

Lecture 8

Lexical Analysis

Slides by David Albrecht (2011), minor modifications by Graham Farr (2013, 2018).

FIT2014 Theory of Computation

Overview

- Lexical Analyzer
- Tokens and lexemes
- Implementing Lexical Analyzer
- Other Algorithms

Simple Calculator Example

```
-2.45 + 3.98 * 0.456\n
```

Lexemes:

```
-2.45 + 3.98 * 0.456 \n
```

Tokens:

- Numbers, spaces, operators, newlines

Java

```
class static Hello { \n public\nvoid main ( String\nargs [ ] ) {\nSystem .\nout . println ( “Hello” ) ; }\n}
```

class		Hello	{	\n	public			
static		void		main	(String		
		args	[])	{	System	.
out	.	println	(“Hello”)	;	}	}

Tokens:

- reserved keywords, identifiers, brackets, etc.



Terminology

- A **token** is a name of a pattern.
 - It may also have an attribute value associated with it.
- A **lexeme** is a sequence of characters that matches the pattern corresponding to a token.
- A **pattern** is a description of the form that the lexemes of a token may take.
 - Often described using regular expressions.

Lexical Analyzer

- Reads the input one character at a time.
- Splits the input up into tokens.
- Implemented using a **Finite Automaton** or **NFA**.

EVEN-EVEN

Write a program which reads in a character string, consisting of **a**'s and **b**'s, one character at a time and identifies whether or not the string belongs to **EVEN-EVEN**.

```
class EvenEven {
    public static void main(String args[]) throws java.io.IOException {
        int countA = 0;
        int countB = 0;
        int peek = System.in.read();

        while (peek != -1) {
            switch(peek) {
                case 'a':
                    countA++;
                    break;
                case 'b':
                    countB++;
                    break;
                default:
                    break;
            }
            peek = System.in.read();
        }
        if (countA % 2 == 0 && countB % 2 == 0)
            System.out.println("Match");
        else
            System.out.println("No Match");
    }
}
```

