Journal of Computer Science 3 (7): 494-498, 2007
ISSN 1549-3636


# Removing Useless Productions of a Context Free Grammar through Petri Net

Mansoor Al-A'ali and Ali A Khan
Department of Computer Science, College of Information Technology, University of Bahrain
P.O. Box 32038, Sakheer Campus, Kingdom of Bahrain

**Abstract:** Following the proposal for a Petri Net (PN) representation of the Context Free Grammar (CFG)[1], we propose in this paper, an algorithm to eliminate the useless productions of CFG. First the CFG is represented by a PN. Then, based on the reachability, an algorithm is developed to eliminate Useless-productions. The algorithm is analyzed and implemented in Pascal using examples of a CFG. The proposed algorithm is better than the existing techniques in the sense that PN model is easy to understand and requires fewer computations and easily implemented on computers.

**Key words:** Formal methods, context- free grammar, useless productions, petri net

## INTRODUCTION

Considerable interest has been shown in Petri Nets (PNs) as suitable models for representing and studying concurrent systems because of its potential advantages. The major use of Petri Net has been the modeling of systems in which it is possible for some events to occur concurrently and or sequentially. They have been found to be useful for describing and analyzing many systems such as production, automatic control, communication protocol, circuit analysis, physics and systems involving human activity such as management and office information systems, legal systems, teaching and knowledge representation[2-6]. In order to allow easy verification and hierarchical decomposition of the system, the reduction method of PNs without changing the properties has also been studied[7]. However, there are still some areas where either this versatile model has not been used at all or used to a very limited extent. Formal languages and automata theory is one of such areas. Algorithms to minimize the context free grammars using the Petri net concept have been published[1,8,9] in which a petri net representation of the CFG has been given and techniques to eliminate λ and unit- productions have been provided. The object of this paper is to exploit this PN model to remove Useless-productions of a Context-Free Grammar (CFG). We assume that λ- and unit- productions have already been removed. First the CFG is represented by a PN. Then algorithm is developed to eliminate Useless-productions. The algorithm is analyzed and implemented in Pascal. using examples of a CFG . The proposed technique is novel in the sense that it provides a better representation and understanding of the problem. It is simple, requires less computation and is easily implemented on computers. Another technique proposed in[3] in which a given finite or infinite labeled transition graphs defined by a graph grammar, in which an algorithm decides whether this graph is isomorphic to the reachable state graph of some finite unlabeled Petri net. The algorithm aims to produce a minimal net realizing the graph. In[10] the Petri net models are established for right linear grammar, expression grammar and property tree formal grammar. The structure algorithms of these models are given and the properties of models are discussed. It is aimed at generating a language process based on Petri net models. These models are intuitional and dynamic

**Petri net:** A Petri net is an abstract, formal model of information flow[6]. PNs are composed of two basic components: a set of places, P and a set of transitions, T. The relationship between the places and the transitions are defined by the input and the output functions. The input function I defines, for each transition $t_j$, the set of input places for the transition $I(t_j)$. The output function O defines, for each transition $t_j$, the set of output places for the transition $O(t_j)$.

Formally, a Petri net C is defined as the four-tuple C = (P, T, I, O), where

$P = \{ p_1, p_2, ....., p_n\}$ is a finite set of places

$T = \{ t_1, t_2, ....., t_m\}$ is a finite set of transitions, $P \cap T = \emptyset$,

$I = P \rightarrow T$,

$O = T \rightarrow P$.

**Corresponding Author:** Mansoor Al-A'ali, Department of Computer Science, College of Information Technology, University of Bahrain, P.O. Box 32038, Sakheer Campus, Kingdom of Bahrain

A Petri net graph is commonly used for better illustration. It consists of two types of nodes: a circle which represents a place and a bar which represents a transition. The input and output functions are represented by directed arcs from the places to the transitions and from the transitions to the places. An arc is directed from a place $p_j$ to a transition $t_i$ if the place is an input of the transition. Similarly, an arc is directed from a transition $t_j$ to a place $p_i$ if the place is an output of the transition.

