مجلة جامعة طرطوس للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (4) العدد(6) 2020
Tartous University Journal for Research and Scientific Studies - engineering Sciences Series Vol. (4) No. (6) 2020

التحويل الكلي للنحو الشكلي المبرمج المقيد الى النحو المبرمج الحر

د. يعرب شحادة ديوب*

(تاريخ الإيداع 8/10 /2020. قُبِل للنشر في 6 /9 /2020)

□ ملخّص □

ان النحو الشكلي المبرمج المقيد غني وذو امكانيات واسعة تسمح باستخدامه لتنفيذ التحليل النحوي لمختلف انواع سلاسل الدخل المحرفية. إلاّ ان استخدامه يتطلب مزيدا" من الوقت ،وذلك لاحتواء قواعد الاشتقاق على قائمة بأرقام قواعد الاشتقاق الممكن استخدامها لدى التنفيذ الصحيح والخاطئ لقواعد الاشتقاق المتاحة، وكذلك ارسال التحكم الى رقم قاعدة الاشتقاق المحددة اصلا" وبشكل عشوائي .

يتطرق هذا البحث للتخلص من المشاكل المطروحة أعلاه من خلال تحويل النحو الشكلي المقيد الى نحو من الشكل الحر ، وبالتالي توسيع مجالات استخدام هذا النوع من النحو ، وتنفيذ التحليل النحوي لعبارات الدخل بشكل افضل وسهولة استخدام اكبر.

الكلمات المفتاحية: النحو –حقل التنفيذ الصح –حقل التنفيذ الخطأ– قاعدة اشتقاق –تحليل.

*استاذ مساعد في قسم هندسة تقانة المعلومات – كلية هندسة تكنولوجيا المعلومات والاتصالات .جامعة طرطوس – طرطوس– سورية

مجلة جامعة طرطوس للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (4) العدد(6) 2020
Tartous University Journal for Research and Scientific Studies - engineering Sciences Series Vol. (4) No. (6) 2020

GLOBAL CONVERTING NON-FREE-CONTEXT PROGRAMED FORMAL GRAMMAR INTO FREE-CONTEXT TYPE

D.Yaroub Dayoub *

(Received 10/8/2020. Accepted 6/ 9/2020)

□ ABSTRACT □

The non-free context programmed formal grammar has rich and powerful possibility for describing and executing the syntaxis analysis of different string input types, but using this formal grammar requires additional time for applying recommended production rules included in additional two success and failed to-go fields and transfers the control to the randomly selected serial number of production rule which was recommended to use in applying process in result the a raised problems decreases the fields where it can be used.

This article introduce an approach to eliminate the above mentioned problems through converting the non-free context programmed formal grammar into free-context type and expanding its exploitation fields where it can be used, then executing the syntaxis processes more easy and better.

Keywords: Field, grammar, success, failed, production rule, syntaxis.

* assistant, Department of Technology Engineering, Faculty of Technology Engineering of Information and Communication, Tartous University, Tartous- Syria.

1. Introduction:

The programed formal grammar non-free context type (type-1 by Chomsky classification) provided with special set of production rule for some secondary elements used as subset production rules started with same let side secondary element and different right side[1,5] ,where each production rule body starts with the sequence serial number, core, failed to-go field and successes to-go field(four elements).

This structure of production rule permits executing syntaxis analysis for wide range of different string input series (formal language) which recovers various scientific fields.

After each applying of formal production rule, it necessary to check all these fields and this it will be, require large additional time.

Any production rule of the set related to the specified secondary related element can substitute by any right side of similar production rules started with the same secondary related element[2,4] at left side, so the length of selected path depending so far on the number of defined formal production rules related to the used number of secondary elements, so the total number of the production rules so far depending on the used number of production rules.

The selection of the production rules with the left or right recursion and applying them in syntaxis process requires very big extra spend syntaxis analysis time where this can leads to faulty applying of production rules and so long passed path to make the syntaxis analysis of input string series in best condition (the input series is correct) and the result of syntaxis analysis can be negative also (all production rules of related production rule was used and the input string was faulty organized) which can require applying a set of production rules, and this requires very big analysis time and this mater limit the using of programed formal grammar non-free context type.

