Analysis of Valid Closure Property of Formal Language

Chen Wenyu
School of Computer Science and Engineering, University of Electronic Science & Technology of China, Chengdu, Sichuan, China
Email: cwy@uestc.edu.cn

Wang Xiaobin
School of Computer Science and Engineering, University of Electronic Science & Technology of China, Chengdu, Sichuan, China
Email: xbwang@uestc.edu.cn

Cheng Xiaou
School of Computer Science and Engineering, University of Electronic Science & Technology of China, Chengdu, Sichuan, China
Email: carlyhawk@gmail.com

Sun Shixin
School of Computer Science and Engineering, University of Electronic Science & Technology of China, Chengdu, Sichuan, China
Email: sxsun@uestc.edu.cn

Abstract—This paper focuses on the basic operations of Chomsky’s languages. The validity and the effectiveness of some closure operations, such as union operator, product operator and Kleene Closure operator, are discussed in detail. The crosstalk problems in Context-Sensitive Languages (CSL) and Phrase Structure Languages (PSL) are analyzed, and a valuable method to solve this problem is presented by using the alphabet of the operating languages. In addition, according to the valid closure property of regular languages (RL), a simple method to create a regular expression (RE) is proposed. The closure property of the permutation operator in Context-Free Languages (CFL) is proved and tested. In conclusion, by using our proposed methods, the exact type of a given language can be proved theoretically. By the way, the grammar to produce complex language can be created easy. Finally, the constructing ε-NFA with the closure property is proved.

Index Terms—language operation, valid closure property, crosstalk, context-free permutation

I. INTRODUCTION

Given alphabet Σ, Ψ is a type of language of Σ, language L_1, L_2 ∈ Ψ, let α be a binary operation of the language:
(L_1, L_2) → α(L_1, L_2)
β is the unary operation of the language:
L_1 → β(L_1)
If for any language of Ψ, L_1 and L_2, α(L_1, L_2) is also a language of Ψ, then we say Ψ is closed on the operation α[1].
For grammars generating languages, given a specified operation, the grammar of the same type language can be created, then the language is effectively closed on that operation.
The closure issue of language operation is important in language research and has significant value in both theory and practice[1,2,3].
Linz Peter proposed the crosstalk problem of context-dependent language and the corresponding solutions[1] without discussing the crosstalk problem of 4 types languages of Chomsky theory by Kleene closure operation. Prof. Jiang Zongli and Prof. Chen Youqi proved that for different alphabets, regular language and context-free language are effectively closed by basic language operations[2,3]. From the standpoint of automata, especially Turing machine, Michael Sipser discussed the valid closure property issue of language operations[4,5].
From the view of formal language, the paper proves that 4 types languages of Chomsky theory are effectively closed by join, product and Kleene closure operations. The paper proposes solution to crosstalk problem of context-dependent language and phrase structure language by product and Kleene closure operations and discusses the valid closure property of context-free language by context-free in-placement.

II. LANGUAGE CLASSIFICATION
For any grammar $G=(\sum,V,S,P)$, $G$ is type-0 grammar, or PSG(Phrase Structure Grammar). $G$ generates type-0 language, or Phrase Structure Language correspondingly.

Grammar $G$, if for any $\alpha \rightarrow \beta \in P$, we have $|\alpha| \leq |\beta|$, then $G$ is type-1 grammar, or Context-Sensitive Grammar(CSG). $G$ generates type-1 language, or Context-Sensitive Language correspondingly.

If for any $\alpha \rightarrow \beta \in P$, we have $|\alpha| \leq |\beta|$ and $\alpha \in V$, then $G$ is type-2 grammar, or Context-Free Grammar(CFG). The language generated by $G$ is type-2 language or Context-Free Language(CFL).

If for any $\alpha \rightarrow \beta \in P, \alpha \rightarrow \beta$, we have forms like $A \rightarrow w$ or $A \rightarrow wB$, in which, $A,B \in V$, $w \in \sum$, then $G$ is type-3 grammar, or Regular Grammar(RG). Correspondingly, the language generated by $G$ is type-3 language or Regular language(RL).