To give a dynamic structure to PN, a marking $\mu$ is an assignment of tokens (represented by small solid dots inside the circles) to places. A Petri net executes by firing transitions. A transition is enabled to fire if each of its input places has at least one token in it. A transition fires by removing one token from each of its input places and depositing one token into each of its output places. Firing a transition will in general change the marking $\mu$ of the Petri net to a new marking $\mu'$.

**Context free grammar:** Definition[11]: Let $G = (V_N, V_T, S, F)$ be a grammar where $V_N$ is a finite set of variables, $V_T$ is a finite set of objects called terminal symbols, $S \in V_N$ is a special symbol called the start variable and F is a finite set of productions. Then G is said to be context-free if productions in P have the form $A \rightarrow x$, where $A \in V_N$ and $x \in (V_N \cup V_T)^*$.

A context free grammar may have $\lambda$ production, or unit productions. These productions make the grammar odd and difficult to parse. The useless productions have unnecessary variables and productions. Therefore eliminating these productions will make the grammar easier.

**Theorem**[11]**:** Let L be a context-free language that does not contain $\lambda$. Then there exists a context-free grammar that generates L and that does not have any useless productions.

**Removing useless productions**[8]**:** An equivalent grammar that does not contain any useless variables or productions can be obtained by first finding those variables which lead to terminal strings. This is done by defining the following sets of variables

$A_o = \{X \mid X \rightarrow P \in F \text{ and } P \in V_T^*\}$

$A_i = A_{i-1} \cup \{X \mid X \rightarrow W \in F \text{ and } W \in (V_T \cup A_{i-1})^*\}$

For some k for which $Ak_{-1} = A_k$, the set of active variables $A_k$ is obtained.

Then only those variables which can be reached from starting symbol are obtained by defining and finding

$R_o = \{S\}$

$R_i = R_{i-1} \cup \{Y \mid X \rightarrow UYW \in F \text{ for some } X \in R_{i-1} \text{ and } U, W \in (V_N \cup V_T)^*\}$

For some m, $Rm_{-1} = R_m$, the set of reachable variables $R_m$ is obtained.

After that we eliminate all variables which do not belong in $A_k \cap R_m$ together with all rules in which they occur. The same process is repeated with the resulting grammar until we get a non-redundant grammar.

**PN Representation of a CFG**[1]**:** A Context Free Grammar can be thought of as represented by interconnections of transitions and places of a PN. The productions are represented by transitions and the variables and terminal symbols by places. To include the order of appearance of variables and symbols on the right side of a production, the Petri net is modified and called an Ordered Petri Net (OPN).

**Definition**[10]: Let $G = (V_N, V_T, S, F)$ be a Context - Free Grammar. An Ordered Petri Net is a PN defined to represent G if and only if it has the following properties:

* The input place of PN is labeled as S.
* The output places are labeled from $V_T \cup \{\lambda\}$.
* The intermediate places have labels from $V_N$.
* Production rules are represented by transitions. The input of the transition $t_j$ has label $A \in V_N$ and outputs of $t_j$ are (from left to right ) labeled as $a_1$ , $a_2$, ...., $a_n$ if and only if F has a production of the form $A \rightarrow a_1, a_2, ...., a_n$.

The string of output places with tokens by reading from left to right omitting any $\lambda$'s encountered is called the yield of OPN.

It must be noted here that there will be exactly one input place for any transition. Two or more productions may have the same variable on the left side. Only one of these productions can be used in one derivation. Therefore, the corresponding transitions can not fire simultaneously. This is guaranteed by having the same place input to such transitions. To derive a sentence, it is sufficient to find the sequence of the transitions which must fire such that the token of the starting symbol place reaches the required output places of leaf nodes.

## PROPOSED TECHNIQUE FOR REDUCTION OF USELESS PRODUCTIONS

Before developing the algorithm, some notations that require explanation and clear exposition are given below:

P is the set of places corresponding to the variables and terminals. $|P| = |(V_N \cup V_T)|$

$P_N$ is the set of places corresponding to the variables.

$P_T$ is the set of places corresponding to the terminals.

T is the set of transitions corresponding to the productions. |T| is the number of productions.

$t_i^o$ is the set of output places of the transition $t_i$. It must be noted that the order of the places in $t_j^o$ is important and must be in the same sequence in which they appear in the corresponding production rule. Further it must be maintained in subsequent operations.

