Lexical Analysis (Scanning)

Lecture Four

Lexical Analyzer (Part 4)

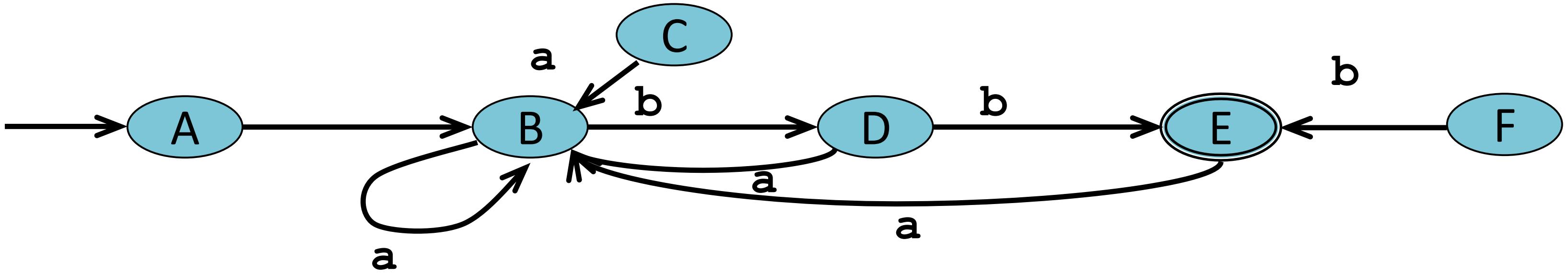
- Minimizing DFA (Hopcroft’s Algorithm)
- Scanner Implementation
  1. Code by hand
     a. Using Doubly nested case
     b. Using transition table
  2. Lex Scanner Generator

Minimizing DFA
Hopcroft’s Algorithm

Minimizing DFA

1. Remove unreachable states

(unreachable from start state using any input symbol)



Unreachable States → C,F

Minimizing DFA(Continued)

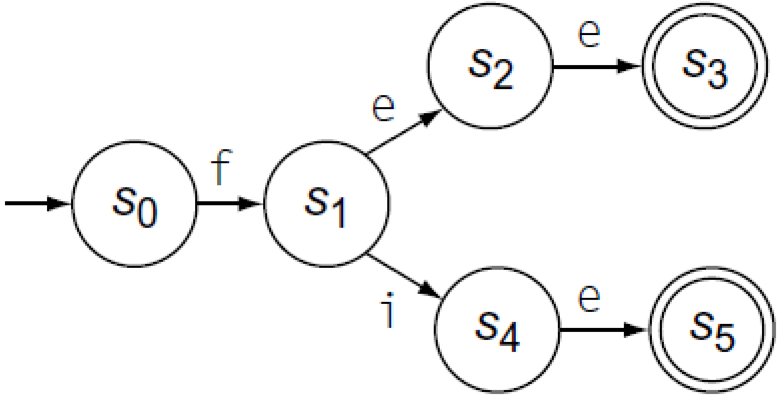
2. Merge Equivalent States

Two DFA states, *Si*,*Sj* are equivalent, have the same behavior in response to all input characters(go to the same partition).

- partition states into 2 partition
  - Final states
  - Non final states
- Explore each partition on every input. Split non-equivalent states.
- Repeat until there is no possible partition.

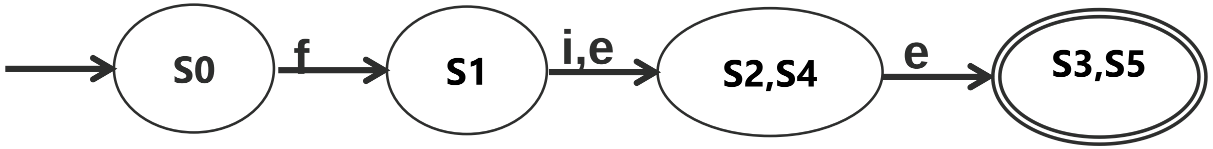
Minimizing DFA(Continued)

DFA State	e	i	f
s0	-	-	s1
s1	s2	s4	-
s2	s3	-	-
*s3	-	-	-
s4	s5	-	-
*s5	-	-	-



(a) DFA for “*fee* | *fie*”