The basic principle to classify grammar disallows $\varepsilon$-formula in type-1, type-2 and type-3 grammars; if $S$ is not on the right side of any formula of the grammar, and if $G$ is type-1, type-2 or type-3 grammar, then $G^*=(\sum,V,S,P \cup \{S \rightarrow \varepsilon\})$ and $G^*=(\sum,V,S,P \cup \{S \rightarrow \varepsilon\})$ are still type-1, type-2 or type-3 grammars, and the languages correspondingly generated are also type-1, type-2 or type-3 languages.

III. BASIC LANGUAGE OPERATIONS

Languages $L_1$ and $L_2$ are based on alphabet $\sum_1$ and $\sum_2$ respectively, the union operation of $L_1$ and $L_2$ is:

$L_1 \cup L_2$ = \{ $w \in L_1$ or $w \in L_2$ \}

the product operation of $L_1$ and $L_2$ is:

$L_1L_2$ = \{ $w$ | $w=w_1w_2, w_1 \in L_1, w_2 \in L_2$ \}

the Kleene closure operation (or Star operation) of $L_1$ is

$L_1^*$ = \{ $w$ | $w=w_1w_2…w_m, w_1 \in L_1, m \geq 0$ \}

∪ $L_1^n$ for $n \geq 0$

IV. THE VALID CLOSURE PROPERTY OF LANGUAGE ON OPERATIONS

The valid closure property can be described as following: given same type grammars $G_1$ and $G_2$

$L_1 = L(G_1)$

$L_2 = L(G_2)$

Same type grammar $G$ must be created to satisfy

$L(G) = u(L_1, L_2)$

or

$L(G) = \beta(L_1)$

V. THE VALID CLOSURE PROPERTY OF BASIC OPERATIONS IN 4 TYPES OF LANGUAGES

Let language $L_1$ and $L_2$ attribute to the languages of alphabet $\sum_1$ and $\sum_2$ respectively, grammar $G_1$ generates language $L_1$

$G_1 = (\sum_1, V_1, S_1, P_1)$

generate $G_2$ generates language $L_2$

$G_2 = (\sum_2, V_2, S_2, P_2)$

Then

$S_1 \Rightarrow \alpha \Rightarrow *w_1 \in L_1$

$S_2 \Rightarrow \beta \Rightarrow *w_2 \in L_2$

Suppose

$\sum_1 \cap \sum_2 = \Phi; V_1 \cap V_2 = \Phi; S \notin V_1, S \notin V_2$

Set

$\sum = \sum_1 \cup \sum_2$

$V = V_1 \cup V_2 \cup \{ S \}$

A. The Valid Closure Property on Union Operation

Create grammar

$G_1 = (\sum, V, S, P_3)$

in which

$P_3 = \{ S \rightarrow S_1 ; S \rightarrow S_2 \} \cup P_1 \cup P_2$

For $i=0, 1, 2$, if $G_1$ and $G_2$ are type-$i$ grammar, then $G_3$ is the same type grammar. $G_3$ could use

$S \Rightarrow S_1 \Rightarrow \alpha \Rightarrow *w_1 \in L_1$

to obtain $L_1$; or use

$S \Rightarrow S_2 \Rightarrow \beta \Rightarrow *w_2 \in L_2$

to obtain $L_2$, that is

$L(G_3) = L_1 \cup L_2$

So, languages of type-0,1,2 are effectively closed on union operation.

For example, type-2 grammar $G_1$ is

$S_1 \rightarrow nSa$

$S_1 \rightarrow bS_1b$

$S_1 \rightarrow cS_1c$

$S_1 \rightarrow a|b|c$

$S_1 \rightarrow aa|bb|cc$

and type-2 grammar $G_2$ is

$S_2 \rightarrow AC$

$A \rightarrow 0A1$

$A \rightarrow 01$

$C \rightarrow 2|2C$

so $L_1$ is

$\{ x | x = x^T, x \in \{a,b,c\}^+\}$

and type-2 grammar $G_2$ is

$S_2 \rightarrow AC$

$A \rightarrow 0A1$

$A \rightarrow 01$

$C \rightarrow 2|2C$

so $L_2$ is

$\{0^m1^n0^m|n.m > 0\}$

Set type-2 grammar $G_3$ is

$S \rightarrow S_1$

$S \rightarrow S_2$

$S_1 \rightarrow nSa$

$S_1 \rightarrow bS_1b$

$S_1 \rightarrow cS_1c$

$S_1 \rightarrow a|b|c$

$S_1 \rightarrow aa|bb|cc$

$S_2 \rightarrow AC$

$A \rightarrow 0A1$

$A \rightarrow 01$

$C \rightarrow 2|2C$

so $L_3$ is

$\{ x | x = x^T, x \in \{a,b,c\}^+ \} \cup \{0^m1^n0^m|n.m > 0\}$

that is

$L_3 = L_1 \cup L_2$

If $G_1$ and $G_2$ are type-3 grammar while $G_3$ is not type-3 grammar, then create type-3 grammar