$^ot_i$ is the set of input places of the transition $t_i$. Obviously this will contain only one element.

$p_i^o$ is the set of output transitions of the place $p_i$

$^op_i$ is the set of input transitions of the place $p_i$

$P^o = \{ p_1^o , p_2^o ,\cdots, p_n^o\}$; $^oP=\{ {}^op_1 , {}^op_2, \cdots, {}^op_n \}$

$T^o = \{ t_1^o , t_2^o ,\cdots, t_n^o\}$; $^oT=\{ {}^ot_1 , {}^ot_2, \cdots, {}^ot_n \}$

There are two types of useless productions; one is the production having their variables on the left side which cannot be reached from the starting symbol variables which do not lead to a terminal string. In the second type of productions the set of variables U which lead to terminal strings can be obtained by using reachability in PN. A place $p_i$ is in U iff a marking with a token in $p_i$ only can reach a marking with tokens only in leaf places by firing transitions. The second type of productions can be eliminated by finding the set of variables R which can be reached from starting symbol. A place $p_i$ is in R iff a token in S can reach to pi on firing transitions. Only those transitions, places given by a U $\cap$R along with their interconnections are included into resulting PN.

**Algorithm**

**Input:** OPN representing CFG with useless productions

**Output:** A reduced OPN for CFG without useless productions

**Steps:**

{Steps 1-4 finds the set of variables U which lead to terminal strings}

* $U_0 = \varnothing; T_0 = \varnothing$ ; i = 0
* $\forall t_j \in T$ , if $t_j^o \subseteq (U_i \cup P_T)$ then
  $U_{i+1} = U_i \cup {}^ot_j$
  $T_{i+1} = T_i \cup \{ t_j\}$
* If $U_{i+1} \neq U_i$ then go to step 2.
* $U = U_i$ ; $T'_1 = T_{i+1}$

{Steps 5-8 find the set of variables R which can be reached from starting symbol}

* $R_0 = \{S\}$; $T_0 = \varnothing$ ; i = 0
* $\forall t_j \in T'_1$ if $^ot_j \subseteq R_i$ then
  $R_{i+1} = R_i \cup ( t_j^o \cap P_N)$
  $T_{i+1} = T_i \cup \{ t_j\}$
* If $R_{i+1} \neq R_i$i. then go to step 6
* $P'_N = U \cap R_i$ ; $T'_N = T_{i+1}$
* {This finds the reduced OPN} $t'_i \in T'$ $^ot'_i = t_i^o$; $^ot'_i = t_i^o$ iff

$\forall t_i \in T'_N$ $^ot_i = {}^ot_i \cap P'_N$ ; $t_i^o = t_i^o \cap ( P'_N \cup P_T )$.The order in $t_i^o$ is to be maintained as before.

The algorithm can easily be shown to be O ($|T|^2$) where $|T|$ is the number of productions.

The algorithm is illustrated with the help of the following example, example 1.

**Example 1:** Consider the Grammar G = ({A, B, C, S}, {a, b, c}, S, F), with the following production rules:

S → aB | bC; A → BAc | bSC | a; B → aSB | bBC; C → SBC | SBc | ac

The Petri net presentation is shown in Fig. 1

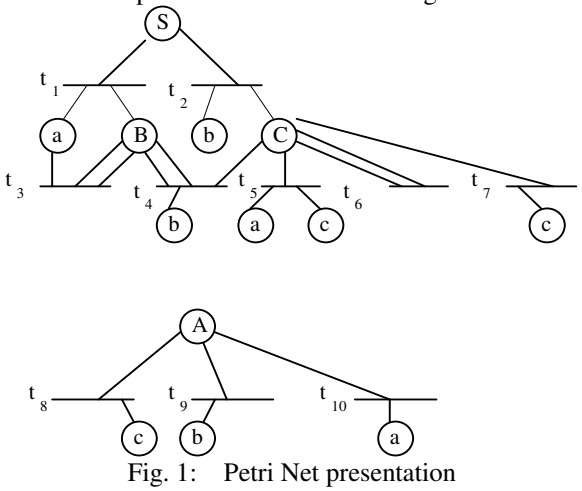


Fig. 1: Petri Net presentation

For this example,
$P_N = \{S, A, B, C\}$ ; $P_T = \{a, b, c\}$; $T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$

