Birla Institute of Technology, Mesra
Department of Computer Science & Engineering
CS 333 : Compiler Design
TUTORIAL 2 (Lexical Analysis); Jan 22, 2024

Definitions, Algorithms and related information are given at the end of the sheet.

Skill set : Writing regular expressions for PL tokens; ability to manually design a dfa directly from regex; represent a dfa in the 4 array scheme.

P1. Find the errors in the following regular expression specification for floating point numbers, with the underlying alphabet {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, •(decimal point)}. Report errors, if any, by giving instances of a) valid floating point numbers that not generated by the regex, and b) invalid float values that are generated by the regex.

pattern-name	**regex**
float	{digit}*\. {digit}*{digit}+

If the regex has errors, then write the corrected version.

P2. Modify the regular expression of P1 so that floating point values of the following form are also captured over and above the float values covered by P1, {1.2e+3, 0.234e3, .23e-21, 256e-1, ….}.

P2. Write regular expression for recognizing single line comments in C/C++. The action required is to count the number of single line comments in a file when you write the specifications in lex.

P3. Consider the regular expressions for the three distinct tokens given below.

(aba)+ Token 1
(ab*a) Token 2
a(a | b) Token 3

Construct a labeled syntax tree for a combination of any 2 of the three tokens given above. Use the symbol '#' as end marker of a token.

P4. For the syntax tree constructed for 2 tokens (chosen by you) in P3

(aba)+ Token 1
(a(b)*a) Token 2
a(a | b) Token 3

(a) Calculate nullable(), firstpos() and lastpos() information and attach to nodes in the syntax tree
(b) Identify all internal nodes where followpos() needs to be constructed, and compute followpos() at each relevant node

P5. Using the sets computed in P4, construct the DFA using the direct algorithm.
(a) Manually check the correctness of the DFA constructed.
(b) In case there is a final state with more than one token associated with it, produce input string(s) that matches with all the associated tokens.

P6. This question concerns the four array representation scheme for a DFA. The function used. Nextstate(), used is reproduced below. Find the minimal and maximal number of 1-dimensional array references required to find the transition, nextstate[s, a], for a state given s on character c.

```
function nextstate (s, a)
{ if CHECK[BASE[s] + a] == s
  then  NEXT[BASE[s] + a]
  else return (nextstate( DEFAULT[s], a))
}
```

P7. Construct manually the 4 array based representation for the following DFA in terms of Default, Base, Next and Check such that array sizes are as low as possible. The final states are {1, 2}.

States	**a**	**b**	***c***	***d***
0	5	4	-	-
1	1	-	-	1
2	-	-	6	5
3	-	-	-	2
4	5	4	-	1
5	2	3	4	4
6	5	4	-	-

P8. Examine with reason whether the DFA of P7 is minimal? If not then minimize it.

Take Home Assignment : Consider the following 2 regular expressions for identifier and number

letter	**a** | **b** | **c** | **d** | **e** | **f** | **g** | **h**
digit	**1 | 2 | 3 | 4 | 5**
Identifier	letter (letter | digit)*
number	digit(0 | digit)*

You are required to manually go through all the steps starting from

- creation of syntax tree
- direct dfa construction using the 4 sets of information
- minimizing the dfa constructed
- laying out the minimized dfa in 4 array scheme

APPENDIX 1 : DEFINITIONS AND ALGORITHMS

The basic regular expression operators and the language denoted by them. Note that the tool “lex” may offer additional

Table 1 : Regular Expression Operators and Underlying Languages

Expression	**Describes**	**Language**	**Example**
ε	Empty string	{ε}	
c	Any non-meta linguistic character c	{c}	a
r*	Zero or more r’s	L(r)*	a*
r+	One or more r’s	$L(r)^+$	a^+
r1 • r2	r1 followed by r2	L(r1) • L(r2)	a•b or ab
r1 | r2	r1 or r2	L(r1) ∪ L(r2)	a | b
(r)	r	L(r)	(a | b)