For discarding above mentioned problems and decreasing the required for syntaxis analysis time it important to make the selection of the applying production rule of given formal grammar is very accurate and discard the possibility of faulty using of production rules, in result it's possible to minimize the number of returning the parent vertexes and so reducing the number of intermediate steps required for doing the syntaxis analysis of input string series.

2.Importance and aim of this work:

This article try to discard the above mentioned raised problems through executing the syntaxis analysis of input string series ,and minimized the required time by reducing the number of failed applying production rules through deleting the sequence serial number of production rule, failed to-go field and [3,5] success to-go field and substitute them by Boolean expressions ,where each production has special related expression with two possible values "true" (represent logical value "1") or "false" (represent logical value "0"),if the expression evaluated to "true" the related production rule will be used, otherwise wouldn't be used, so the production rule of recommended formal grammar has only 3 elements.

3. Search methods and materials:

The formal generated languages by non-free contest programmed formal grammar (type1) aren't powerful and not so enough to describe the natural languages and programming languages, for this reason it's not sufficient for executing syntaxis analysis of natural and programmed languages.

The set of production rules related to each secondary element is limited, in result, the set of possible to generate string series similar to input string series is also limited, so this kind of formal grammar is very weak instrument for executing the syntaxis analysis input string related to natural or programing languages.

From other side each production rule has limited number of possible to use formal production rules in failed to-go field and succeed to-go field in addition to the sequence serial number of the production rule, these fields must be checked at each intermediate operation through executing the syntaxis analysis of input string.

The programmed formal grammar defined as set of five elements as follows:

$G=\{V_T, V_N, S, J, P\}$ where:

V_T / V_N-finite set of terminal /non-terminal elements (characters),

$V = V_T \cup V_N$ –dictionary of G, $V = V_T \cap V_N = \Phi$, $S \in V_N$-start symbol,

J –label of production rule.

$A \longrightarrow \alpha$, α- can substitute by any production rules { $\alpha \longrightarrow 0A$, $\alpha \longrightarrow 0$ },

$A \in V_N$, $\alpha \in V^*$(*-zero, one or more occurrence)

ε – empty string of charaters.

P-Finite set of grammar's productions rules with following format:

$\ell \quad \propto \longrightarrow Y \quad S(U) \quad F(W)$,

where:

$\ell \in$J-Finite set of labels.

S(U)/ F(W)- succeeded / failed to-go field respectively.

U/W-list of labels where to-go if the substitution operation of the production rule was succeeded /failed respectively, where $U, W \subset J$.

The syntaxis analysis starts with applying the first production rule (ℓ=1) if there is any production rule has the same right side or can generate the same element in given input string, then it will be check if the applying of production rule was successful the next serial number recommended to use production rule is selected from the succeeded

to-go field " S(U) " otherwise the next expected to use serial number of production rule is selected from failed to-go field " F(W) ", in both conditions the next production rule for applying will selected randomly, in result executing more useless operation through syntaxis analysis of input string, which requests more time for recognition input string and using this grammar wouldn't be suitable.

From above mentioned remarks at each applying of production rule it's necessary to check first the right sight of production rules to find the searched input string element then succeeded / failed to-go field respectively taking in consideration the labels at which starts each production rule and this increase the syntaxis analysis time of input string and generally if applying of production rules was failed it's necessary to return to the parent vertex for selecting next production rules serial number.

It very important to notice the fact, where after applying many production rules it is possible to find at the end of partially or totally processing of syntaxis analysis the searched input element was not recognized (was faulty organized) and this increases the required time for executing syntaxis analysis.

It's very important to discard all as possible the useless operation and parts of the production used rule, by replacing succeeded / failed to-go field S(U)/ F(W) by bodies of production rules it selves in both conditions and discard also the labels $l_{\overline{i,i=1,n}}$ of

used production rules and through giving each production rule with an weight, which will be used later for giving (or not) permission in applying processes at different intermediate steps of syntaxis analysis.

