Regular Expression to DFA

Jun 21, 2011
By ProjectsGeek

Regular Expression to DFA

Aim : Regular Expression to DFA (To be taken from compiler point of view)

Objective: – To understand the role of regular expressions and finite automata in applications such as Compilers.

Theory: –

Regular expressions are used to specify regular languages and finite automata are used to recognize the regular languages. Many computer applications such as compilers, operating system utilities, text editors make use of regular languages. In these applications, the regular expressions and finite automata are used to recognize this language.

Compiler is a program which converts the given source program in high-level language into an equivalent machine language. While doing so, it detects errors and reports errors. This process is quite complex and it is divided into number of phases such as Lexical Analysis, Syntax and Semantic Analysis, Intermediate Code generation, Code Generation and Code Optimization.

The lexical analysis phase of compiler reads the source program, character by character and then groups these characters into tokens which are further passed to next phase, which is nothing but parsing or syntax or semantic analysis. After syntax and semantic analysis, Intermediate Code is generated which is followed by actual code generation.

Lexical Analyzer recognizes the tokens from series of characters. A "C" program consists of tokens such as Identifiers, Integers, Floating Point Numbers, Punctuation symbols, relational and logical and arithmetic operators, keywords and comments (to be removed). To identify these tokens, lexical analyzer needs the specification of each of these symbols. The set of words belonging to a particular token type is a regular language. Hence each of these token types can be specified using regular expressions. For example, consider the token Identifier. In most of the programming languages, an identifier is a word which begins with an alphabet (capital or small) followed by zero or more letters or digits (0..9). This can be defined by the regular expression

(letter) . (letter | digit)* where

letter = A|B|C|……|Z| a| b |c |……|z

and digit = 0|1|2|….|9

One can specify all token types using regular expressions. These regular expressions are then converted to DFA's in the form of DFA transition table. Lexical analyzer reads a character from a source program and based on the current state and current symbol read, makes a transition to some other state. When it reaches a final state of DFA, it groups the series of characters so far read and outputs the token found.

Formal definition of Regular expression

The class of regular expressions over ∑ is defined recursively as follows:

1. The letters φ and Є are regular expressions over ∑ .

2. Every letter 'a' c ∑is a regular expression over ∑ .

3 If 'R1' and 'R are regular expressions over ∑, then so are '(R1|R2)', '(R1.R2)' and (R1)*

Where

'|' indicates alternative or parallel paths.

'.' Indicates concatenation

'*' indicates closure

4. The regular expressions are only those that are obtained using rules (1) and (2).

Formal definition of DFA:

The formal definition of finite automata is denoted by a tuple

(Q, ∑,d, qo, f)

Where

Q à Finite set of table

∑à finite input alphabet

qoà Initial state of FA,qo, qo Q

Fà set of final states, F c Q

dà Transition function called as state function mapping

Q * ∑ à Q

i.e. d= Q*∑à Q

A FA is called deterministic (DFA) if from every vertex of its transition graph, there is an unique input symbol which takes the vertex state to the required next state.

DFA is constructed directly from an augmented regular expression (r)#. We begin by constructing a syntax tree T for (r) # and then we compute four functions Nullable, Firstpos, Lastpos and Followpos. The functions Nullable, Firstpos, Lastpos are defined on the nodes of a syntax tree and are used to compute Followpos which is defined on set of positions. We can short circuit the construction of NFA by building the DFA whose states correspond to the sets of positions in the tree. Positions, in particular , encode the information regarding when one position can follow another. Each symbol in an input string to a DFA can be matched by certain positions. An input symbol 'c' can only be matched by positions at which there is a 'c' but not every position with a 'c' can necessarily match a particular occurrences of 'c' in input stream.

Algorithm

The steps in algorithm are

Tags

.Net Project
Android Pro
Codes C & C++ Pr
Class Diagrams Comp
Datastructur
Download Visu
Java Proje
Problems php Pro
Papers Projec

Projects
Like P

Be the first of your fr

1. Accept the given regular expression with end of character as #
2. Covert the regular expressions to its equivalent postfix form manually. (students need not write the code for converting infix to postfix but, they can directly accept postfix form of the infix expression)
3. Construct a syntax tree from the postfix expression obtained in step 2.
4. Assign positions to leaf nodes
5. Compute following functions.