${}^{o}t_1 = \{S\}$, $t_1{}^{o} = \{a, B\}$

${}^{o}t_2 = \{S\}$, $t_2{}^{o} = \{b, C\}$

${}^{o}t_3 = \{B\}$, $t_3{}^{o} = \{a, S, B\}$

${}^{o}t_4 = \{B\}$, $t_4{}^{o} = \{b, B, C\}$

${}^{o}t_5 = \{C\}$, $t_5{}^{o} = \{a, c\}$

${}^{o}t_6 = \{C\}$, $t_6{}^{o} = \{S,B,C\}$

${}^{o}t_7 = \{C\}$, $t_7{}^{o} = \{S,B,c\}$

${}^{o}t_8 = \{A\}$, $t_8{}^{o} = \{B, A, c\}$

${}^{o}t_9 = \{A\}$, $t_9{}^{o} = \{b, S, C\}$

${}^{o}t_{10} = \{A\}$, $t_{10}{}^{o} = \{a\}$

**Steps 1-4 gives the following:**

$U_0 = \varnothing; T_0 = \varnothing$ ; $i = 0$

$t_1{}^{o}$, $t_2{}^{o}$, $t_3{}^{o}$ and $t_4{}^{o} \subseteq (U_0 \cup P_T)$ hence $U_0 = \varnothing; T_0 = \varnothing$

$t_5{}^{o} = \{a, c\} \subseteq (U_0 \cup P_T)$ hence $U_1 = \{{}^{o}t_5\}$; $U_1 = \{C\}; T_1 = \{t_5\}$

$t_6{}^{o}$, $t_7{}^{o}$ , $t_8{}^{o}$ and $t_9{}^{o} \subseteq (U_1 \cup P_T)$ hence $U_1 = \{C\}; T_1 = \{t_5\}$

$t_{10}{}^{o} = \{a\} \subseteq (U_1 \cup P_T)$ hence $U_2 = \{C,A\}; T_2 = \{t_5, t_{10}\}$

$t_1{}^{o}$, $t_2{}^{o}$, $t_3{}^{o}$ and $t_4{}^{o} \subseteq (U_0 \cup P_T)$

$t_1{}^{o} \subseteq (U_2 \cup P_T)$ hence $U_2 = \{C,A\}; T_2 = \{t_5, t_{10}\}$

$t_2{}^{o} \subseteq (U_2 \cup P_T)$ hence $U_3 = \{C,A\}; T_3 = \{t_2, t_5, t_{10}\}$

$t_3{}^{o}$ and $t_4{}^{o} \subseteq (U_3 \cup P_T)$ hence $U_3 = \{C,A\}; T_3 = \{t_2, t_5, t_{10}\}$

$t_5{}^{o} \subseteq (U_3 \cup P_T)$ hence $U_4 = \{C,A\}; T_4 = \{t_2, t_5, t_{10}\}$

$t_6{}^{o}$, $t_7{}^{o}$ and $t_8{}^{o} \subseteq (U_4 \cup P_T)$ hence $U_4 = \{C,A\}; T_4 = \{t_2, t_5, t_{10}\}$

$t_9{}^{o} \subseteq (U_4 \cup P_T)$ hence $U_4 = \{C,A,S\}; T_4 = \{t_2, t_5, t_9, t_{10}\}$

$t_{10}{}^{o} \subseteq (U_4 \cup P_T)$ hence $U_5 = \{C,A,S\}; T_5 = \{t_2, t_5, t_9, t_{10}\}$

Since $U_5 = U_4 = \{C,A,S\}$ hence $U = \{C,A,S\}$; $T'_1 = T_5 = \{t_2, t_5, t_9, t_{10}\}$

**Steps 5-8 gives the following:**

$R_0 = \{S\}$; $T_0 = \varnothing$ ; $i = 0$; $T'_1 = \{t_2, t_5, t_9, t_{10}\}$ $P_N = \{S, A, B, C\}$