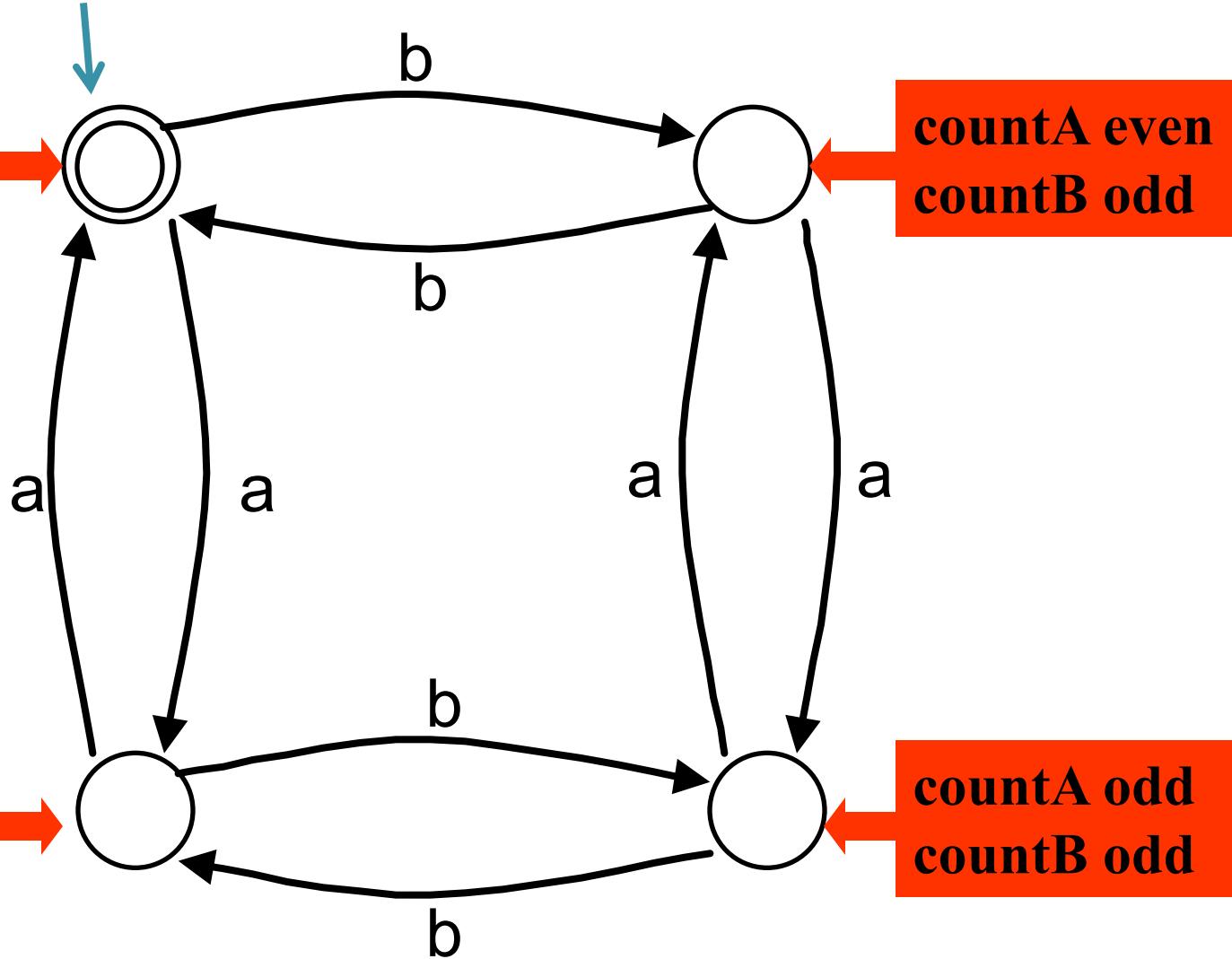
EVEN-EVEN

countA even
countB even

countA even
countB odd

countA odd
countB even

countA odd
countB odd

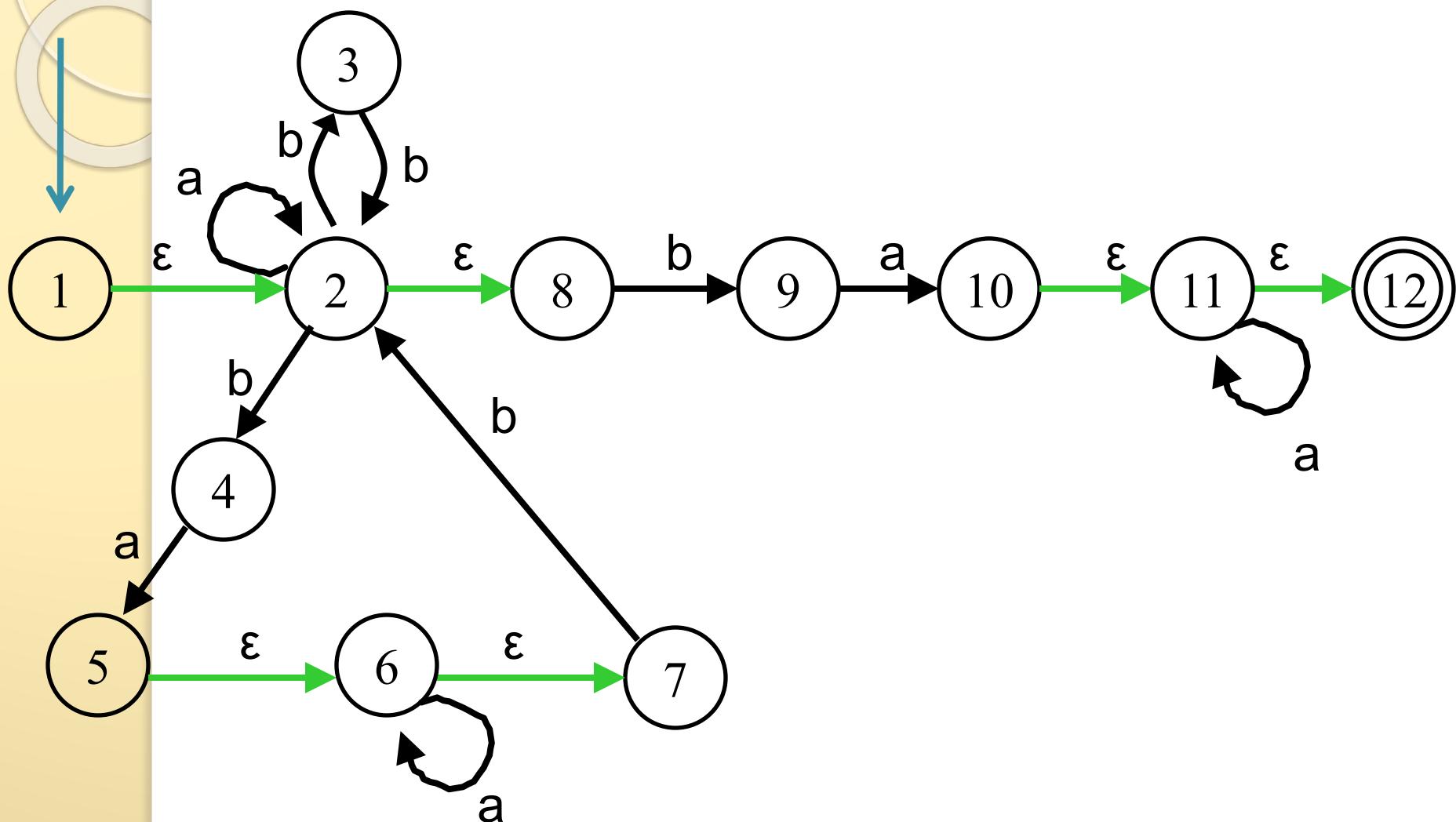


Matching a Regular Expression

Write a program which reads in a character string, consisting of **a**'s and **b**'s, one character at a time and identifies whether or not the string matches the following regular expression.

$$(a \cup bb \cup baa^*b)^*baa^*$$

NFA



DFA

	a	b
Start $\{1,2,8\}$	$\{2,8\}$	$\{3,4,9\}$
$\{2,8\}$	$\{2,8\}$	$\{3,4,9\}$
$\{3,4,9\}$	$\{5,6,7,10,11,12\}$	$\{2,8\}$
Final $\{5,6,7,10,11,12\}$	$\{6,7,11,12\}$	$\{2,8\}$
Final $\{6,7,11,12\}$	$\{6,7,11,12\}$	$\{2,8\}$

Can we simplify this DFA?

DFA

A **Final State** and a **non-Final State** are
fundamentally different.

They *cannot be combined.*

So: give all **Final States** one colour,
and all **non-Final States** a *different* colour.

Different colours \Rightarrow
different states; they cannot be combined.

Same colours $\not\Rightarrow$ *same states;*
the states may or may not be combined.
We have not yet ruled out combining them.

DFA

	a	b
Start {1,2,8}	{2,8}	{3,4,9}
{2,8}	{2,8}	{3,4,9}
{3,4,9}	{5,6,7,10,11,12}	{2,8}
Final {5,6,7,10,11,12}	{6,7,11,12}	{2,8}
Final {6,7,11,12}	{6,7,11,12}	{2,8}

DFA

	a	b
Start {1,2,8}	{2,8}	{3,4,9}
{2,8}	{2,8}	{3,4,9}
{3,4,9}	{5,6,7,10,11,12}	{2,8}
Final {5,6,7,10,11,12}	{6,7,11,12}	{2,8}
Final {6,7,11,12}	{6,7,11,12}	{2,8}

DFA

Different colour pattern along row \Rightarrow different states; they cannot be combined.

The different colour patterns mean they *must* treat some strings differently.

So, give those states different colours.

Same colour patterns $\not\Rightarrow$ same states;
the states *may or may not* be combined.

We have not yet ruled out combining them.