$G_3 = (\sum, V, S, P_4)$

in which

$P_4 = \{ S \rightarrow \alpha S_1 \rightarrow \alpha \in P_1 \}$
For example, type-2 grammar \( G \) is
\[
S \rightarrow \alpha \rightarrow \star \{x \in \{a, b, c, d\}^* \}
\]
and type-2 grammar \( G_2 \) is
\[
S_2 \rightarrow \alpha C
A \rightarrow 0A|0
C \rightarrow 1C2|12
\]
so \( L_2 \) is
\[
\{0^n1^m2^m|n,m>0\}
\]
Set type-2 grammar \( G_3 \) is
\[
S \rightarrow S_1S_2
S_1 \rightarrow aS_1a
S_1 \rightarrow bS_1b
S_1 \rightarrow cS_1c
S_1 \rightarrow dS_1d
S_1 \rightarrow aa|bb|cc|dd
S_2 \rightarrow AC
A \rightarrow 0A|0
C \rightarrow 1C2|12
\]
so \( L_3 \) is
\[
\{a^n|n>0\}
\]
that is
\[
L_3 = L_1 \cup L_2
\]
If \( G_1 \) and \( G_2 \) are type-3 grammar while \( G_3 \) is not type-3 grammar, create type-3 grammar,
\[
G_5 = (\Sigma, V, \Gamma, S, P_6)
\]
in which
\[
P_6 = \{ A \rightarrow wS_2 | A \rightarrow w \in P_1 \}
\]
For every formula like
\[
A \rightarrow w
\]
rewritten as
\[
A \rightarrow wS_2
\]
Grammar \( G_6 \) uses
\[
S_1 \Rightarrow \star r_1r_2...r_nA
A \Rightarrow r_1r_2...r_nwS_2
A \Rightarrow \star wS_2 \in L_1 \cup L_2
\]
in which, \( r_1r_2...r_nw \in L_1 \), that is,
\[
L(G_6) = L_1 \cup L_2
\]
So, language of type-3 is effectively closed on product operation.

For example, type-3 grammar \( G_1 \) is
\[
S_1 \rightarrow aS_1a
A \rightarrow bA|cA|bB|cB
B \rightarrow dB|d
\]
so \( L_1 \) is
\[
\{a^n|n>0\}
\]
Set type-3 grammar \( G_6 \) is
\[
S_1 \rightarrow aS_1a
A \rightarrow bA|cA|bB|cB
B \rightarrow dB|d
S_2 \rightarrow 0C
\]
C→0|1|0C|1C

so $L_6$ is

$a^*(b+c)^*d^*(0+1)^*$

that is

$L_6=aL_1L_2$

C. The crosstalk of Product operation

$G_1$ and $G_2$ are type-0 or type-1 grammar, if

$\Sigma_1 \cap \Sigma_2 \neq \Phi (\Sigma_1=\Sigma_2$ is possible)

the grammar $G_3$ is not always correct. For example:

Grammar $G_1$:

$$S_1\rightarrow a$$

Grammar $G_2$:

$$S_2\rightarrow aS_2$$
$$aS_2\rightarrow bc$$

then

$L_1=\{a\}, L_2=\{a, bc\}$

$L_1L_2=a\cdot bc$

However, if $G_4$ uses

$S\rightarrow S_2S_2 \rightarrow aS_2 \rightarrow a'\cdot S_2\rightarrow^+ a'\cdot bc$

there can also be

$S\rightarrow S_2S_2 \rightarrow aS_2 \rightarrow bc$

The language generated by grammar $G_3$ is

$a\cdot bc \neq L_1L_2= a\cdot bc$

The crosstalk between sentence patterns generated by $S_1$ and $S_2$ is the reason why $G_3$ is not what we want sometimes.