Table 2 : Properties of Regular Expression Operators

	Regular expression operators with precedence values			
Operator	Parentheses ( )	Closure * Reflexive closure +	Concatenation •	Alternate |
Precedence	4	3	2	1
Associativity	Left	Right	Left	Left

Table 3 : Rules for computation of the four functions

node n	***nullable(n)***	***firstpos(n)***	***lastpos(n)***	***followpos(i)***
Leaf labeled ϵ	**true**	∅	∅	
Leaf labeled with **a** ∈ ∑ at position i	**false**	**{i}**	**{i}**	
$c_1 \mid c_2$	nullable(c_1) **or** nullable (c_2)	firstpos(c_1) ∪ firstpos(c_2)	lastpos(c_1) ∪ lastpos(c_2)	
$c_1 \bullet c_2$	nullable(c_1) **and** nullable (c_2)	**if** nullable(c_1) **then** firstpos(c_1) ∪ firstpos(c_2) **else** firstpos(c_1)	**if** nullable(c_2) **then** lastpos(c_1) ∪ lastpos(c_2) **else** lastpos(c_2)	**if** i ∈ lastpos(c_1) **then** followpos(i) += firstpos(c_2)
c^*	**true**	firstpos(c)	lastpos(c)	**if** i ∈ lastpos(c) **then** followpos(i) += firstpos(c)
c^+	nullable(c)	firstpos(c)	lastpos(c)	**if** i ∈ lastpos(c) **then** followpos(i) += firstpos(c)

REGEX TO DIRECT DFA ALGORITHM

1. Construct the tree for r# for the given regular expression r.
2. Construct functions *nullable()*, *firstpos(), lastpos()* and *followpos()*
3. Let *firstpos(root)* be the start state. Push it on top of a stack.
   While (stack not empty)
   do begin
      pop the top state U off the stack; mark it;
      for each input symbol **a** do
      begin
        let p_1, p_2, p_3, ... p_k be the positions in U corresponding to symbol **a;**
        Let V = *followpos*(p_1) ∪ *followpos*(p_2) ∪ **.....** ∪ *followpos*(p_k);
        place V on stack if not marked and not already in stack;
        make a transition from U to V labeled **a**;
      end
   end
4. Final states are the states containing the positions corresponding to #.

FOUR ARRAY REPRESENTATION :

Default[] and Base[] are the same size as the number of states. The size of the Next[] and Check[] arrays depend on the extent of exploitation of sparsity and common transitions among different states. The generic structure is included below.

Index	Default	Base		Index	Next	Check
0				0		
1				1		
2				2		
3				3		
4				4		
				5		
				6		

Table 4 : Four Array data structure for representing a DFA

The calculation of the next state information in a 4array scheme is outlined in the following function.

```
function nextstate (s, a)
{ if CHECK[BASE[s] + a] = s
  then  NEXT[BASE[s] + a]
  else
        return (nextstate( DEFAULT[s], a))
}
```

A heuristic, which works well in practice to fill up the four arrays, is to find for a given state, the lowest BASE, so that the special entries of the state can be filled without conflicting with the existing entries.

APPENDIX 2

OPERATORS & THEIR ATTRIBUTES IN C / C++

Precedence	Associativity	Arity	Operator	Function of the operator
16	L	binary	[]	array index
15	R	unary	++, --	increment, decrement
15	R	unary	~	bitwise NOT
15	R	unary	!	logical NOT
15	R	unary	+, -	unary minus, plus
15	R	unary	*, &	dereference, address of
13	L	binary	*, /, %	multiplicative operators
12	L	binary	+, -	arithmetic operators
11	L	binary	<<, >>	bitwise shift
10	L	binary	<, <=, >, >=	relational operators
9	L	binary	==, !=	equality, inequality
8	L	binary	&	bitwise AND
7	L	binary	^	bitwise XOR
6	L	binary	|	bitwise OR
5	L	binary	&&	logical AND
4	L	binary	||	logical OR
3	L	ternary	?:	arithmetic if
2	R	binary	=, *=, /=, %=, +=, -=, <<=, >>=, &=, |=, ^=	assignment operators

End of Tutorial Sheet 2