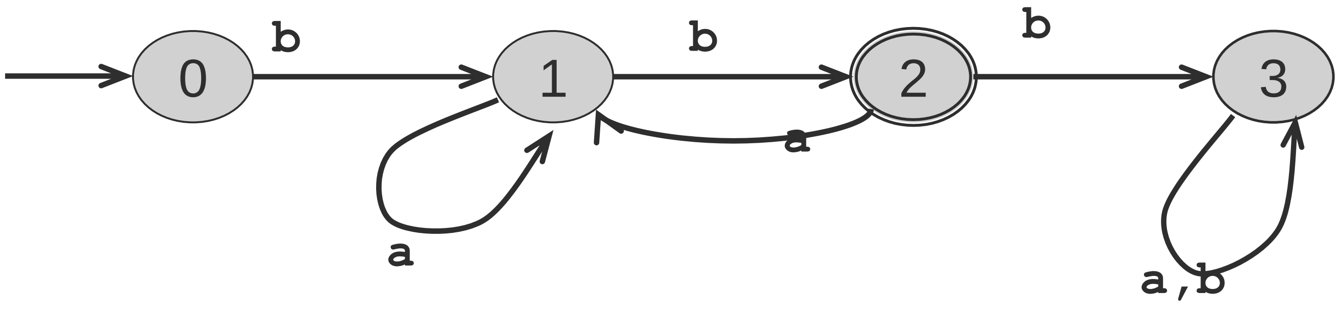
Partitions	Set	input	action
{s0,s1,s2,s4}{s3,s5}	{s3,s5}	all	none
{s0,s1,s2,s4}{s3,s5}	{s0,s1,s2,s4}	e	split {s2, s4}
{s2,s4}{s0,s1}{s3,s5}	{s0,s1}	e	Split s1
{s0}{s1} {s2,s4}{s3,s5}	all	all	none



Minimizing DFA(Continued)

3- Remove dead states

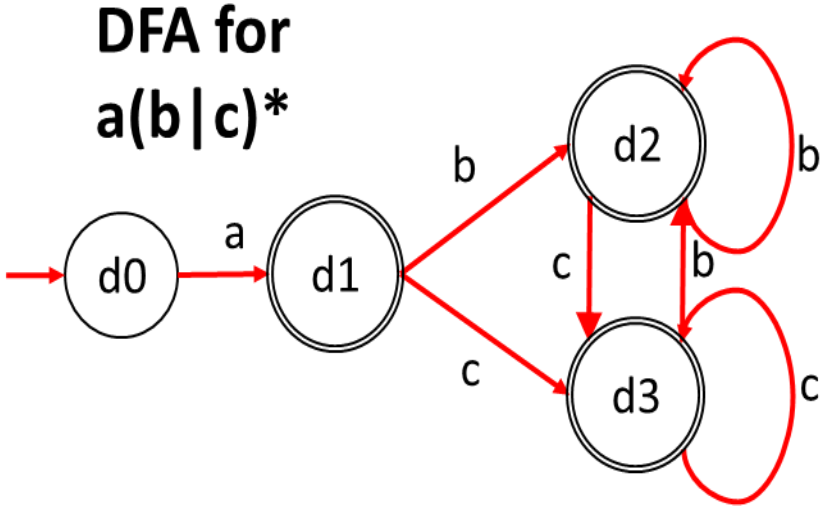
(non final states whose transition on every input terminate on itself.



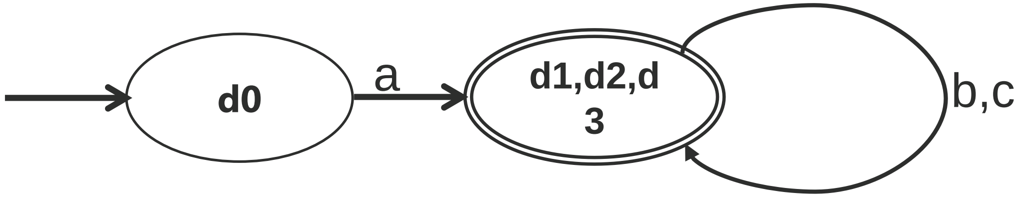
Dead State → 3

Minimizing DFA(Continued)

DFA State	a	b	c
d0	d1	-	-
*d1	-	d2	d3
*d2	-	d2	d3
*d3	-	d2	d3



Partitions	Set	input	action
{d0}{d1,d2,d3}	{d1,d2,d3}	all	none



Scanner Implementation

Implementation of DFA

- There are several ways to translate a DFA into code

  - Using doubly nested case analysis

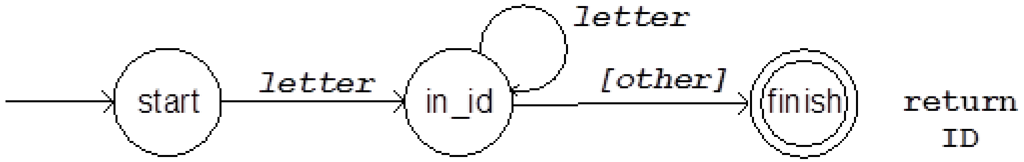
    Use a variable to maintain the current state and write the transitions as a doubly nested case statement inside a loop, where the first case statement tests the current state and the nested one tests the input character, given the state.