Namely, the sentence pattern generated by $S_1$ might take for the sentence generated by $S_2$ as the following text, while the sentence generated by $S_2$ might take for the sentence generated by $S_1$ as the preceding text; and the crosstalk could only be caused by the terminal symbol.

To solve the problem above, copy $\Sigma$ as $\Sigma'$ and $\Sigma''$

$$\Sigma' = \{x' \mid x \in \Sigma\}$$
$$\Sigma'' = \{x'' \mid x \in \Sigma\}$$

Replace $x$ in $P_1$ by $x'$ and then obtain $P'$, replace $x$ in $P_2$ by $x''$ and then obtain $P''$, the process is to distinguish the terminator symbols between $G_1$ and $G_2$ in deduction. Finally, $x'$ and $x''$ need to be restored to the original terminator symbols.

Create grammar

$$G_e=(\Sigma, V \cup \Sigma', \Sigma'', S, P_e)$$

in which

1. $P_e= \{ S \rightarrow S_2S_2 \cup P' \cup P'' \}$
2. $\cup \{ x' \rightarrow x \mid x \in \Sigma \}$
3. $\cup \{ x'' \rightarrow x \mid x \in \Sigma \}$

$G_7$ uses

$S\rightarrow S_2S_2 \rightarrow w_1w_2 \rightarrow^+ w_1w_2 \rightarrow L_1L_2 = \in L_1L_2$

thus, the crosstalk problem is solved.

In the example above,

Grammar $G_1$:

$$S_1\rightarrow a$$

Grammar $G_2$:

$$S_2\rightarrow aS_2$$

P_7 is

$aS_2\rightarrow bc$

$S \rightarrow S_2S_2$
$S_1 \rightarrow a'\cdot S_2$
$S_2 \rightarrow a''\cdot S_2$
$a''\cdot S_2 \rightarrow b''\cdot c''$
$a'' \rightarrow a$
$b'' \rightarrow b$
$c'' \rightarrow c$

$G_7$ uses

$S \rightarrow S_2S_2$

$\rightarrow a'\cdot S_2$ \hspace{1cm} //Can’t use $a''\cdot S_2 \rightarrow b''\cdot c''$

$\rightarrow a''\cdot S_2$

$\rightarrow a''\cdot b''\cdot c''$

$\rightarrow a''\cdot b''\cdot c''$

to create the product language $a'\cdot bc$ of $L_1$ and $L_2$.

D. The Valid Closure Property on Kleene Closure operation

The generation of sentence $\epsilon$ and any number of products must be considered in Kleene Closure operation.

Adding a formula

$$S \rightarrow \epsilon \mid S_1S_1$$

to generate empty sentence and any number of products of $L_1$.

Since $S$ is on the right side of the formula, which is not satisfied the principle of closure, and can generate other extra strings so we add a new non-terminal symbol to solve the problem.

Rewrite the newly added formula,

$$S \rightarrow \epsilon \mid S'$$
$$S' \rightarrow S_1S_1$$

then only $\epsilon$ and $S_1^n(n \geq 1)$ can be deduced from $S$.

Create grammar

$$G_e=(\Sigma, V \cup \{S, S'\}, S, P_e)$$

in which

$P_e= \{ S \rightarrow \epsilon \mid S' \rightarrow \{S_1S_1\} \} \cup P_1$

If $G_1$ is type-2 grammar, then $G_5$ is also type-2 grammar and

$L(G_{G_5})=L_1^*$

So, language of type-2 is closed on Kleene Closure. If $G_1$ is type-0 or type-1 grammar, grammar $G_3$ may also has crosstalk problem. That because

$S \rightarrow S_1 \cdots S_i \cdots S_1$

each $S_i$ could only generate sentence of $L_1$ from the formula of $P_1$, and the sentence patterns generated by any two consecutive $S_i$ might be following and preceding text with each other, then crosstalk is appear.

