Lexical Analysis - Scanner-Contd

66.648 Compiler Design Lecture 3(01/21/98)

Computer Science

Rensselaer Polytechnic

Lecture Outline

- More on Lexical Analyzer
- Examples and Algorithms
- Administration

Non-regular Languages

Regular Expressions can be used to denote only a fixed number or unspecified number of repetitions.

Examples of nonregular languages:
1. The set of all strings of balanced parentheses e.g.., (()), (()()(())), etc. - nested comments are also nonregular.
2. The set of all palindromes. {wv| v is the reverse of w, w is a string over the alphabet.}
3. Repeating Strings { ww| w a string over the alphabet}.

Examples of Constructing NFA from a reg. expr

A NFA for a regular expression can be constructed as follows:

1. There is a single transition labeled with an alphabet. (this includes an epsilon symbol). There are two states, the start state and the final state and one edge/transition.
2.For E1.E2, construct a new start state and a new final state. From the start state, add an edge labeled with epsilon to start state of E1. From the final state of E1, add an epsilon transition to Start state of E2.

NFA Counted.

Add a transition/edge from the final state of E2 to the constructed Final state.
3. For E1|E2, Construct new start state, new final state. Add a transition from the start state to the start states of E1 and E2. These transitions are labeled with epsilon symbol
4. For E*, Construct new start state and new final state. Add an epsilon transition from the start state to the start state of E, and epsilon transition from the final stat

NFA Contd

of E to the constructed final state. Finally add an epsilon transition from the final state of E to the start state of E.

This gives an algorithm to construct the transition graph from a regular expression.

e.g.. identifier, comments, floating constants.

Simulation of NFA

An epsilon closure of a state x is the set of states that can be reached (including itself) by making just transition labeled with epsilon.

We want to get the next token from the input stream.
Properties:
1. The longest sequence of characters starting at the current position that matches a regular exp. for a token.
2. Input buffer is repositioned to the first character following the token.
3. Nothing gets read after the end-of-file.

Algorithm page 126 of text alg.3.3

```
getNextToken() {
t.error = true; // t is a token that will be found
S = epsilon_closure({start});
while(true) {
if (S is empty} break;
if (S contains a final state) {
t.eror=false;
//fill in t.line and other attributes.}
if (end_of_file) break;
c= getchar():
T=move(S,c); S=epsilon_closure(T);}
reset_inputbuffer(t.line,t.lastcol+1);
 return t}
```

Analysis of the Alg

Simulation time = O(size of input string)
Simulation Space=O(size of NFA).

It is inefficient to read the entire program as scanner input. Th
scanner converts the characters into token on the fly.
The scanner keeps an internal buffer of bounded size to
hold the largest possible token size and largest lookahead
needed. This is usually much smaller than the entire program.

Discussion contd

Often, in practice, parser requests a scanner to provide with a token. The parser tries to construct a parse tree (by doing a shift/reduce operations) to get the parse tree.

High-level Structure of a scanner

```
repeat {
  t= getNextToken();
  if (t.error) {
  print error message;
  exit from compiler or recover from the
  error;}
  output_token(t);}
until(t.EOF)
```

Output tokens for sample program

Token	Attrib	line
tok_public		1
tok_class		1
tok_id	first	1
tok_lbrace		1
tok_public		2
tok_static		2
tok_void		2
tok_main		2
tok_lparen		2

Lex- program format

Format
%{
included as is
%}
defintions
%%
patterns   actions
%%
program

Sample lex program

```
%{
char reserved_word[12][20];
%}
%%
[a-z]+ { if (lookup(yytext)==-1) {
   printf("tok_id\t%s\t%d\n",yytext,yylineno);
} else {printf("tok_%s\t\t%d\n",
        reseved_word[I],yylineno);}
[0-9]+ { printf("tok_intconst\t%s\t%d\n",
   yytext,yylineno); }
```

Program Contd

```
"="   printf("tok_eq\t\t%d\n",yylineno);
";"   printf("tok_semi\t\t%d\n",yylineno);
"("   printf("tok_lparen\t\t%d\n",yylineno);
")"   printf("tok_rparen\t\t%d\n",yylineno);
"{"   printf("tok_lbrace\t\t%d\n",yylineno);
"}"   printf("tok_rbrace\t\t%d\n",yylineno);
"["   printf("tok_lsqb\t\t%d\n",yylineno);
"]"   printf("tok_rsqb\t\t%d\n",yylineno);
%%
```

Administration

- We are in Chapter 3 of Aho, Sethi and Ullman's book. Please read that chapter and chapter 1 which we covered in Lectures1 and 2.
- Work out the first few exercises of chpater 3.
- Lex and Yacc Manuals are handed out. Please read them.

First Project is in the web.

It consists of three parts.

1) To write a lex program

2) To write a YACC program.

3) To write five sample Java programs. They can be either applets or application programs

Comments and Feedback

- Please let me know if you have not found a project partner.
- A sample Java compiler is in the class home page.