CS480
Translators

Finishing Lex Analysis
Chap. 3
# NFA to DFA

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$-closure($s$)</td>
<td>Set of NFA states reachable from NFA state $s$ on $\epsilon$-transitions alone.</td>
</tr>
<tr>
<td>$\epsilon$-closure($T$)</td>
<td>Set of NFA states reachable from some NFA state $s$ in set $T$ on $\epsilon$-transitions alone; $= \bigcup_{s \text{ in } T} \epsilon$-closure($s$).</td>
</tr>
<tr>
<td>move($T$, $a$)</td>
<td>Set of NFA states to which there is a transition on input symbol $a$ from some state $s$ in $T$.</td>
</tr>
</tbody>
</table>

```python
while ( there is an unmarked state $T$ in $Dstates$ ) {
    mark $T$;
    for ( each input symbol $a$ ) {
        $U = \epsilon$-closure(move($T$, $a$));
        if ( $U$ is not in $Dstates$ )
            add $U$ as an unmarked state to $Dstates$;
        $Dtran[T, a] = U$;
    }
}
```
Figure 3.34: NFA $N$ for $(a|b)^*abb$

<table>
<thead>
<tr>
<th>NFA State</th>
<th>DFA State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${0, 1, 2, 4, 7}$</td>
<td>$A$</td>
<td>$B$</td>
<td>$C$</td>
</tr>
<tr>
<td>${1, 2, 3, 4, 6, 7, 8}$</td>
<td>$B$</td>
<td>$B$</td>
<td>$D$</td>
</tr>
<tr>
<td>${1, 2, 4, 5, 6, 7}$</td>
<td>$C$</td>
<td>$B$</td>
<td>$C$</td>
</tr>
<tr>
<td>${1, 2, 4, 5, 6, 7, 9}$</td>
<td>$D$</td>
<td>$B$</td>
<td>$E$</td>
</tr>
<tr>
<td>${1, 2, 4, 5, 6, 7, 10}$</td>
<td>$E$</td>
<td>$B$</td>
<td>$C$</td>
</tr>
</tbody>
</table>
DFA - \((a \mid b)^*abb\)

Figure 3.36: Result of applying the subset construction to Fig. 3.34
NFA w/ $\varepsilon$-transitions

- $\text{(aa* | bb*)}$
Lexical-Analyzer Generator

Figure 3.22: Creating a lexical analyzer with Lex
What does Lex do?

• A program that takes regular expressions and action pairs, and builds a recognizer.
• Why actions?
Using Lex

1. LEX makes NO assumptions about tokens, must recognize ALL of input, including spaces, comments even illegal symbols ('.' default action).
2. Multiple rules can match the same input. Ambiguity is resolved using the following two rules:
   - A) choose pattern which matches the longest input string
   - B) if two patterns match the same size string, choose that which was listed first.
{%
    /* definitions of manifest constants
     LT, LE, EQ, NE, GT, GE,
     IF, THEN, ELSE, ID, NUMBER, RELOP */
%

    /* regular definitions */
    delim     [ \t\n]
    ws        {delim}+
    letter    [A-Za-z]
    digit     [0-9]
    id        {letter}({letter}|{digit})*
    number    {digit}+(/.*{digit}+)?(E[+-]??{digit}+)?
%
%
    {ws}   /* no action and no return */
    if     {return(IF);}
    then   {return(THEN);}
    else   {return(ELSE);}
    {id}   {yyval = (int) installID(); return(ID);}
    {number}   {yyval = (int) installNum(); return(NUMBER);}
    "<"     {yyval = LT; return(RELOP);}
    "\<"    {yyval = LE; return(RELOP);}
    ";"     {yyval = EQ; return(RELOP);}
    ";"     {yyval = NE; return(RELOP);}
    ">"     {yyval = GT; return(RELOP);}
    ">="    {yyval = GE; return(RELOP);}
%}
int installID() { /* function to install the lexeme, whose
first character is pointed to by yytext,
and whose length is yyleng, into the
symbol table and return a pointer
thereeto */
}

int installNum() { /* similar to installID, but puts numer-
cical constants into a separate table */
}