To avoid crosstalk, copy $\Sigma$ as $\Sigma'$ and $\Sigma''$, create $P'$ and $P''$; rewrite $S_1$ as $S'$, create grammar

$$G'=(\Sigma, V \cup \Sigma', \cup \{S'\} \rightarrow S_1, S', P'')$$

Rewrite $S_1$ as $S''$, create grammar

$$G''=(\Sigma, V \cup \Sigma'' \cup \{S''\} \rightarrow S_1, S'', P'')$$

Create grammar

$$G_e=(\Sigma, V \cup \Sigma' \cup \Sigma'' \cup \{S'\} \rightarrow S_1, S_1S_2), S, P_e)$$

in which

$P_e= \{ S \rightarrow \epsilon \mid S_1S_2\}$
\[ S \to \varepsilon \mid S_1 S_2 \]
\[ S_1 \to S \mid S_2 \]
\[ \cup \{ S \to S' \mid S' \subseteq S \} \]
\[ \cup \{ \varepsilon \to \\} \]
\[ \cup \{ S \to S'' \mid S'' \subseteq S \} \]
\[ \cup \{ x \to x \mid x \in \Sigma \} \]

To avoid crosstalk itself, \( S' \) and \( S'' \) must be alternated to satisfy:

\[ S \Rightarrow S_1 \Rightarrow S' S'' S' S'' \Rightarrow \cdots \Rightarrow S' S'' \]

or

\[ S \Rightarrow S_2 \Rightarrow S'' S' S'' S' \Rightarrow \cdots \Rightarrow S'' S' \]

and

\[ S \Rightarrow S_2 \Rightarrow S'' S' S'' S' \Rightarrow \cdots \Rightarrow S'' S' \]

or

\[ S \Rightarrow S_2 \Rightarrow S'' S' S'' S' \Rightarrow \cdots \Rightarrow S'' S' \]

then the consecutive \( S_i \) are replaced by alternated \( S' \) and \( S'' \), each \( S' \) and \( S'' \) could only deduce from the formula of \( P' \) or \( P'' \) respectively, and crosstalk is avoided.

\( S' \) and \( S'' \) each generates language of alphabet \( \Sigma' \) and \( \Sigma'' \) (The sentence structures are equal to the sentence structure of \( L_1 \)), then after restoration, \( L_1^* \) is obtained, that is

\[ L(G_0) = L_1^* \]

So, language of type-0 and type-1 are closed on Kleene Closure operation.

For Example, type-1 grammar \( G_1 \) is

\[ S \to a S B C \]
\[ S_1 \to a B C \]
\[ CB \to B C \]
\[ a B \to a b \]
\[ b B \to a B \]
\[ b C \to B C \]
\[ c C \to C C \]

so \( L_1 \) is

\[ \{ a^b b^c | n > 0 \} \]

Set \( \Sigma' \) is

\[ \{ a', b', c' \} \]

Set \( \Sigma'' \) is

\[ \{ a'', b'', c'' \} \]

Set type-1 grammar \( G' \) is

\[ S' \to a' S' B' C' \]
\[ S \to a' B' C' \]
\[ a' B' \to a' b' \]
\[ b' B' \to b' b' \]
\[ b' C' \to b' c' \]
\[ c' C' \to c' c' \]

Set type-1 grammar \( G'' \) is

\[ S'' \to a'' S'' B'' C'' \]
\[ S \to a'' B'' C'' \]
\[ a'' B'' \to a'' b'' \]
\[ b'' B'' \to b'' b'' \]
\[ b'' C'' \to b'' c'' \]
\[ c'' C'' \to c'' c'' \]

Set type-1 grammar \( G_0 \) is

\[ S \to \varepsilon | S_1 S_2 \]
\[ S_1 \to S' | S'' \]
\[ S_2 \to S'' | S' \]

\[ S \to a' B' C' \]
\[ B' \to B' C' \]
\[ b' B' \to b' b' \]
\[ b' C' \to b' c' \]
\[ c' C' \to c' c' \]

\[ S \to a'' B'' C'' \]
\[ B'' \to B'' C'' \]
\[ b'' B'' \to b'' b'' \]
\[ b'' C'' \to b'' c'' \]
\[ c'' C'' \to c'' c'' \]

\[ S \to \varepsilon | S_1 S_2 \]

\[ S_1 \to S' | S'' \]
\[ S_2 \to S'' | S' \]

\[ S \rightarrow a' S' B' C' \]
\[ S \rightarrow a'' B'' C'' \]

\[ \varepsilon \text{ is generated, add } S \to r \]

in which

\[ S_1 \to r \in P_1 \]

to deduce \((r \bowtie w B \text{ or } r=whw)\).