  - Using the transition table (Table driven)

    Express the DFA as a data structure and then write a "generic" code that will take its action from the data structure. A simple data structure that is adequate for this purpose is a two-dimensional array, indexed by state and input character, that expresses the values of the transition function

Implementation of DFA (Continued)

- Using doubly nested case analysis



```
state = start;
getchar(input);
while (state != finish && state != error)
  switch (state) {
    case start: if (isalpha(input)) {
                       advance(input); state = in_id;}
                       else state = error; break;
    case in_id: if (!isalpha(input))
                       state = finish;
                       else advance(input); break;
    default: break;
  }
if (state == finish) return ID;
   else return ERROR;
```

Implementation of DFA (Continued)

- Using the transition table (Table driven)

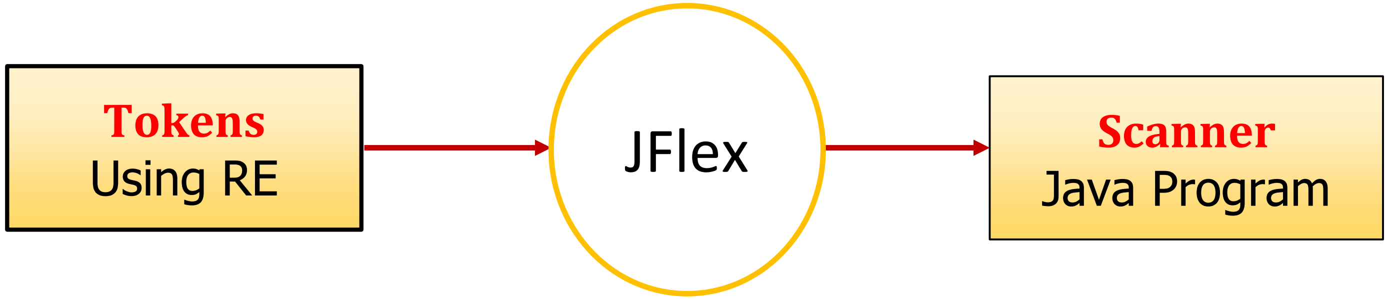
```
i=0;
state =0;
while (input[i])
{
        state=DFA[state,input[i++]];
}
```

Lex
Scanner Generator

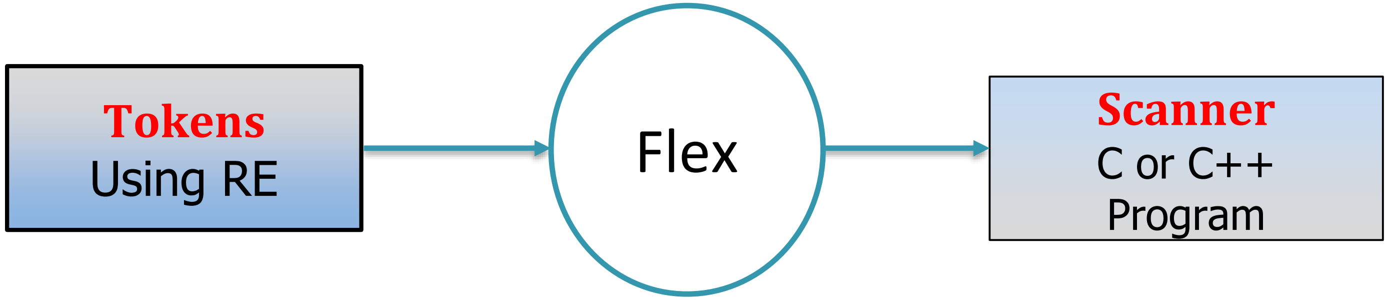
Scanner Implementation(2)

- using Scanner Generator

- (1)