DFA

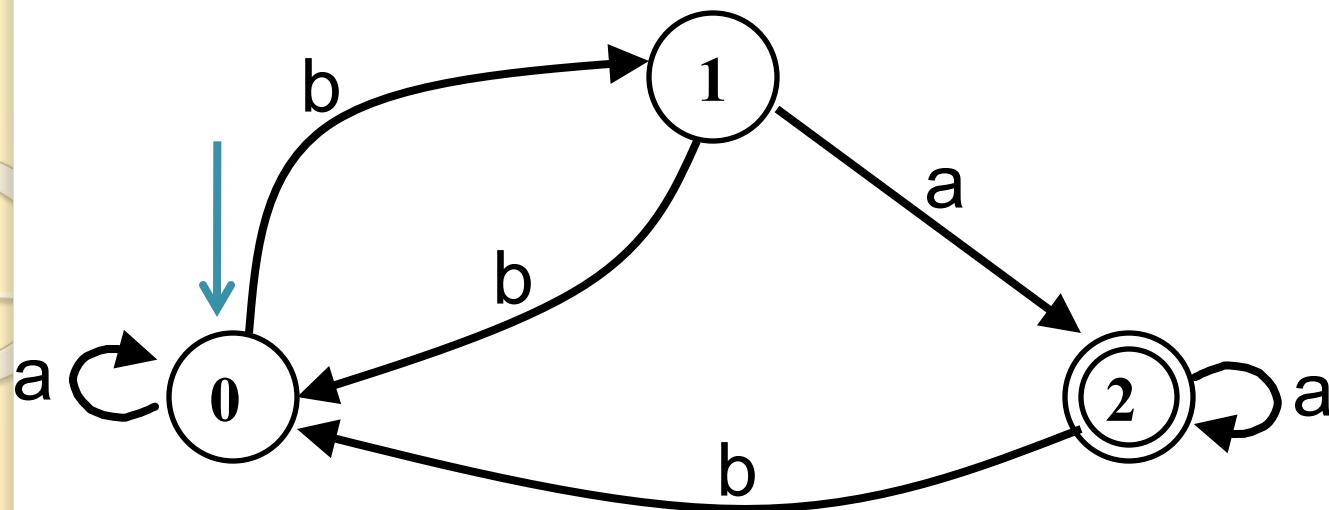
	a	b
Start {1,2,8}	{2,8}	{3,4,9}
{2,8}	{2,8}	{3,4,9}
{3,4,9}	{5,6,7,10,11,12}	{2,8}
Final {5,6,7,10,11,12}	{6,7,11,12}	{2,8}
Final {6,7,11,12}	{6,7,11,12}	{2,8}

DFA

	a	b
Start {1,2,8}	{2,8}	{3,4,9}
{2,8}	{2,8}	{3,4,9}
{3,4,9}	{5,6,7,10,11,12}	{2,8}
Final {5,6,7,10,11,12}	{6,7,11,12}	{2,8}
Final {6,7,11,12}	{6,7,11,12}	{2,8}

Minimum DFA

- Colour all **Final States** with one colour, and colour all **Non-Final States** with a different colour.
- Repeat until no new colour is added:
 - For each colour:
 - Consider all states with that colour.
 - If their rows in the transition table do not have the same pattern of colours, then
 - The states with different colour patterns along their rows must get different colours. So ...
 - Give each different row pattern a different colour.
 - Each set of states having the same row pattern gets the colour for that row pattern.
- Give each colour a unique number, and use these numbers to form the transition table.



	a	b
Start 0	0	1
1	2	0
Final 2	2	0

```
class Match {  
    public static void main(String args[]) throws java.io.IOException {  
        int currentState = 0;  
        int[][] table = {{0, 1}, {2, 0}, {2, 0}};  
        int peek = System.in.read();  
  
        while (peek != -1) {  
            switch(peek) {  
                case 'a':  
                    currentState = table[currentState][0];  
                    break;  
                case 'b':  
                    currentState = table[currentState][1];  
                    break;  
                default:  
                    break;  
            }  
            peek = System.in.read();  
        }  
        if (currentState == 2)  
            System.out.println("Match");  
        else  
            System.out.println("No Match");  
    }  
}
```