For every formula like \( A \rightarrow w \), add

\[ A \rightarrow w S_1 (A \rightarrow w \text{ is not deleted}) \]

from \( S \), the sentence pattern could be deduced,

\[ r_1 r_2 \cdots r_k A \]

in which

\[ r_1, r_2, \ldots, r_k \in L_1 \]

Stop deduction when

\[ r_1 r_2 \cdots r_k w \]

is deduced or having deduced another sentence from

\[ r_1 r_2 \cdots r_k w S_1 \]

until \( L_1^* \).

\( G_1 \) is type-3 grammar, create \( -3 \text{type grammar}, \)

\[ G_{10} = (\Sigma, V, F, \{S\}, S, P_{10}) \]

in which

\[ P_{10} = \{ S \to \varepsilon \} \cup \{ P_1 - \{ S_1 \to \varepsilon \} \} \]
\[ \cup \{ S \to r | S_1 \to r \in P_1 \} \]
\[ \cup \{ A \to w S_1 | A \to w \in P_1 \} \]

then

\[ L(G_{10}) = L_1^* \]

So, language of type-3 is closed on Kleene Closure operation.
For example, type-3 grammar $G_1$ is

\[
\begin{align*}
S_1 & \rightarrow aS_1bS_1 \\
S_1 & \rightarrow aA\mid bB \\
A & \rightarrow aA\mid bA \\
A & \rightarrow aC \\
B & \rightarrow ab\mid bB \\
B & \rightarrow bC \\
C & \rightarrow a\mid ab \\
C & \rightarrow aS_1\mid bS_1
\end{align*}
\]

so $L_1$ is

\[(a+b)^\ast a(a+b)^\ast a(a+b)\ast (a+b)^\ast b(a+b)^\ast b(a+b)^\ast \]

Set type-3 grammar $G_{10}$ is

\[
\begin{align*}
S & \rightarrow \varepsilon \\
S & \rightarrow aS_1\mid bS_1 \\
S & \rightarrow aA\mid bB \\
A & \rightarrow aA\mid bA \\
A & \rightarrow aC \\
B & \rightarrow ab\mid bB \\
B & \rightarrow bC \\
C & \rightarrow a\mid ab \\
C & \rightarrow aS_1\mid bS_1
\end{align*}
\]

so $L_{10}$ is

\[(a+b)^\ast a(a+b)^\ast a(a+b)\ast (a+b)^\ast b(a+b)^\ast b(a+b)^\ast \]

that is

$L_{10} = L_{1\ast}$

Therefore, whether alphabet

$\Sigma_1 \cap \Sigma_2 = \emptyset$

or

$\Sigma_1 \cap \Sigma_2 \neq \emptyset \text{ (} \Sigma_1 = \Sigma_2 \text{ is included)}$

language of type-0, type-1, type-2 and type-3 are closed on union, product and Kleene Closure operations.

VI. THE CREATION OF REGULAR EXPRESSION

For regular language, regular expression can be generated as the method above.

$R_1$ and $R_2$ are regular expressions of language $L_1$ and $L_2$.

Suppose

$L = L_1 \cup L_2$

regular expression of $L$ is $(R_1) + (R_2)$

$L = L_1L_2$

regular expression of $L$ is $(R_1)(R_2)$

$L = L_1^\ast$

VII. CFL IS EFFECTIVELY CLOSED TO CONTEXT-FREE IN-PLACEMENT

For context-free language, there is another useful operation, that is in-place operation[1].

Suppose $X$ and $Y$ are alphabets, mapping

$g: X \rightarrow Y^\ast$

if

$g(\varepsilon) = \varepsilon$

and for any $n \geq 1$

$g(x_1x_2\cdots x_n) = g(x_1)g(x_2)\cdots g(x_n)$

in which

$x_i \in X$

$g(x_i) = y \in Y^\ast$

or

$g(x_i) = \{y_1, y_2, \cdots\}$

then $g$ is a context-free in-placement.