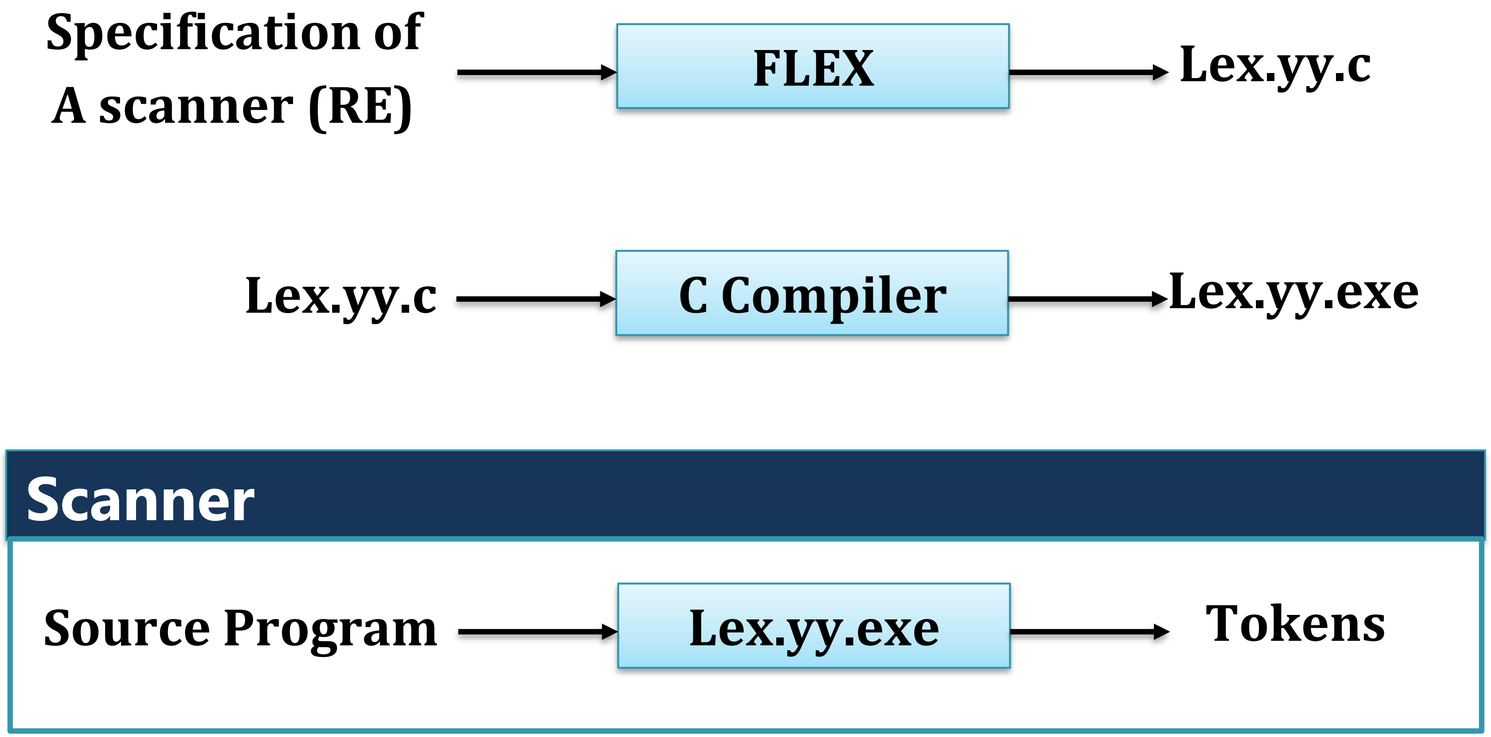


- (2)



Flex

- The most popular version of LEX is called FLEX (Fast LEX)
- Steps



FLEX input file

Definitions (optional)

%%

Rules

%%

Functions

FLEX input file

- Definitions
  - Any code external to any function should be enclosed in the delimiters %{ and %}
  - Names for regular expressions
- Rules
  - This section contains a sequence of regular expressions followed by the C code that is to be executed when the corresponding regular expression is matched
- Functions
  - This section contains the C code for functions used.

FLEX program(1)

```
%{
// a Lex program that adds line numbers
// to lines of stdin, printing to stdout
#include <stdio.h>
int lineno = 1;
%}
line .*\n
%%
{line} { printf("%5d %s",lineno++,yytext); }
%%
int main()
{ yylex(); return 0; }
```

FLEX program(2)

```
%{// Selects only lines that end
  // or begin with the letter 'a'
#include <stdio.h>
%}
ends_with_a .*a\n
begins_with_a a.*\n
%%
{ends_with_a} ECHO;
{begins_with_a} ECHO;
.*\n ;
%%
int main()
{ yylex(); return 0; }
```

FLEX internal names

Lex internal name	Meaning/Use
`lex.yy.c` or `lexyy.c`	Lex output file name
`yylex`	Lex scanning routine
`yytext`	string matched on current action
`yyleng`	length of `yytext`
`yyin`	Lex input file (default: `stdin`)
`yyout`	Lex output file (default: `stdout`)
`input`	Lex buffered input routine
`ECHO`	Lex default action (print `yytext` to `yyout`)