TITLE

Fuzzy Lexical Analyzer: Design and Implementation[1]

INTRO

It is important to handle the errors, due to insertion, deletion, letter sequencing, substitution and typing in the lexical analysis phase of compiler. A fuzzy token is a sequence of characters which can have one or more errors due to insertion, deletion, letter sequencing and typing errors. Here, fuzzy automata model is used for accepting fuzzy tokens and a compiler is implemented based on that concept. Fuzzy keywords, their fuzzy regular expressions and minimized fuzzy deterministic automaton are constructed for this purpose. The issue of membership of fuzzy keyword is successfully solved by using an algorithm. In typical lexical analysis every token belongs to only one type - keywords, identifier, operators etc. Where it’s default membership value is set to 1. Whereas in fuzzy lexical analysis a token may belong to more than one token type with changing degree of membership

Implementation of fuzzy lexical analyzer is also described along with the algorithm used.

ALGORITHM

An algorithm for fuzzy lexical analysis is given below.

Step 1: Find length L of each keyword.

Step 2: Find occurrences of each letter l in the crisp keyword as

O( l ).

Step 3: Find degree of each letter for fuzzy token as

D( l )=(1/L)\*O( l )

Step 4: Initialize A ( l ) = 0.0 for all letters in crisp keyword and

M (keyword) = 0.0

Step 5: For each letter „l‟ in input string if „l‟ is in crisp keyword

then update

A ( l ) = A ( l ) + 1

Step 6: Compute actual degree of each letter d ( l ) as below:

If A( l ) > 0 then

{ If (A ( l ) > O ( l ) then D ( l )= D ( l ) /A ( l )

Else if A ( l ) = = O ( l ) then D ( l )= D( l )

Else D ( l )= (A ( l ) / O ( l )) \* D ( l )

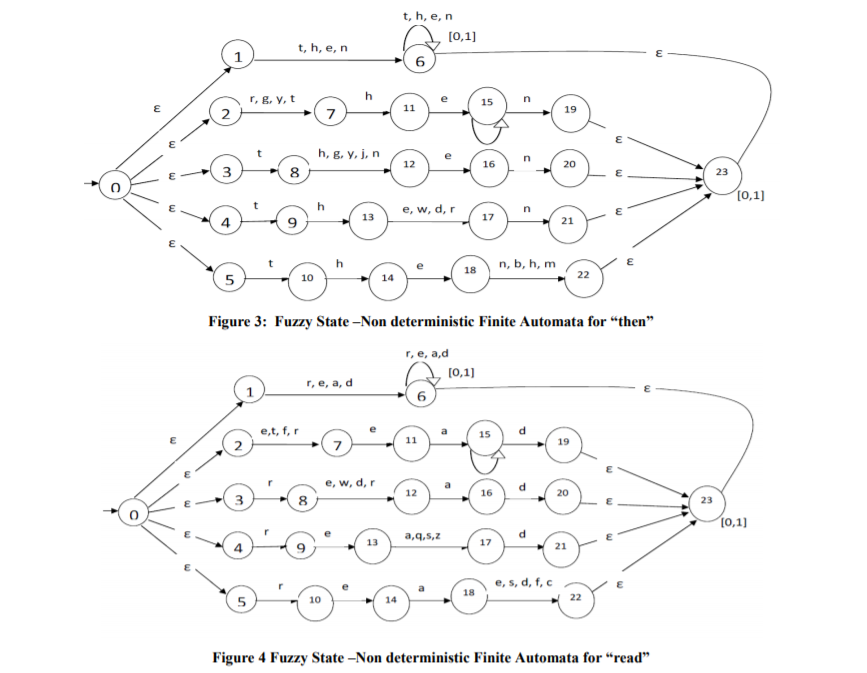
}

Step 7: M ( l ) = sum (D ( l ) ) for all l in crisp keyword

Step 8: If M ( l ) > 0.5, then it is recognized as a fuzzy keyword

else as an identifier.

ARCHITECTURE



APPLICATION

The synonyms for programming language constructed from natural language can use fuzzy tokens concept for use. The work can be further upgraded to allow more flexibility in tokens such that the program will look like a psuedocode. Use fuzzy automata concepts to allow the flexibility in token recognition process i.e. lexical analysis. This can be used 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.

TITLE

A Multi-Language Lexical Analysis Package[2]

INTRO

The multi-language lexical analysis package called Lexical Analyzer G which presents itself as a generic c unit, for which the generic parameters are subprograms into modules for input system .