If $L$ is a language of alphabet $X$, then

$g(L) = L_\ast$

in which

$w \in L$

Context-free grammar $G = \langle X, V, S, P \rangle$, generates context-free language $L$, $g$ is a context-free in-placement:

$g(x) = L_\ast$

in which

$x \in X$

Copy $X$ as $X'$

$X' = \{x' \mid x \in X\}$

for every formula of $P$, replace the terminal symbol $x$ on the right side by $x'$, and $P'$ is obtained.

Rewrite $G$ as:

$G' = \langle Y, V \cup \Sigma, S, P' \rangle$

The language generated by grammar $G$ is based on alphabet $X$, and the language generated by grammar $G'$ is based on alphabet $X'$. The sentence structures of the languages are all the same. (Only differ in alphabet.)

For every $x'$, add a group of context-free formulas to satisfy:

$x' \rightarrow^* L_\ast$

$P'$ is obtained.

Create context-free grammar, #

$G'' = \langle Y, V \cup \Sigma, S, P'' \rangle$

Grammar $G''$ first uses $P'$ to generate

$x'_1 x'_2 \cdots x'_n$

and then uses the new formulas to obtain

$L_\ast_1 L_\ast_2 \cdots L_\ast_n$

Language $g(L)$ generated by grammar $G''$ is also context-free. For example,

Context-free grammar $G$ generates $a'b^n$ for

$S \rightarrow aSb$

$S \rightarrow a\mid b$

Suppose context-free in-placement is:

$g(a) = 0 = L_\ast_a$

$g(b) = 101^\ast = L_\ast_b$

Create grammar $G'$

$S \rightarrow a' \mid S \rightarrow b'$

$S \rightarrow a' \mid b'$

$a' \rightarrow b^n\mid a^n$

$0^\ast$ is generated.

Add formula

$b' \rightarrow 1010A$
101* is generated
Create G”:

\[
\begin{align*}
S &\rightarrow a\ S\ b' \\
S &\rightarrow a'\ b' \\
a' &\rightarrow 0\ 0\ a' \\
b' &\rightarrow 1\ 0\ 1\ 0\ A \\
A &\rightarrow 1\ \vdash\ A
\end{align*}
\]

language \( \varepsilon(101^*) \)’ is generated.

VIII. CONSTRUCTING NFA WITH THE CLOSURE PROPERTY
Suppose \( L_1, L_2 \) be two type-3 languages, the DFA which receive these two languages is

\[
M_1 = (Q_1, \sum, \delta_1, q_0, \{f_1\})
\]

and

\[
M_2 = (Q_2, \sum, \delta_2, q_0, \{f_2\})
\]

Suppose \( Q_1 \) and \( Q_2 \) not be intersect.

Construct

\[
\varepsilon \text{-NDA} = (Q_1 \cup Q_2, \sum, \delta, q_0, \{f_1\} \cup \{f_2\})
\]

function \( \delta \) is:

\[
\begin{align*}
\delta(q_0, \varepsilon) &= q_1 \\
\delta(q_0, a) &= q_2
\end{align*}
\]

to all states \( q \in Q_1, a \in \sum \), \( \delta(q, a) = \delta(q, \varepsilon) = \delta_1(q, a) \)

to all states \( q \in Q_2, b \in \sum \), \( \delta(q, b) = \delta_2(q, b) \)

This can be shown visually as Fig.1.

\[
\text{Figure 1e} \hspace{1cm} \text{-NDA for union operator}
\]

This \( \varepsilon \) -NDA concludes all function \( \delta \) of \( M_1 \) and \( M_2 \), and adds \( 4 \delta \) functions that scan \( \varepsilon \), then we get: setting out from the \( \varepsilon \) -NDA beginning appearance, passing two actions:

\[
\begin{align*}
\delta(q_0, \varepsilon) &= q_1 \\
\delta(q_0, a) &= q_2
\end{align*}
\]

can arrive the beginning appearance \( q_1 \) or \( q_2 \) of \( M_1 \) or \( M_2 \), then, with the usage of own \( \delta \) function that belong to \( M_1 \) or \( M_2 \), it can reach the only receiving states \( f_1 \) or \( f_2 \), finally, enter the only receiving states \( f_0 \).

Obviously, the language that \( \varepsilon \) -NDA receive is union of \( L(M_1) \) and \( L(M_2) \).

