MNK Publication

International Journal of Latest Research in Science and Technology **ISSN (Online):2278-5299**
Volume 4, Issue 6: Page No.30-34, November-December 2015 (special Issue Paper)
http://www.mnkjournals.com/ijlrst.htm
Special Issue on International conference on Methematics Sciences-2015(MSC-2015)
Confernece Held at Sadguru Gadage Maharaj College, Karad ,Maharastra,(India)

C-MINUS: FUZZY SCANNER

[1]Vaishali Bhosale, [2]S.R.Chaudhari
[1]Assistant Director/ Assistant Professor YCSRD,Shivaji University, Kolhapurvaishali.p.bhosale@gmail. com
[2]Professor and Head Dept. Of Mathematics, North Maharashtra University, Jalgaon shrikant_chaudhari@yahoo.com

Abstract- ***in this paper we describe the design and implementation of scanner for C-MINUS programming language. This is extended to fuzzy scanner implementation using the concepts like fuzzy regular expression, fuzzy non-deterministic automata, fuzzy deterministic automata and it's minimization. The aim of this paper is to allow fuzzy tokens due to insertion, deletion, substitution, typing and letter sequencing errors.***

Keywords - Fuzzy lexical analysis, Fuzzy finite automata, Fuzzy regular expression, Fuzzy tokens, Tiny compiler

I. INTRODUCTION

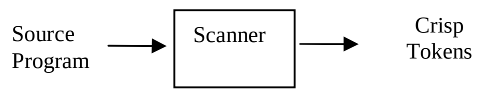
This paper is in parallel with the paper [2], which describes the design and implementation of fuzzy lexical analyzer for Tiny language. Scanning is a phase of compiler where input file is scanned character by character to separate the tokens viz. keywords, identifier, operators etc.. Scanning is also called as lexical analysis. In traditional scanning, a string is either a token or a non-token, and hence there is no middle possibility [3]. Whereas in fuzzy scanning a token may belong to more than one token type with varying degree of membership in [0, 1].

In 'C' programming language, if you type "character" (due to substitution) it does not mean that it is the keyword "char" to the compiler, but it is treated as an identifier only. If you type "charrrr" (an insertion error may be due to the key sticks), again it will also not treated as "char". If you type "whil" it does not mean "while" to the compiler but as an identifier only (deletion error). Also if you type "wlse" it does not mean "else" to the compiler (a typing error). If you type "lese" it does not mean "else" to the compiler (letter sequencing error).

Would it be more friendly if the compiler will simply decide for you "char" in first two cases, "while" in the third case and will it ask you whether you meant "else" in last two cases? The answer probably would be 'no' with existing compilers. Fuzzy scanning will make it possible.

PRELIMINARY

Regular languages are represented by regular expressions and they are analyzed by scanning. The model of scanning is shown in the following Figure 1. [11]



The familiarity with the basic definitions of formal language theory[] is assumed here. Definitions for fuzzy language are given below:

Definition 2.1: [3] Let ∑ be a finite alphabet and f: ∑*→[0,1]. Then set Ã = { (w,f(w)) | w ϵ ∑* } is called a fuzzy language over ∑ and f the membership function of Ã.

Definition 2.2: [3] Let Ã be a fuzzy language over ∑ and f Ã : ∑*→[0 , 1] the membership function of Ã. Language Ã

can be called a regular fuzzy language, if for each m ϵ M , S Ã (m) is regular, where S Ã (m) = { w ϵ ∑* | f Ã (w)=m }.

Definition 2.3: [3] Let e be a regular expression over ∑ and m Є [0, 1]. Then (e) / m is a fuzzy regular expression. If e1, e2, . . . , en are fuzzy regular expressions over ∑ and m1, m2,. . . , mn are their respective degrees, then one can write it as

e1 / m1 + e2 / m2 + . . . + en / mn such form of writing of regular expression is called normalized form of the fuzzy regular expression.