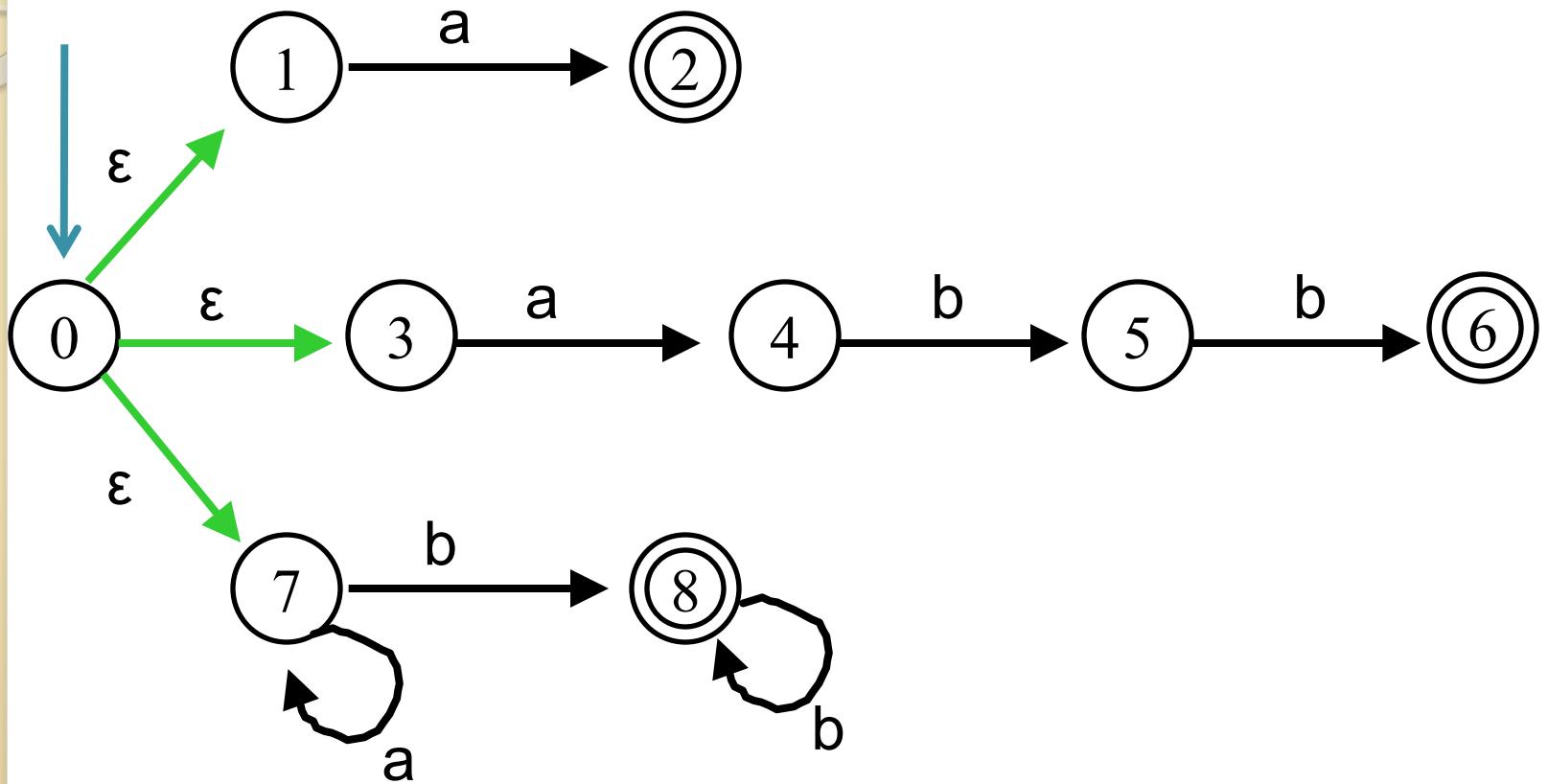
Matching Regular Expressions

Write a program which reads in a character string, consisting of **a**'s and **b**'s, one character at a time and identifies whether or not the string matches one the following regular expressions, and which one.

a, abb, a^{*}b⁺

Conventions

- Often it is possible to split a sequence of characters up into tokens in more than one way.
 - Consider **abbbb**
 - **Convention:** Match the largest possible lexeme at each stage.
- Often a sequence of characters can match more than one token.
 - Consider **abb**
 - **Convention:** If the lexemes are the same length choose the first token that is listed.