4. **Discussion:**

At each applying of production rule it's necessary to check first the label with which the production rule start then core through checking the first element of right sight of the production rules if equal the searched input element if the answer was "yes" it will be use any production rule of recommended to use production rules in case succeeded lets suppose consisting of "m" production rule or failed to-go field taking consisting of n rules, so the total number of possible executing operation "N" at each step of syntaxis analysis is given as follow:

$N=m+n+1;$ (1)

Where: 1-the number of labels of production rule.

If the formal grammar contains "**K**" production rules, so maximum expected to use production rule is calculate by the next formula:

$$N_{total} = N*K = (m+n+1)*K \quad (2)$$

It's clear that is each applying of production rule through executing syntaxis analysis of input string requires " N_{total} " time unite additional by comparing with using other types of formal grammar this makes it complex to use with low using efficiency value and this leads to put some bounders of using programmed formal grammar with a high spend time for recognizing input string.

Here it is necessary to solve raised different problems with using programmed formal grammar type through eliminating the succeeded S(U) and failed F(W) to-go fields with the labels of production rules by the production rules themselves and in this condition discard the un necessary labels of production rules.

Each production rule of succeeded S(U) and failed F(W) to-go fields has special weight given as Boolean expression $w_{i,\overline{i=1,n}}$, where depending on it's the evaluated value it will be use the related production rule in condition "true" value otherwise the related production rule wouldn't be use and the processes will be continue to find any production rule starts with right side equal to input element otherwise the input string wouldn't be recognized and the syntaxis analysis gives a message that's the input string was faulty constructed. At each step of syntaxis analysis for string input, it's possible to recognize two conditions:

a)For the first element of given string input, there is only one production rule at maximum with $w_{i,\overline{i=1,n}}$ = "true" value which can be used at current step of syntaxis analysis if and only if the current input element equal to terminal element at left side(using left or right recursion) of selected to use production rule and discard all remain production rules from visiting list ,then the next input element will be taken for continuing analysis processes string then it will be check the weight of next (second) production rule if it was in "true" condition it will be used in syntaxis processes otherwise it will look for any weight in "true" condition ,so it will be selected to all production rules with true condition up to get the end of the expected to use production rules set.

b)There aren't any weight $w_{i,i=\overline{1,n}}$ = "true" ,so it will be fix an error message that is the input string was constructed faulty.

In both conditions the number of possible to use production rules was minimized by k*(m+n) times.

For decrease the required for syntaxis analysis time it's recommended to replace the succeeded / failed to-go field S(U)/ F(W) with set of production rules for each one of production rules .let's suppose the set of production rules for succeeded to-go field S(U) as the following(by the same way can be construct the set of production rules for the failed to-go field F(W)):

$$\vartheta \xrightarrow{w_1} \beta_1\, \vartheta \qquad w_1 : t>c_1 \mid d>c_2$$
$$\vartheta \xrightarrow{w_2} \beta_2\, \vartheta \qquad w_2 : t=c_3 \mid d<c_4 \qquad (4)$$
$$\vartheta \xrightarrow{w_3} \beta_3\, \vartheta \qquad w_3 : t>c_5 \;\&\; d!=c_6$$
……………………………………………………..
$$\vartheta \xrightarrow{w_n} \beta_n\, \vartheta \qquad w_n : t<c_n \;\&\; d>c_{n+1}$$

Where:

$w_1\ w_2, \ldots w_n$ -Boolean expression (the weight of production rules).

$$w_1 != w_2 != \ldots w_n \qquad (5)$$

$V_T U\ V_N = V$

$\vartheta, \beta_1, \beta_2, \beta_3, \ldots, \beta_n \in V^* V_N V^*$ (*-zero, one or more occurrence) $\vartheta \in V^*$

t-expired time for related applying of production rule, d-the length of passed path.

$c_1, c_2, \ldots, c_{n+1}$-some integer constants used as limiters.

In the free -context programed grammar it was inserted the production rules set itself instead of their sequence serial numbers (labels) and if the weight of first production rule where can recognized this two conditions:

1) $w_1 = "true"$ the related production rule will be used in current intermediate step of syntaxis analysis and all other rules will be discarded from the processes at this moment and it will be repeated the above mentioned operations.

2) If $w_1 = "false"$,it will be check if the weight of second production rule $w_2 = "true"$ so it will be used in syntaxis analysis processes, this processes will be repeated to get the end of given input string.