Definition 2.4: [12] A nondeterministic finite automata (NDFA) is a 5-tuple, (Q, Σ, δ, q0, F), where Q is finite nonempty set of states; Σ is finite nonempty set of inputs; δ is the transition function mapping from Q × Σ into 2Q , q0 ∈ Q is initial state; and F is subset of Q is the set of final states.

Instead of F as subset of Q, if F is a fuzzy subset of Q, then the nondeterministic finite automata is treated as a nondeterministic automata with fuzzy final states (NFA-FS). The fuzzy language accepted by Ã, is denoted by L(Ã), is the set { (x, d Ã (x)) | x ϵ ∑* }, where d Ã (x) = max { F Ã (q) | (s, x, q) ∈ δ* }.

Definition 2.5: [12] A deterministic finite automata (DFA) is represented by a 5-tuple, (Q, Σ, δ, q0, F), where Q is finite nonempty set of states; Σ is finite nonempty set of inputs; δ is the transition function mapping from Q × Σ into Q and is usually called direct transition function. This is the function which describes the changes of states during the transition. The mapping δ is usually represented by a transition table or a transition diagram. q0 ∈ Q is initial state; and F is subset of Q is the set of final states.

If F is fuzzy subset of Q instead of crisp subset of Q, then the DFA is called deterministic finite automata with fuzzy final states (i.e. FS-DFA). The fuzzy language accepted by Ã denoted by L(Ã), is the set { (x, d Ã (x)) | x ϵ ∑* }, where

$$d_{\tilde{A}}(x) = \begin{cases} F(q), \text{ if } \delta(s,x) = q \\ 0 \text{ if } \delta(s,x) \text{ is not defined} \end{cases}$$

Theorem 2.1: [12] For every NDFA, there exists a DFA that simulates the behavior of NDFA. The converse of the theorem is trivial.

FUZZY LEXICAL ANALYSIS

In designing scanner for C-MINUS language first regular expressions are defined based on lexical conventions of it and then NFA designed and converted into equivalent DFA and minimized it further if possible. The implementation is possible using Lex tool or using switch case statements in "C" programming language. The diagram below gives DFA for tokens in C-MINUS.
To allow flexibility in scanning fuzzy regular expressions are defined for tokens in C-MINUS.

Construction of Fuzzy regular expressions:
Firstly fuzzy regular expressions will be constructed for keywords that exists in C- Minus language [1] which allow insertion, deletion, substitution, letter sequencing and typing errors. Also this compiler allows synonyms for keywords wherever possible. We now construct FREs for all the keywords:

a) The FREs for reserved word "**if**":

if / 1 + $(ii^{+}ff^{*} + ii^{*}ff^{+})$ / m

Only insertion error is considered for keyword "if". For the sake of convenience no deletion, substitution, letter sequencing and typing errors will allowed for "if" the keyword, as it has string length two.

b) The FREs for reserved word "**else**":

else / 1 + $(ee^{+}ll^{*}ss^{*}ee^{*} + ee^{*}ll^{+}ss^{*}ee^{*} + ee^{*}ll^{*}ss^{+}ee^{*} + ee^{*}ll^{*}ss^{*}ee^{+})$ / m1 + (ee*ll*ss*e* + ee*ll*s*ee* + e*ll*ss*ee*) /m2 + $(e+l+s+e)^{+}$/m3 + ((e+w+d+r)lse + e(k+o+p+l)se +el(a+w+d+s)e+els (e+w+d+r))/m4

The fuzzy regular expressions can be simplified and put together as :

else / 1+ $(e+l+s)^{+}$/m1+((r+w+d+e)lse +e(k+o+p+l)se +el(a+w+d+s)e+els (e+w+d+r)) /m2

Hence onwards we give only simplified fuzzy regular expressions for the remaining keywords.