Construct

\[
\varepsilon \text{-NDA} = (Q_1 \cup Q_2, \sum, \delta, q_0, \{f_1\} \cup \{f_2\})
\]

function \( \delta \) is:

\[
\begin{align*}
\delta(q_0, \varepsilon) &= q_1 \\
\delta(q_0, a) &= q_2
\end{align*}
\]

to all states \( q \in Q_1 \), \( a \in \sum \), \( \delta(q, a) = \delta(q, \varepsilon) = \delta_1(q, a) \)

to all states \( q \in Q_2 \), \( b \in \sum \), \( \delta(q, b) = \delta_2(q, b) \)

This can be shown visually as Fig.3.

\[
\text{Figure 3e} \hspace{1cm} \text{-NDA for Kleene Closure operator}
\]

This \( \varepsilon \) -NDA concludes all function \( \delta \) of \( M_1 \), and adds \( 4 \delta \) functions that scan \( \varepsilon \), then we get: setting out from the \( \varepsilon \) -NDA beginning appearance, passing two \( \varepsilon \) actions:

\[
\begin{align*}
\delta(q_0, \varepsilon) &= q_1 \\
\delta(q_0, a) &= q_0
\end{align*}
\]

can arrive the beginning appearance \( q_1 \) or \( q_2 \) of \( M_1 \) or \( M_2 \), then, with the usage of own \( \delta \) function that belong to \( M_1 \) or \( M_2 \), it can reach the only receiving states \( f_1 \) or \( f_2 \), finally, enter the only receiving states \( f_0 \).

Obviously, the language that \( \varepsilon \) -NDA receive is the Kleene Closure of \( L(M_1) \).
IX. CONCLUSION

Usually, complex language could be decomposed into several simple languages of the same type and recomposed by union, product and Kleene closure operations. The paper proves that the 4 types of language of Chomsky theory are effectively closed on the above three operations, and proposes a general method to create grammar of complex languages.

The valid closure property of positive closure operation can be referred to the effective closure of Kleene closure without considering the generation of sentence $\varepsilon$.

The closure of other operations, like intersection and complementary operations are not discussed in this paper.

We can construct NFA with the closure of language calculation.

ACKNOWLEDGMENT

This paper is supported by the the Si Chuan science and technology (2006J13-068).

REFERENCES


Chen Wenyu, male, born in 1968. He is an associate professor of School of Computer Science and Engineering, UESTC. His research interests include computer Language and compile, formal language and automation, and neural networks.

He is a mentor for postgraduate student; a grade college teacher (for undergraduate); a quality outstanding young teacher (for postgraduate).

He completed a number of research tasks, both through the Ministry, the provincial identification; publish over 20 papers and 6 teaching materials.

Associate prof. Chen is a senior member of China Computer Federation(E2000111786S).

Wang Xiaobin, male, born in 1964.

He is an associate professor of School of Computer Science and Engineering, University of Electronic Science & Technology of China. His research interests include data structure, computer Language and compile, software engineering, neural networks, and optimization.

He received the B.S. and M.S. degrees in computer science and engineering from University of Electronic Science and Technology of China, Chengdu, China, in 1985 and 1988, respectively. He is currently working toward the Ph.D. degree in Computational Intelligence Laboratory, School of Computer Science and Engineering, University of Electronic Science and Technology of China. From 1988 to 1998, he was a researcher at the School of Computer Science and Engineering, University of Electronic Science and Technology of China. From 1998, he was a Professor at the School of Computer Science and Engineering, University of Electronic Science and Technology of China. He is currently the dean of ChengDu College, University of Electronic Science and Technology of China.


She is candidate for Master’s degree in Software Engineering, School of Software, University of Electronic Science & Technology of China. Her research interest include: compiling technique, theory of software.

Sun Shixin, male, born in 1940.

He is a Ph.D supervisor and professor of School of Computer Science and Engineering, University of Electronic Science & Technology of China. His research interest include: theory of Computer Science, Parallel Computer Systems, Numerical Algorithm & Non-Numerical Algorithm (including Parallel Algorithm).

Prof Sun Sun 1966. B.A. in Mathematics,Sichuan University, Chengdu, China;1984-1987. visiting scholar at the Grenoble University, France; 1990. doing research at Roma University, Italy and Grenoble; University, France.