If all elements of input string was simulated by production rules because their weights was in "true" condition , the string input was correctly constructed and if any input element wasn't recognized the input string will be refused and error message will be given.

The recommended free-context programmed formal grammar is defined as four tuples as follow:

$G_p = \{V_T, V_N, P, S\}$ Where:

V_T/V_N-Finite set of terminal/non terminal elements respectively.
S-start symbol (element)
P-finite set of production rules with next format:

$$\vartheta \xrightarrow{w_i} \beta_i \vartheta \qquad w_i : t>c_1 \mid d>c_2 \qquad (6)$$

Where:

w_i -the weight of production rule with sequence serial number "i".
t/d-expired time/long of distance respectively.
C1,C2-positive numeric constants used as limiters.
$\vartheta \in V^*$, $\beta_i \in V^* V_N V^*$

As seen from above the production rule's format consisting two parts the core and it's weight only and the succeeded and failed to-go fields and the production's label also are eliminated.

Each production rule of formal grammar $S_{i,\overline{i=1,n}}$ has special weight $w_{i,\overline{i=1,n}}$, which permits to change the production rule from non-free context (type1)to free-context type(type2) and minimizing the number of used production rules by N_{mini} times as follow:

$N_{mini} = N*K=(m+n)*K$ (7)

In the result, the required time to make syntaxis analysis is minimized by N_{mini} time units unite of time and in the result the efficiency of using like this formal grammar was increased by the same value $N_{mini,\ this}$ type of formal grammar opens widely different fields in control systems.

At each intermediate step of syntaxis analysis it possible to use only one production rule if the input element is equal to any left side element of production rules and the related weight $w_{i,\overline{i=1,n}}$ was in "true" condition ,so no place to be faulty use the production rules and no need to return to parent vertex .

EXAMPLE:

Suppose it is required to make the syntaxes analysis of the next input expression

$X=0^n1^n0^n$ (8)

By using, the recommended non-free context programmed formal grammar:
For generation the given input string let's suppose the following conditions:

n=3-the occurrence number of each input element,
$N_{derv.lev}$=6-he number of derivation levels,
N_{weight}=19 -the number of related weights $w_{i,\overline{i=1,n}}$,

For making syntaxis analysis, the number of possible to use formal production rules is calculated with the formula. $N_{total}=2^n*n_{sub}+1$ (9)

where :

n-the occurrence of input character,(let's suppose n=3)
n_{sub}-the number of sub-tree (let's suppose n_{sub} =3).

By substitution in formula (9) we get the $N_{total}=2^n*n_{sub}+1=2^3*3+1=25$
Therefore, the relation 8 takes the next form:

$X=0^3 1^3 0^3$ (10)

Let's construct the recommended free-context programmed formal grammar "G_f" for executing the syntaxis analysis of input string "X" , which defined as four tuples as follow:

$G_f=\{V_T, V_N, P, S\}$

Where:

V_T/V_N-Finite set of terminal/non terminal elements respectively.

$V_T=\{0,1\}$, $V_N=\{S,A,B,C,D,F,Y\}$,

S-start symbol (element),

P-finite set of production rules with next format:

1: <S> $\xrightarrow{\omega_1}$ <D> <F> <Y> , ω_1 : d=3 & f_1=0 |f_1=1

2: <S> $\xrightarrow{\omega_2}$ <A> <B> <C>, ω_2 : d>3 & f_2=0

3: <A> $\xrightarrow{\omega_3}$ 0 <A> , ω_3 : d>3 & f_1=0

4: <A> $\xrightarrow{\omega_4}$ 1 <A>, ω_4 : isend&f_2=1. (11)