c) The FREs for reserved word "**int**":

int / 1 + $(i+n+t)^{+}$/m1 + ((i+u+k+o)nt+i(n+b+h+m)t+in(t+r+g+y))/m2 + (integer)/0.6

d) The FREs for reserved word "**return**":

return / 1 + $(r+e+t+u+r+n)^{+}$ /m1 + ((r+e+t+f)eturn + r(e+w+d+r)turn + re(t+r+g+y)urn + ret(u+y+j+i)rn + retu(r+e+t+f)n+ retur(n+b+j+m))/m2

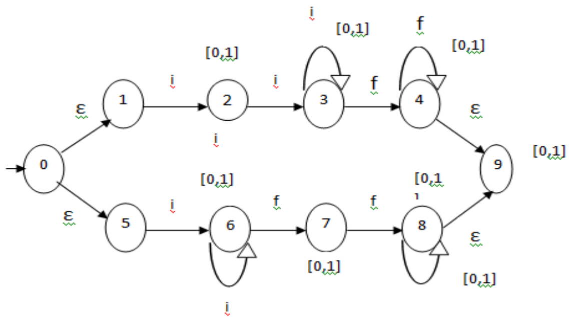
e) The FREs for reserved word "**while**":

while/1+$(w+h+i+l+e)^{+}$/m1+ ((w+q+s+e)hile+w(g+y+j+n+h)ile+wh(i+u+k+o)le+whi(k+o+p+l)e+whil(r+w+d+e))/m2

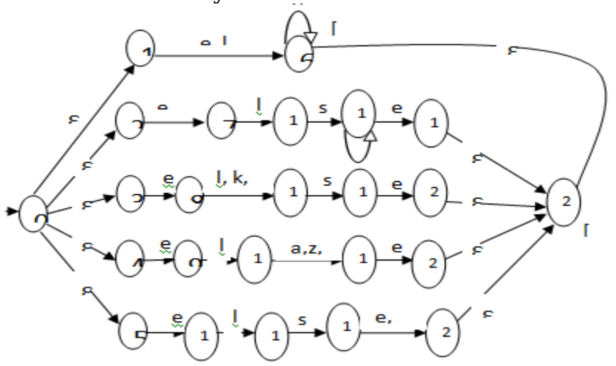
f) The FREs for reserved word "**void**":

void/1 + $(v+o+i+d)^{+}$ / m1 + ((v+c+g+b)oid+v(o+i+l+p)id+vo(i+u+k+o)d+voi(d+s+c +f+e))/m2

Construction of fuzzy state NFA from above Fuzzy regular expressions:



Fuzzy state NFA for "if"



Fuzzy state NFA for "else"

Conversion of fuzzy state NFA to fuzzy state DFA:

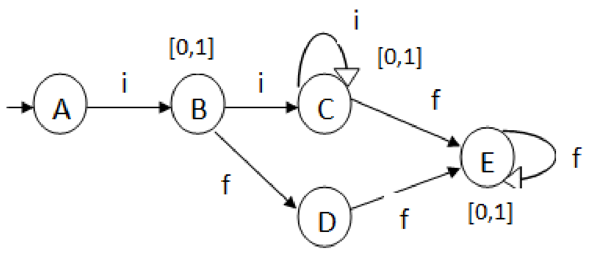
s\∑	i	f
A	B	-
B	C	D
C	C	E
D	-	F
E	-	G
F	-	F
G	-	G

Table : fuzzy state DFA for "if"