Lexical analysis techniques are well known but they are difficult and somewhat tricky to realize. This analysis package concentrates on the semantics rather than the pattern matching aspect of the problem . Despite the Lexical Analyzer generators are often slower than handwritten analyzers, they are still of interest, for they have no errors and can be ready in a very short time .

DESIGN

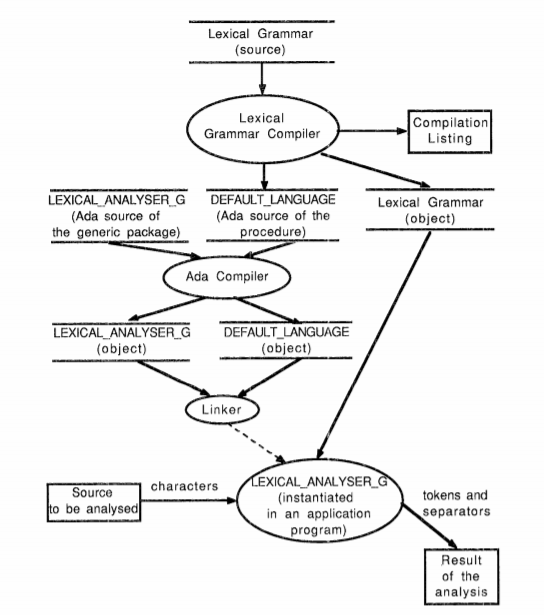
Starting from a lexical grammar, the compiler generates two files that contains the necessary tables to run a lexical analyzer. One of the files contains these tables in binary form, while the other is the proper body of an Ada procedure format . This procedure has the instructions required to fill the tables of the lexical analyzer further. There are two possibilities to fill it out, considering the two files for initializing the analyzer and therefore of choosing of language :

1. a binary file can be created while the application is running ;

b) the procedure can be compiled along with the application and then displayed when running.

The second alternative (b) is more efficient in because no files have to be read in it.

Data Flow Diagram



APPLICATION

Here the lexical analyzer presented is already being used in several applications and for the construction of new tools. For example : the lexical analyzer of a grammar compiler, an analyzer for arithmetic expressions in a spreadsheet database and a generator of command language analyses . The lexical analysis package is used to create the syntactic analysis package . This syntactic analyzer can further invoke actions during the analysis that have been compiled with syntactic constructions of the language of grammar .

TITLE

Dynamic Installation and Configuration of Functionally Extended, Efficient Lexical Analyzers[3]

INTRO

The module LEXXO renders a functionally that is upgraded and efficient alternative to common scanner generators. Through a procedural interface, lexical analyzers can be installed in a program and dynamically programmed to recognize various text patterns in the code. The restriction to a fixed set of basic tokens allows them to deliver not only token codes and lexemes but also the corresponding attribute values. Due to a very general structure of the token syntax details, LEXXO analyzers are adaptable to most requirements and powerful enough to recognize the basic symbols of Pascal or C . A window and mouse based interactive front end gives the user with the possibility to try out different configurations of an analyzer and display their effects when dealing with tokens. It also has the competence to generate code for the LEXXO interface.

IMPLEMENTATION

The module LEXXO is implemented in C under UNIX as a basic data type. Basically, analyzers are opaque handles used so that it can be instantiated, manipulated and released by LEXXO inbuilt functions. In every request sent, the instance handle has to be passed as an additional required parameter . Internally, these handles are basically pointers that dynamically allocate data structures which contain the information of the corresponding analyzer such as configuration parameters, current input stream data and position therein, etc. This is used when someone calls a function to set the input stream, to install a special type of buffer and to enable the standard tokens and white space where he desires . He may then use the separate functions to specify the syntax details of every symbol entered where the defaults are not appropriately mentioned. C's capability of different length argument lists allows a client to pass only the mere minimum details of configuration parameters and to receive default values for the remaining ones.