5: <A> $\xrightarrow{\omega_5}$ 0, ω_5 : isend&f_1=0

6:<A> $\xrightarrow{\omega_6}$ 1, ω_6 : d!=2 & f_2=1

7:<A> $\xrightarrow{\omega_7}$ ε (ε -empty string) ω_7 : isend & f_1= ε

8:<A> $\xrightarrow{\omega_8}$ <B> ω_8 :L>=1

9:<A> $\xrightarrow{\omega_9}$ <C> ω_9 :L>=1

10: <B> $\xrightarrow{\omega_{10}}$ 0 <B> , ω_{10} : d>3 & f_1=0

11: <B> $\xrightarrow{\omega_{11}}$ 1 <B>, ω_{11} : isend&f_2=1

12: <B> $\xrightarrow{\omega_{12}}$ 0, ω_{12} : isend&f_1=0

13:<B> $\xrightarrow{\omega_{13}}$ 1, ω_{13} : d!=2 & f_2=1

14:<B> $\xrightarrow{\omega_{14}}$ ε (ε -empty string) ω_{14} : isend & f_1= ε

15:<B> $\xrightarrow{\omega_{15}}$ <C> ω_{15} : d<3 & f_2=0

16: :<B> $\xrightarrow{\omega_{16}}$ <A> ω_{16} : d>3 & f_1=0

17: <C> $\xrightarrow{\omega_{17}}$ 0 <C> , ω_{17} : d>3 & f_2=0

18: <C> $\xrightarrow{\omega_{18}}$ 1 <C>, ω_{18} : isend&f_1=1

19: <C> $\xrightarrow{\omega_{19}}$ 0, ω_{19} : isend&f_2=0

20:<C> $\xrightarrow{\omega_{20}}$ 1, ω_{20} : d!=2 & f_1=1

21:<C> $\xrightarrow{\omega_{21}}$ ε (ε -empty string) ω_{21} : isend & f_2= ε

22:<C> $\xrightarrow{\omega_{22}}$ <A> ω_{22} : d<=10 & f_1=1

23:<C> $\xrightarrow{\omega_{23}}$ <B> ω_{23}: d>=5 & f_2=1

……

Where:

d-length of expected to pass path. , L,f_1,f_2-optional constants.

$f_1 \in \{1, \varepsilon, 0\}$-previously input element

$W_{i,\overline{i=1,n}}$ ={ $\omega_1, \omega_2, .., \omega_{23}$ -special weights of production rules with sequence serial number "i":

isend-function answers on the question if there was casting finishing reanalysis processes.

Let's construct the binary syntaxis tree for input string $X=0^3 1^3 0^3$ through using the recommended free-context programmed formal grammar (11).

The syntaxis binary tree for string input consisting of 6 levels and three branches so the expected to use $N_{total}=2^n * n_{sub}+1=2^3*3+1=25$ number of production rules.

Suppose the failed to-go field F(W) of labels for production rules is empty and succeeded to-go field S(U)=3,Here it's possible to meet tow conditions:

1) Input string was **correctly constructed** t.e $X \in L(G)$ (X-an element of generated by formal programed grammar set of string series) (case a).

a) Using traditional **non-free context programed** formal grammar.

The number of expected to use production rules $N_{expec.\mathbf{non}}$ in syntaxis analysis is the multiplying of number of vertexes possible to use $N_{ver.\mathbf{non}}$ at each vertex by the expected number of production rules with the same at left side $N_{expec.ver}$ which calculated by the next formula:

$N_{expec.non}=N_{ver.\mathbf{non}} * N_{\mathbf{expec}.ver}=13*5+1=66.$

-Tthe **minimum** number $N_{min.suc.prog}$ and the maximum number of used production rules in case the input string was correctly(succeeded to-go field will be use) constructed ($X \in L(G)$ as follow:

$N_{min.suc.non}= N_{expec.\mathbf{non}} + N_{\mathbf{intermid}.ver} =66+2=68.$

where:

$N_{intermid.ver}$ -the number of intermediate used vertexes (represents in our case the vertexes "S" and "Z" ,so $N_{intermid.ver}=2$).

-The **maximum** number $N_{max.\mathbf{suc}.non}$ number of used production rules consisting of The number of expected to use production rules $N_{\mathbf{expec}.non}$ and the numer of production rules with the same left side $N_{recom.ver}$ in syntaxis analysis t.e:

$N_{max.\mathbf{suc}.non}= N_{\mathbf{expec}.non} + N_{\mathbf{recom}.ver} =68+11=79.$

In result the average number of possible to use production rules $N_{aver.prog}$ is given as follow:

$N_{aver.non=} (N_{min.\mathbf{suc}.non} + N_{max.\mathbf{suc}.non})/2=(68+79)/2=82.5$

b) Using **recommended free-context programed** formal grammar.

