$	(0,	0,	0,	**1**,	0,	**1**,	0,	0,	**1**,	0)
$B(r_4') =$	(0,	0,	0,	0,	**1**,	0,	**1**,	0,	0,	**1**)
$B(r_5') =$	(0,	**1**,	0,	0,	0,	**1**,	0,	0,	0,	0)
$B(r_6') =$	(0,	0,	**1**,	0,	0,	0,	**1**,	0,	0,	0)
$B(r_7') =$	(0,	**1**,	0,	**1**,	0,	0,	0,	0,	0,	0)
$B(r_8') =$	(0,	0,	**1**,	0,	0,	0,	0,	**1**,	0,	0)
$B(r_9') =$	(0,	0,	0,	**1**,	0,	0,	0,	0,	**1**,	0)
$B(r_{10}') =$	(0,	0,	0,	0,	0,	0,	0,	0,	0,	**1**)
$m_0 =$	(**1**,	0,	0,	0,	0,	0,	0,	0,	0,	0)

The graphical form of this Petri net is not presented here and is left to reader as an exercise.

3.3 Properties

Let H be a random context grammar and PN a Petri net created from H using the algorithm specified and m_0 its initial state. Let $q = r_{i1}, r_{i2}, ..., r_{in}$ be a sequence of production rules and $w_{i1}, w_{i2}, ..., w_{in}$ a sequence of words over (terminal and nonterminal) symbols such that $S \Rightarrow w_{i1}\ [r_{i1}]$, $w_{i1} \Rightarrow w_{i2}\ [r_{i2}]$, ..., $w_{in-1} \Rightarrow w_{in}\ [r_{in}]$, and w_{in} contains only terminal symbols. For simplification we denote S as w_{i0}. At any time instant we may apply some production rules and some not (depending on nonterminals appearance in the current word w_{ik}), let us denote APR_k a set of applicable production rules in step k. It can be seen that

$$\forall k,\ 1 \le k \le n\text{: } r_{ik} \in APR_k.$$

This derivation process may be simulated by Petri net PN in the sense that $m_0, m_1, ..., m_n$ is the sequence of markings of PN such that

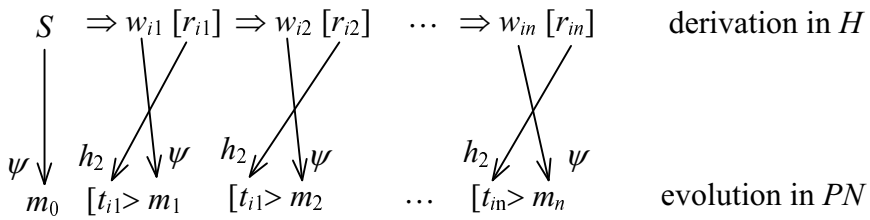
$$\forall k,\ 0 \le k \le n\text{: } m_k = \psi(w_{ik})$$

and $s = t_{i1}, t_{i2}, ..., t_{in}$ sequence of fired transitions of PN such that $m_0\ [t_{i1}\rangle\ m_1$, $m_1\ [t_{i2}\rangle\ m_2$, ..., $m_{n-1}\ [t_{in}\rangle\ m_n$. Let us denote ET_k a set of enabled transitions in step k, clearly $t_{ik} \in ET_k$. The algorithm is constructed so that ET_1 of PN is equal to $H_2(APR_1)$, informally: the set of enabled transitions is the same as the set of transitions to which are mapped the applicable production rules of the grammar. By choosing and applying a production rule r_{i1} from APR_1 the current word changes from S to w_{i1} and we get new set APR_2. If we fire the transition corresponding to r_{i1}, that means t_{i1}, the state of PN changes so that new set of enabled transitions ET_2 is again equal to $H_2(APR_2)$. This process continues until no production rule is applicable, e.g. $APR_n = \varnothing$, and similarly in PN, no transition is enabled, e.g. $ET_n = \varnothing$. Then, we assume $\forall k,\ 1 \le k \le n$: $t_{ik} = h_2(r_{ik})$.

We may write:

$$\forall k,\ 0 \le k \le n\text{: if } w_{ik} \Rightarrow w_{ik+1}\ [p_{k+1}] \text{ in } H \text{ then } \psi(w_{ik})\ \ [h_2(p_{k+1})\rangle\ \ \psi(w_{ik+1}) \text{ in } PN$$

Graphically:



If we want to compare two languages, the language $Sz(H)$ of all possible sequences $r_{i1}r_{i2}...r_{in}$ of production rules of grammar H, leading to a terminal word w_n, and the language $L(PN)$ of all possible sequences $t_{i1}t_{i2}...t_{in}$ leading to such marking of the Petri net, where all places corresponding with nonterminals are empty [$m(t') = 0\ \forall t \in N_G$], we state that the languages are **isomorphic** according to mapping h_2.

In the process of the Petri net creation, not the exact right side of each production rule (e.g. word) but only its Parikh image is used; therefore if we permute the right sides of the production rules, the Petri net will be still the same. Therefore there may be more different grammars whose corresponding Petri net is same.

4 Conclusions

We have investigated that the derivation process in random-context grammars is similar to the firing sequences in Petri nets. We have formulated the algorithm that transforms any random-context grammar to the corresponding Petri net. These assertions hold also for context-free grammars since they are just a special case of random-context grammars.

The mapping from a grammar to a Petri net is not bijective since there can be more different grammars that are mapped to the same Petri net. But, those grammars are equivalent as for the possible sequences of the production rules that lead to a terminal word. These sequences are also isomorphic to the language of the Petri net.

This work has been done within the project CEZ:J22/98: 262200012 "Research in Information and Control Systems" and also supported by the Grant Agency of Czech Republic grants No. 102/01/1485 "Environment for Development, Modelling, and Application of Heterogeneous Systems".

References

1. HAUSCHILDT, D., JANTZEN, M. Petri Net Algorithms in the Theory of Matrix Grammars. *Acta Informatica*. Berlin, Heidelberg: Springer Verlag, 1994, Vol. 31, No. 8, pp. 719–728. ISSN 0001-5903 (print version), 1432-0525 (electronic version).
2. CRESPI-REGHIZZI, S. Petri Nets and Szilard Languages. *Information and Control*. New York, London: Academic Press, February 1977, Vol. 33, No. 2, pp. 177–192.
3. ČEŠKA, M. *Cybernetics and Concepts of Theory of Formal Languages*. Las Palmas de Gran Canaria: Universidad de Las Palmas, 1999, pp. 57–59. ISBN 84-8416-950-2.