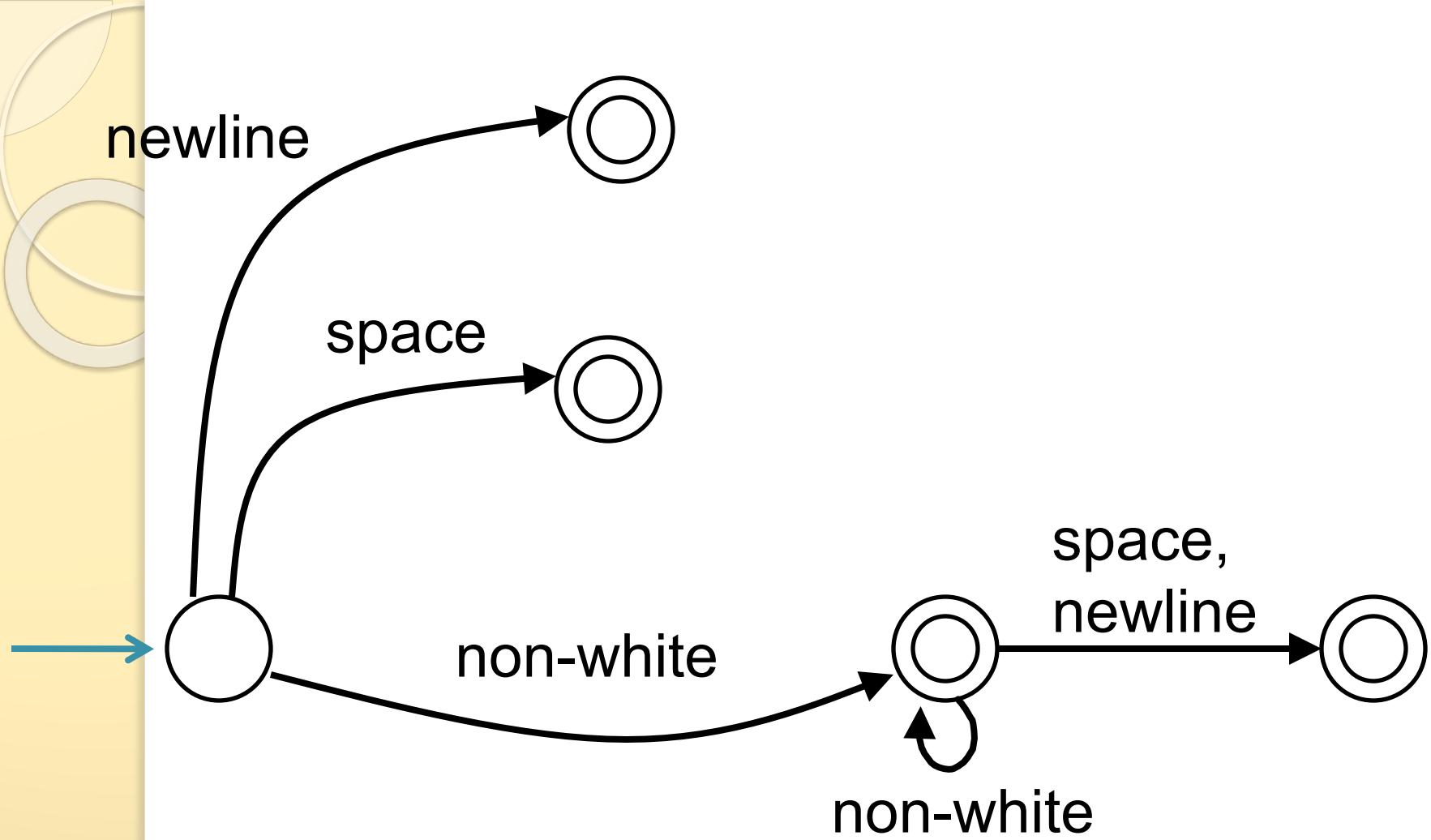


	a	b
Start $\{0,1,3,7\}$	$\{2,4,7\}$	$\{8\}$
Final a $\{2,4,7\}$	$\{7\}$	$\{5,8\}$
Final a^*b^+ $\{8\}$	f	$\{8\}$
$\{7\}$	$\{7\}$	$\{8\}$
Final a^*b^+ $\{5,8\}$	f	$\{6,8\}$
Final abb $\{6,8\}$	f	$\{8\}$

Word identification

Write a program which reads in one character at a time and identifies the following tokens:

- **newline**,
- **space**, and
- **word**.



Other Algorithms

- There are algorithms that can take a regular expression and produce a minimum state DFA without constructing a NFA.
- There are algorithms that produce fast and more compact representations of a DFA transition table than the straightforward two-dimensional table.

Revision

- Understand what a lexical analyzer does.
- Know how to find the DFA with the minimum number of states
- Know how to implement a finite automaton.