The maximum possible to use number of production rules $N_{max.\mathbf{free}}$ in syntaxis analysis is the result of multiplication the total number of possible to visit vertexes by the number of expected to use production rules is calculated by the next formula:

$N_{max.\mathbf{free}}= N_{ver.\mathbf{free}} * N_{expec.\mathbf{free}}=14*1=14.$

While the minimum possible to use number of production rules $N_{min.\mathbf{free}}$ in syntaxis analysis is calculated by the next formula:

$N_{min.\mathbf{free}} = N_{ver.\mathbf{free}} = 14$

The average number of possible to production rules $N_{aver.suc.free}$ is given as follow:

$N_{aver.free} = (N_{max.\mathbf{free}} + N_{min.\mathbf{free}})/2 = (14+14)/2 = 14.$

2) Input string was faulty constructed t.e $\mathbf{X \notin L(G)}$ (X-isn't an element of generated by **non-free context** formal programed grammar set of string series

e) Using **non-free context** programed formal grammar.

The minimal number of used production rules is consisting of minimal number of used production rules in succeed case $N_{min.\mathbf{faild}.non\text{-}free}$ and the number of expected to use production rules $N_{expec.\ non\text{-}free}$ calculated as follow:

$N_{min.\mathbf{faild}.\ non\text{-}free} = N_{min.\mathbf{suc}.non\text{-}free} + N_{\mathbf{expec}.\ non\text{-}free} = 68+18 = 86.$

-At case of using failed to-go field the maximum number of used production rules $N_{max.\mathbf{suc}\ .non\text{-}free}$ includes the maximum number of used production rules in succeed case $N_{max.\mathbf{suc}.non\text{-}free}$ and the number of not used production rules $N_{not\text{-}used.\ non\text{-}free}$ is calculated as follow:

$N_{max.\mathbf{faild}\ .non\text{-}free} = N_{max.\mathbf{suc}.non\text{-}free.} + N_{\mathbf{not\text{-}used}.\ non\text{-}free.} = 79+18 = 97.$

Let's calculate the average number of used production rules in failed input (non-free context) as follow:

$N_{ave.\ \mathbf{faild}\cdot non\text{-}free} = (N_{\mathbf{min}.\mathbf{faild}.\ non\text{-}free} + N_{\mathbf{max}.\mathbf{faild}\ .non\text{-}free})/2 = (86+97)/2 = 91.5$

f) Using recommended free context programed formal grammar

In faulty applying of production rule the minimum number of used production rules is only one production rules with the Boolean expression evaluated to "false" value $N_{min.faild.free} = 1$ and no more than production rules will be use and the syntaxis analysis will be terminates immediately and fix an input error pointing that's the input string was faulty constructed.

While the syntaxis analysis can star with one of two possible to use formal grammar so the maximum number recommend to use of production rule is $N_{max.\mathbf{faild}.free} = 2$.

It's very important to notice that the average number of using production rules in failed constructed input string is calculate as the following:

$N_{ave.\ \mathbf{faild}.free} = (1+2)/2 = 1.5$

This factor gives this free context programmed formal grammar more new possibilities and open for it new using technical and scientific fields.

*****Regarding to the above-mentioned results we recognize **two conditions:**

a) The input string was constructed successfully t.e, X∈L(G) lets the find the value of decreasing factor $N_{min\cdot\mathbf{decr}}$ as the difference between the value of two conditions through using non-free context then free- context programed formal grammar t.e as follow:

$N_{min\cdot\mathbf{decr}}$ =68-14=54

By the same way the the decreasing factor is:

$N_{max\cdot\mathbf{decr}}$ =79-14=65

b) The input string was constructed successfully t.e, X ∉ L(G) lets the find the value of decreasing factor $N_{min\cdot decr}$ as the difference between the value of two conditions Through using non-free context then free- context programed formal grammar t.e as follow:

$N_{min\cdot\mathbf{decr}}$ =86-1=85

By the same way the the decreasing factor as follow: $N_{max\cdot\mathbf{decr}}$ =115-2=113.We summarized all these results in table.1

Table1. The used number of production rules for non-free & free-context types formal grammar.