DESIGN

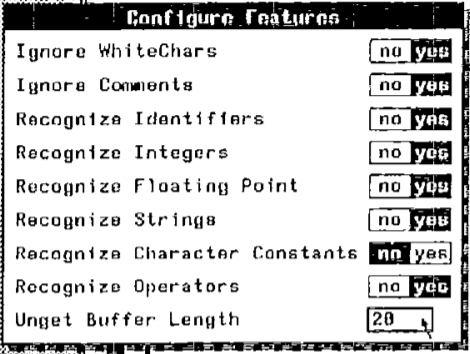


Fig. 1: Feature configuration in LexTool

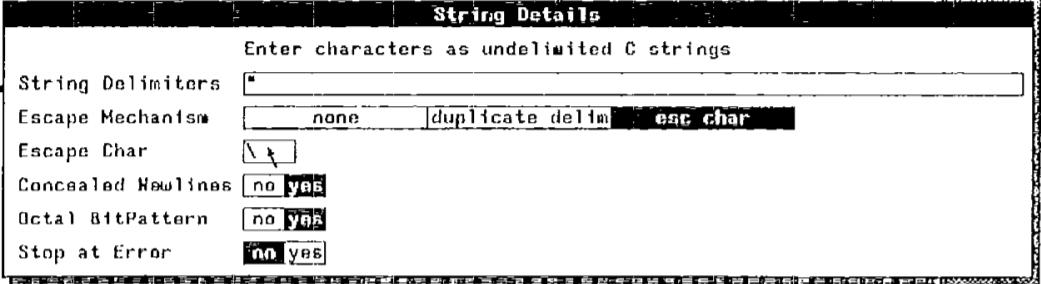


Fig. 2: String detail specification in LexTool

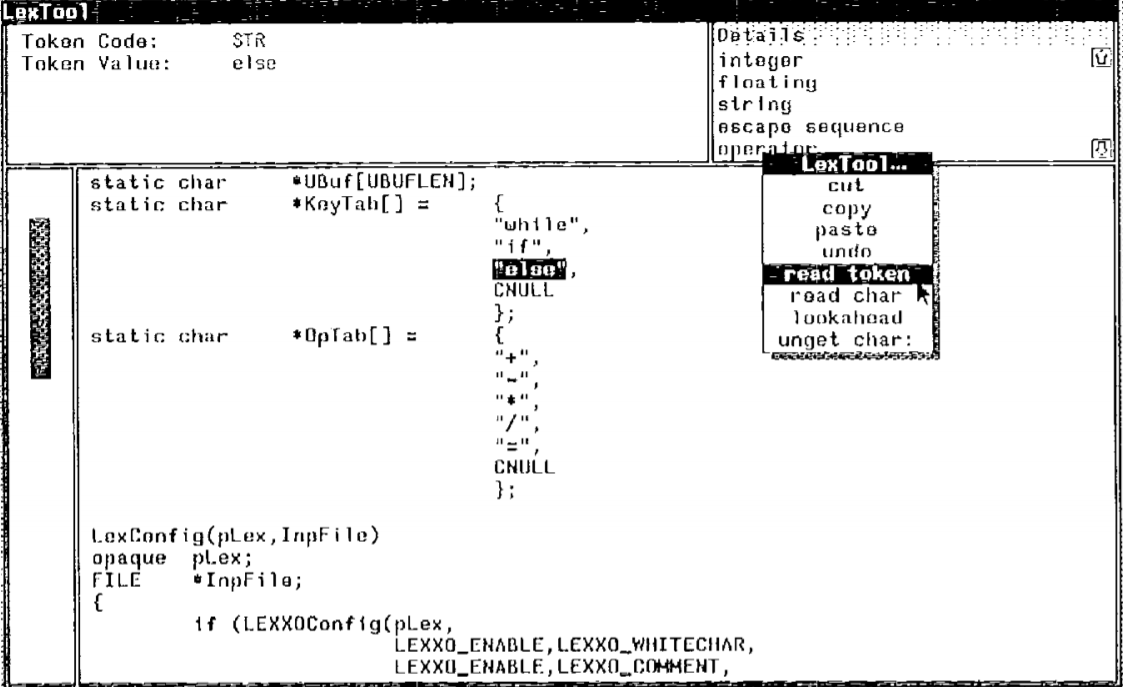


Fig. 3: Trying out an analyzer in LexTool

APPLICATION

Doing with the practical completeness of regular expressions, allows it to bring in more functionality and efficiency. Every token can be delivered with its attribute value, revealing a user from the routine task of converting the lexemes frequently . Many important features such as multiple analyzer instances per process, different types of input streams , dynamic reconfiguration or special buffers have been added to meet practical requirements . This interactive interface makes it easy to use with different configurations, sort them instantly on any text and generate the code for the procedural interface. This considerably simplifies the design of the lexical structure of a new language.

References

1. Vaishali Bhosale,S.R. Chaudhari- Fuzzy Lexical Analyzer: Design and Implementation. International Journal of Computer Applications (0975 – 8887) Volume 123 – No.11, August 2015
2. Steven Ha den Swiss Federal Institute of Technology (EPFL )-A Multi-Language Lexical Analysis Package. Ada Letters, January/February 1990 Volume X, Number 1
3. Peter Schnorf- Dynamic Installation and Configuration of Functionally Extended, Efficient Lexical Analyzers. SIGPLAN Notices, Vol . 23, No . 10