Figure 3.23: Lex program for the tokens of Fig. 3.12
```c
#define DIGIT 2
#define NUM 3

ws [ \t\n]+
digit [0-9]
num \{digit\}+

{ws} { } // ignore whitespace
{digit} { return(DIGIT); } // match on digit
{num} { return(NUM); } // match integer numbers
{ yyerror(); } // error on anything else

int main(void)
{
    int n;
    while ( n = yylex() ) // call scanner until it returns 0 for EOF
        printf ("Code:%d Text:%s TextLength:%d\n", n, yytext, yyleng);
    return 0;
}

int yyerror(void) // default action in case of error in yylex()
{
    printf(" error\n");
    exit(0);
}

int yywrap(void)
{
    return 1; // won't compile on Linux w/o it
} 31L, 682C written
```
flip1 ~/480test 184% lex lex.1
flip1 ~/480test 185% gcc lex.yy.c -o scanner
flip1 ~/480test 186% cat input
56 7 8
flip1 ~/480test 187% ./scanner < input
Code: 3 Text: 56 TextLength: 2
Code: 2 Text: 7 TextLength: 1
Code: 2 Text: 8 TextLength: 1
flip1 ~/480test 188% vi input
flip1 ~/480test 189% cat input
56 7 8 if
flip1 ~/480test 190% ./scanner < input
Code: 3 Text: 56 TextLength: 2
Code: 2 Text: 7 TextLength: 1
Code: 2 Text: 8 TextLength: 1
error
flip1 ~/480test 191%
```c
#define DIGIT 2
#define NUM 3

ws [ \t\n]+
digit [0-9]+
num {digit}+

{ws} { } // ignore whitespace
{digit} { return(DIGIT); } // match on digit
{num} { return(NUM); } // match integer numbers
.

%%

int main(void)
{
    int n;
    while ( n = yylex() ) // call scanner until it returns 0 for EOF
        // token code, lexeme string, length
        printf ("Code:%d Text:%s TextLength:%d\n", n, yytext, yyleng);
    return 0;
}

%*/

int yyerror(void) // default action in case of error in yylex()
{
    printf(" error\n");
    exit(0);
}

int yylex(void) // won't compile on Linux w/o it
{
    return 1;
}
```
```c
#include <stdio.h>

int yylex ();   // scanner prototype
extern char* yytext;
extern int yyleng;

int main(void) {
    int n;
    while ( n = yylex() )   // call scanner until it returns 0 for EOF
        // output the token code, lexeme string, length
        printf ("Code:%d  Lexeme:%s  Length:%d\n", n, yytext, yyleng);
    return 0;
}
```
flip1 ~/480test 207% lex lex.1
flip1 ~/480test 208% gcc test.c lex.yy.c -o scanner
flip1 ~/480test 209% ./scanner < input
Code:3  Lexeme:56  Length:2  
Code:2  Lexeme:7   Length:1  
Code:2  Lexeme:8   Length:1  
error
flip1 ~/480test 210% cat input
56 7 8 if
flip1 ~/480test 211%  

```c
#define DIGIT 2
#define NUM 3

ws [ \t\n]+
digit [0-9]
num {digit}+

{ws} {}  // ignore whitespace
{num} { return(NUM); }  // match integer numbers
{digit} { return(DIGIT); }  // match on digit
.
{ yyerror(); }  // error on anything else

/*
int main(void)
{
  int n;
  while ( n = yylex() )  // call scanner until it returns 0 for EOF
    // token code, lexeme string, length
    printf ("Code:%d Text:%s TextLength:%d\n", n, yytext, yyleng);

  return 0;
}
*/

int yyerror(void)  // default action in case of error in yylex()
{
  printf(" error\n");
  exit(0);
}

int yywarp(void)
{
  return 1;  // won't compile on Linux w/o it
} 8,0-1 All
```
```
flip2 ~/480test 164% cat input
56 7 8 if
flip2 ~/480test 165% lex lex.1
lex.1:12: warning, rule cannot be matched
flip2 ~/480test 166% gcc test.c lex.yy.c -o scanner
flip2 ~/480test 167% ./scanner < input
Code: 3  Lexeme: 56  Length: 2
Code: 3  Lexeme: 7  Length: 1
Code: 3  Lexeme: 8  Length: 1
error
flip2 ~/480test 168%
```
```c
1  
2  #define DIGIT 2
3  #define NUM 3
4  void *yyval;
5  
6  ws    [ \t\n]+ 
7  digit  [0-9] 
8  num    {digit}+
9  
10  
11  {ws}   { }                  // ignore whitespace
12  {num}  { yyval=yytext; return(NUM); } // match integer numbers
13  {digit} { return(DIGIT); } // match on digit
14  .      { yyerror(); } // error on anything else
15  
16  /*
17    */
18  int main(void)
19  { int n;
20      while ( n = yylex() ) // call scanner until it returns 0 for EOF
21          // token code, lexeme string, length
22          printf ("Code:%d Text:%s TextLength:%d\n", n, yytext, yyleng);
23      return 0;
24  }
25  */
26  
27  int yyerror(void) // default action in case of error in yylex()
28  { printf(" error\n");
29      exit(0);
30  }
31  
32  int yywrap(void)
33  { return 1; } // won't compile on Linux w/o it
```

#include <stdio.h>

int yylex();    // scanner prototype
extern int yyleng;
extern void *yyval;

int main(void)
{
    int n;
    while (n = yylex())    // call scanner until it returns 0 for EOF
    // output the token code, lexeme string, length
    printf ("Code:%d  Lexeme:%s  Length:%d\n", n, yyval, yyleng);
    return 0;
}
flip2 ~/480test 174% lex lex.1
lex.1:13: warning, rule cannot be matched
flip2 ~/480test 175% gcc test.c lex.yy.c -o scanner
flip2 ~/480test 176% ./scanner < input

Code: 3  Lexeme: 56  Length: 2
Code: 3  Lexeme:  7  Length: 1
Code: 3  Lexeme:  8  Length: 1
  error
flip2 ~/480test 177%  

Quiz #4

- Convert the NFA to DFA

- Write a finite state automata that will recognize any string consisting of a and b characters where the number of a's is even (or zero) and the number of b's is odd.