FSA
&
Scanning

COMP360

“Captain, we’re being scanned.”

Spock

FSA Assignment

- The second assignment in COMP360 has been posted to Blackboard
- You have to write some Regular Expressions and draw some FSA
- Due Monday, February 3, 2020 at 2:00pm
- You can hand in paper at the beginning of class or submit your answers to Blackboard in any readable format

Assembler Assignment

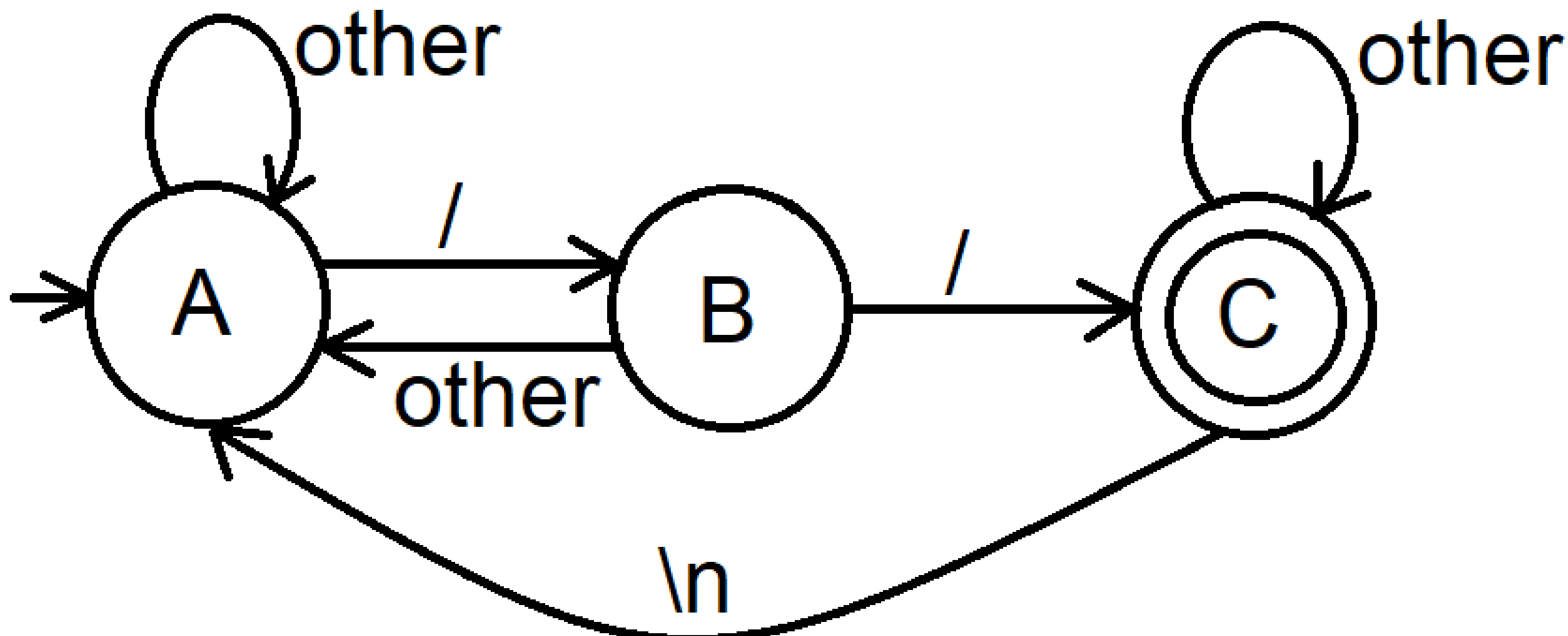
- The assembler program due today
- Upload the source code of your assembler to Blackboard before midnight

Create a FSA

- Draw a Finite State Automata that recognizes a comment in the // anything until the end of line
- Use \n to represent an end of line character
- Use “other” to represent other uninteresting characters