	X∈L(G)		X ∉ L(G)		N_{Aver}	
	N_{min}	N_{max}	N_{min}	N_{max}	X∈L(G)	X ∉ L(G)
Non-free context programed	68	79	86	97	82.5	91.5
Free-context programed	14	14	1	2	14	1.5
Enhancement value	54	65	85	113	59.5	99

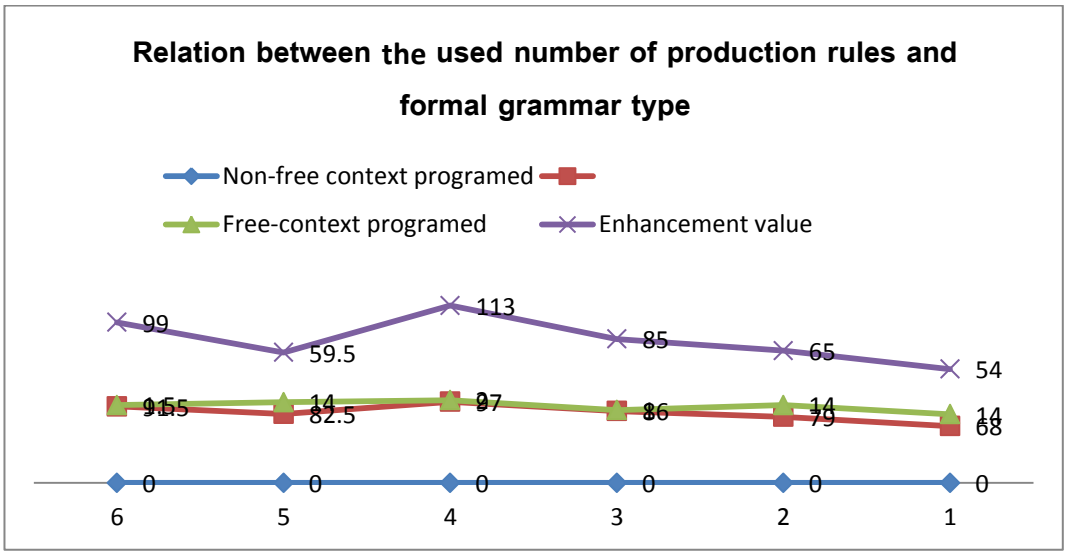

Fig2. Relation between non-free & free-context formal grammar.

Conclusions:

The minimized structure of production rule of recommended non-free context programmed formal grammar permits the following operations:

- Eliminate failed /succeeded to-go field and the production rule's serial sequence number.

- Minimized structure of production leads to minimize the required time for executing

the syntax analysis of input string.

-Each production rule has special Boolean expression, which can be evaluated to "true" or "false" value, it will no necessary to return to the parent vertex.

-Increasing the power of programed formal grammar and simplifying the structure of the grammar and became easy simple to use.

-The above mentioned factors gives the grammar the possibility to use in different new fields.

REFERENCES:

1.Luis M Augusto. Languages, Machines, and Classical Computation Paperback – February 4, 2019.

2.Luis M. Augusto, Languages, machines, and classical computation, London: College Publications, 2019. ISBN 978-1-84890-300-5. Web page.

3.C Mag Staff ."Encyclopedia Definition of Compiler". PCMag.com. Retrieved 2017

4. Sun, Chengnian ; Le, Vu; Zhang, Qirun; Su, Zhendong (2016). "Toward Understanding Compiler Bugs in GCC and LLVM". ACM.

5. Silberztein, Max (2013). "NooJ Computational Devices". Formalizing Natural Languages with NooJ. pp. 1–13. ISBN 978-1-4438-4733-9.

6. 35 المجلد2013، مجلة جامعة البعث "نمذجة المحادثة التعليمية المؤتمتة مع الكائن" .ديعرب ديوب

7.Dayoub.y. Predictive adaptive dynamic object's traversals control, Tartous univer.Voluem 3,N:6 (2019).