${}^{o}t_2 = \{S\} \subseteq R_0$ hence $R_1 = R_0 \cup (t_2{}^{o} \cap P_N)$

$= \{S\} \cup (\{b, C\} \cap \{S, A, B, C\}) = \{S,C\}$

$T_1 = \{t_2\}$

${}^{o}t_5 \subseteq R_1$ hence $R_2 = R_1 \cup (t_5{}^{o} \cap P_N)$

$= \{S,C\} \cup (\{a, c\} \cap \{S, A, B, C\}) = \{S,C\}$

$T_2 = \{t_2, t_5\}$

${}^{o}t_9$ and ${}^{o}t_{10} \subseteq R_2$ ; $R_2 = \{S,C\}$; $T_2 = \{t_2, t_5\}$

Since $R_2 = R_1$ hence $P'_N = U \cap R_i = \{S,C\}$ and $T'_N = T_{i+1} = \{t_2, t_5\}$

**Step 9 gives:**

$\forall t_i \in T'_N = \{t_2, t_5\}$

${}^{o}t_2 = {}^{o}t_2 \cap P'_N = \{S\} \cap \{S,C\} = \{S\}$;

$t_2{}^{o} = t_2{}^{o} \cap (P'_N \cup P_T) = \{b, C\} \cap (\{S,C\} \cup \{a, b, c\}) = \{b,C\}$

${}^{o}t_5 = {}^{o}t_5 \cap P'_N = \{C\} \cap \{S,C\} = \{C\}$

$t_5{}^{o} = t_5{}^{o} \cap (P'_N \cup P_T) = \{a,c\} \cap (\{S,C\} \cup \{a, b, c\}) = \{a,c\}$

Reduced Petri net is shown in Fig. 2 and the corresponding reduced grammar is

$S \rightarrow bC$; $C \rightarrow ac$

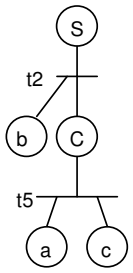


Fig. 2: Reduced Petri net

## CONCLUSION

In this paper an algorithm to eliminate the useless productions of CFG was given. It is based on the PN representation of a CFG and reachability concept of PN. The technique has been analyzed and implemented. The proposed algorithm is better than the existing techniques in the sense that PN model is easy to understand, requires less computations and easily amenable on computers. The algorithm accepts the required PN form of a CFG without λ and unit-

productions, removes the useless productions and gives the result in PN form. The automated translation of a CFG into PN has not been provided, only a technique has been given.

## REFERENCES


1. Khan, A.A., M. Al-A'ali and N. Al-Shamlan, 1996. Simplification of Context- Free Grammar Through Petri Net. Computers & Structures, 58: 1055-1058.
2. Peterson, J.L., 1981. Petri Net Theory and the Modeling of Systems. Prentice Hall Inc.
3. Rennes, D.P., 2001. On the Petri net realization of context-free graphs. Theoretical Computer Sci., 258: 573-98.
4. Tilak, A., 1979. Putting Petri Nets to Work. IEEE Computer, pp: 85-94.
5. Yan, C., 1999. Petri Net models of grammars and its structure algorithm. J. Appl. Sci., 17: 58-63.
6. Esparza, J., 1997. Petri Nets, Commutative Context-Free Grammars and basic Parallel Processes. Fundamental Informaticae, 31: 13-25.
7. Lee K.-H. and F. Joel, 1985. Hierarchical Reduction: Method for Analysis and Decomposition of Petri Nets. IEEE Trans. on Systems, Man and Cybernetics, SMC-15: 272-280.
8. Darondeau, P., 2001. On the Petri net realization of context-free graphs. Theor. Computer Sci., 258: 573-598.
9. Erqing, X., 2004. A Pr/T-Net model for context-free language parsing, Fifth World Congress on Intelligent Control and Automation (IEEE Cat. No.04EX788), pt. 3, 1919-22, Vol.3.
10. de Lara, J., 2004. Defining visual notations and their manipulation through meta-modeling and graph transformation. J. Visual Languages and Computing, 15: 309-30.
11. Linz, P., 1990. An Introduction To Formal Languages And Automata, D.C. Heath and Company.