Nullable, Firstpos, Lastpos, Followpos

Computation of Nullables : All nodes except the * nodes are not nullable.

Also if some leaf node is for ἐ, then it is also nullable.

Firstpos (Firstposition): At each node n of the syntax tree of a regular expression, we define a function firstpos(n) that gives the set of first positions that can match first symbol of a string generated by sub expression rooted at 'n'.

Lastpos (lastposition) : At each node n of the syntax tree of a regular expression, we define a function lastpos(n) that gives the set of last positions that can match last symbol of a string generated by sub expression rooted at 'n'.

To compute firstposition and last position, we need to know which nodes are the roots of sub expression that generate languages that include the empty string. Such nodes are Nullable. We define nullable(n) to be true if node 'n' is nullable , false otherwise.

Computation of Followpos : Followpos(i) tells us what positions can follow position *i*

in the syntax tree. This can be computed as follows.

1. if n is a '.' (cat) Node, with a left child C1 and right child C2 and i is a position in the Lastpos(C1), then all positions in Firstpos(C2) are in Followpos(i)

2. if n is a * (closure) Node and i is a position in the Lastpos(n), then all positions in Firstpos(n) are Followpos(i)

6. Construct DFA from Follow Pos**.**

Note : Step 5 can be done during construction of tree, since you are building the tree from bottom to top, and when computations at some root of sub tree are to be done, information of sub tree is available. So no need to do any traversal.

Data Structures:

Node Structure for Parse Tree

{

Leftchild and Rightchild : pointers to the node structure

Nullable : Boolean Type

Data : Character Type

Fistpos and Lastpos : set of integers

Pos : integer (this may or may not be part of tree node)

}

Stack : Stack is required to build the tree. This can be implemented either using link list (preferable) or as an array. Item or data that will be pushed into or popped out from stack is pointer to the node structure of a Parse Tree and not just a single character..

Computations of Firstpos and Lastpos.

Node n	Nullable(n)	Firstpos(n)	Lastpos(n)
N is a leaf labeled ἐ	**true**	**ø**	**ø**
N	Nullable(c1) **or** Nullable (c2)	Firstpos (C1) U Firstpos(C2)	Lastpos (C1) U Lasttpos(C2)
N is a leaf labeled with position i	**false**	{ i }	{ i }
n	Nullable(c1) **and** Nullable (c2)	If nullable (C1) then Firstpos (C1) U Firstpos(C2) else Firstpos(C1)	If nullable (C2) then Lastpos (C1) U Lastpos(C2) else

			Lastpos(C2)
n	**true**	Firstpos (C1)	Lastpos (C1)

Algorithm for construction of DFA transition table

1. Initially , the only unmarked state in Dstates is firstpos(root), where root is the root of a syntax tree.
2. While there is an unmarked state T in Dstates do

Begin

Mark T

For each input symbol a do

Begin

Let U be the set of positions that are in Followpos(P) for some P in T,

such that the symbol at position P is a.

If U is not empty and is not in Dstates then add U as an unmarked

state to Dstates

Dtran [T,a] = U

End

End

« *Macro Processor Algorithm* *Lexical Analyzer for Subset of C* »

4 Comments

JASWINDER SINGH DHILLON REPLY
SEP 04, 2011 @ 18:32
Thanks Dude...!!

ANONYMOUS REPLY
OCT 05, 2011 @ 17:24
This comment has been removed by a blog administrator.

HEMANT KR. SINGH REPLY
JAN 24, 2012 @ 12:26
This comment has been removed by the author.

RAJKIN REPLY
JUL 05, 2015 @ 18:22
Thanks sir but i just want to know if any regular expression have some other operator just like ? [] ^ – Then can i convert regular expression to DFA by using this techniques? Thanks!

Leave a Reply

YOUR NAME *

YOUR EMAIL *

YOUR WEBSITE

POST COMMENT