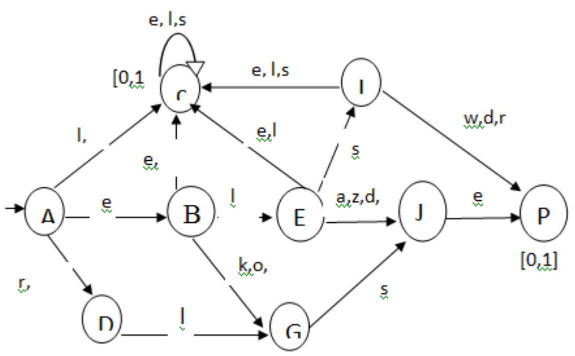
S\∑	e	l	s	w	d	r	a	z	k	o	p
A	B	C	E	D	D	D	-	-	-	-	-
B	F	G	H	-	-	-	-	-	I	I	I
C	J	C	H	-	-	-	-	-	-	-	-
D	-	K	-	-	-	-	-	-	-	-	-
E	E	E	E	-	-	-	-	-	-	-	-
F	F	C	H	-	-	-	-	-	-	-	-
G	J	C	L	L	L	-	L	L	-	-	-
H	J	E	H	-	-	-	-	-	-	-	-
I	-	-	M	-	-	-	-	-	-	-	-
J	J	E	E	-	-	-	-	-	-	-	-
K	-	-	M	-	-	-	-	-	-	-	-
L	N	E	H	O	O	O	-	-	-	-	-
M	O	-	-	-	-	-	-	-	-	-	-
N	J	E	E	-	-	-	-	-	-	-	-
O	-	-	-	-	-	-	-	-	-	-	-

Table : Fuzzy State DFA for "else"

Construction of minimized fuzzy state DFA:



Minimized fuzzy state DFA for "if"



Minimized fuzzy state DFA for "else"

In the section below fuzzy regular expressions are given for C-MINUS tokens , followed by representative FS-NFA for keywords if and else, followed by transition table representation of their fuzzy state state DFA and finally minimized DFA for them are represented as diagram.

3. Implementation of fuzzy lexical analyzer:

This section explains full implementation of fuzzy scanner for C-MINUS language. In the previous section, fuzzy regular expressions (FREs) for all keywords in C-MINUS are defined. In the implementation phase fuzzy tokens themselves are defined using enumerated types as below:

```
typedef enum
    /* book-keeping tokens */
    {ENDFILE,ERROR,
    /* reserved words */ IF,ELSE,INT,RETURN,
    VOID, WHILE,
    /* multicharacter tokens */ ID,NUM,
    /* special symbols */ ASSIGN, REQ, LT,
    PLUS, MINUS, TIMES, OVER, LPAREN,
    RPAREN, SEMI, LTEQ, GT, GTEQ, NTEQ,
    COM,SQLPAREN, SQRPAREN, CRLPAREN,
    CRRPAREN
    } TokenType;
```

In the steps above first FS-NFA for each FRE have designed. FS-NFA are then converted to FS-DFA . Then minimized FS-DFA for each fuzzy keyword is constructed. Fuzzy lexical analyzer implemented using switch-case constructs of 'C' programming language. The table "KeyWords" stores keyword structures as given below:

```
static struct
  { char* str; TokenType tok;   }
reservedWords[MAXRESERVED]
  ={{"if",IF},{"else",ELSE},{"int",INT},
{"return",RETURN}, {"void",VOID},
     {"while",WHILE} };
```

In the implementation step sequence of alphabets is accepted as identifier first. The procedure call performs a lookup of crisp keywords, substitution keywords by string comparison.

```
strcmp(input_string, KeyWords)
    return KeyWords;
```

If no match found, then the “reserved_lookup” calls “check_fuzzy” function is used to check the token for fuzzy keyword due to insertion, deletion, letter sequencing and typing errors, if any.

```
currentToken=checkfuzzy();
    return currentToken;
```

Again if there also no match found, then current token type retained as an identifier (i.e. ID) only. The experimental results shows that the fuzzy keyword belongs to more than one category with varying degree of membership. Default membership of crisp keyword is 1, for substitution keyword it is predefined as 0.6, for character sequencing error it is 1 and computed runtime for insertion, deletion and typing errors. All fuzzy keywords are identifiers. Therefore all fuzzy keywords have membership value 1 for token type identifier.

Experimental Results: Consider input file contain following strings

If iif else less while whil void coid int integer

Result of crisp lexical analysis:Result of fuzzy lexical analysis:

Sr. No.	Input	μ(keyword)	μ(identifier)
1	if	1	0
2	iif	0	1
3	else	1	0
4	less	0	1
5	while	1	0
6	whil	0	1
7	void	1	0
8	coid	0	1
9	int	1	0
10	integer	0	1

Input	μ(keyword)	μ(identifier)
if	1	0
iif	0.87	1
else	1	0
less	1	1
while	1	0
whil	0.80	1
void	1	0
coid	0.75	1
int	1	0
integer	0.6	1

CONCLUSION

The paper described the possibility of fuzziness in keywords due to insertion, deletion, substitution, typing and letter sequencing errors and their implementation in this paper. The implementation is restricted to those mentioned errors. The synonyms for programming language constructs from natural language can be used to allow fuzzy tokens. The work can be further extended to allow more flexibility in tokens such that the program will look like psuedocode. In this paper the implementation of fuzzy keywords is fully emphasized. The approach is to use fuzzy automata concepts to allow flexibility (or fuzziness) in token recognition process i.e. lexical analysis. It is termed as fuzzy lexical analysis. As a result of fuzzy lexical analysis a token may belong to more than one category with different degree of membership. To finalize the token category the need is to go further and discuss fuzzy parsing. In future the work can be extended for fuzzy parsing that will finalize the token category mainly based on its position in the given sentence. Fuzzy context free grammar will allow fuzziness in syntax analysis phase of compiler in order to model grammatical errors. The work can be extended using fuzzy translation rules for syntax directed semantic analysis for fuzzy relations.

REFERENCES

1. Kenneth C. Louden, Compiler Construction Principles and Practice, Cengage Learning, India Edition, 2008.
2. Vaishali Bhosale, S. R. Chaudhari, Fuzzy Lexical Analyzer: Design and Implementation, 2015.
3. Mateescu A., Salomaa A., Salomaa K., Yu S., Lexical Analysis with a Simple Finite Fuzzy Automaton Model, Journal of Universal Computer Science, 1995, 292-311.
4. Bai M. Q., Sun F., Mo Z., Closure and Commutation of Fuzzy Regular Languages, 2009 IEEE.
5. Bai M. Q., “On the Relation between Fuzzy Regular Expression and Fuzzy Finite State Automata”, Pure and Applied Mathematics, 16(2000)4, pp. 1-6.
6. Gupta M. M., Saridis G. N. and Gaines B. R., Fuzzy Automata and Decision Processes, North-Holland, New York, 1977, pp149-168
7. Kumbhojkar, H. V. and Chaudhari, S. R.(2002b), “Fuzzy Recognizers and recognizable sets”, Int. J. of Fuzzy Sets and Systems, Vol. 131, pp. 381-92.
8. Mordeson J. N., Malik D. S., Fuzzy Automata and Language: Theory and Applications, Chapman & Hall, 1 edition, 2002.
9. Santos E. S. (1968a), “Maxmin automata”, Information and control, Vol.21, pp. 27-47.
10. Wee W. G. and Fu, K. S. (1969), “A formulation of fuzzy automaton and its application as a model of learning systems”, IEEE Trans Syst. Sci. Cyber, Vol. 5, pp. 215-23.
11. Hopcroft, Motwani, Ullman,Introduction to Automata Theory, Languages and Computation, Pearson Education.
12. Mishra K.L.P., Chandrasekaran N., Theory of Computer Science, Prentice Hall of India, 1998.
13. Asveld P.R.J., Fuzzy Context Free Languages – Part 1, Generalised fuzzy context free grammars, Theore. Comp. Sc., volume (2005).
14. Asveld P.R.J., Fuzzy Context Free Languages – Part 2, Recognition and Parsing Algorithms, Theore. Comp. Sc., volume(2005), pp 191-213.

15. Hsuan Shih Lee, Minimizing Fuzzy finite Automaton, 2000 IEEE.

16. Kremer D., New Directions in Fuzzy Automaton, 2005 Science Direct.

17. Lee E. T. and Zadeh L. A., “Note on Fuzzy Languages”, Information Sciences, 1(1969) 421-434.

18. Xuli, Jinga, Diancheng, A Fuzzy Automaton Model and It’s Applications, 1996 IEEE.