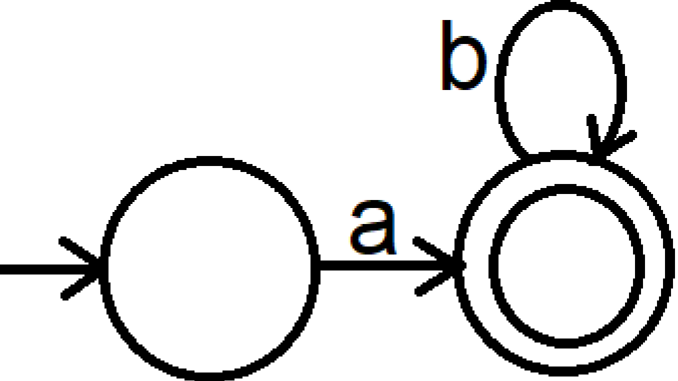
Possible Solution

- Draw a Finite State Automata that recognizes a comment in the // anything until the end of line format
- Use \n to represent an end of line character
- Use “other” to represent other uninteresting characters

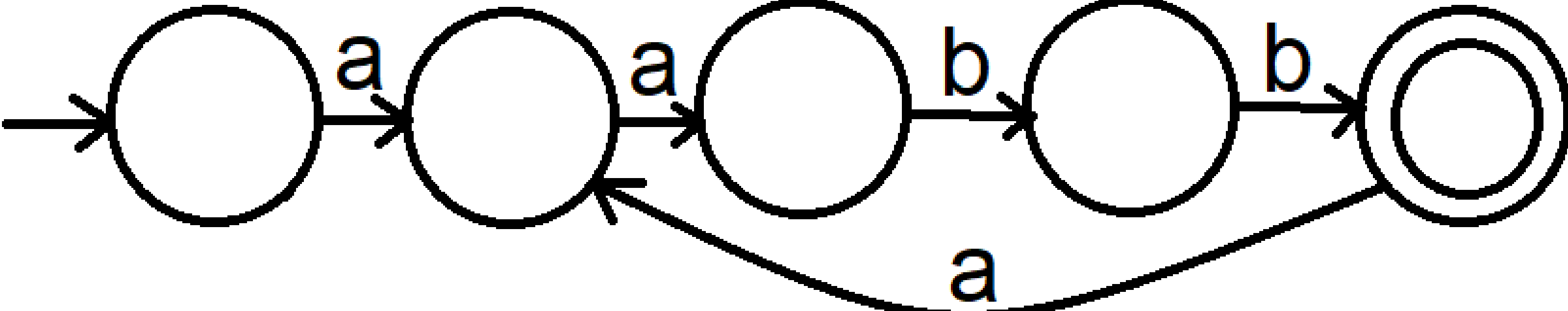


Merging FSAs

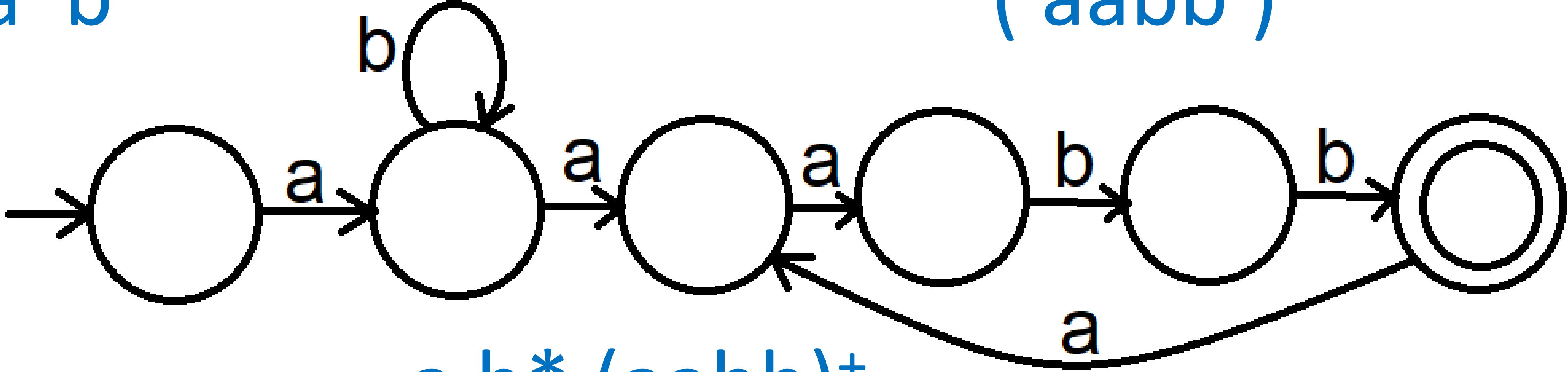
- Merging sequentially



a b*

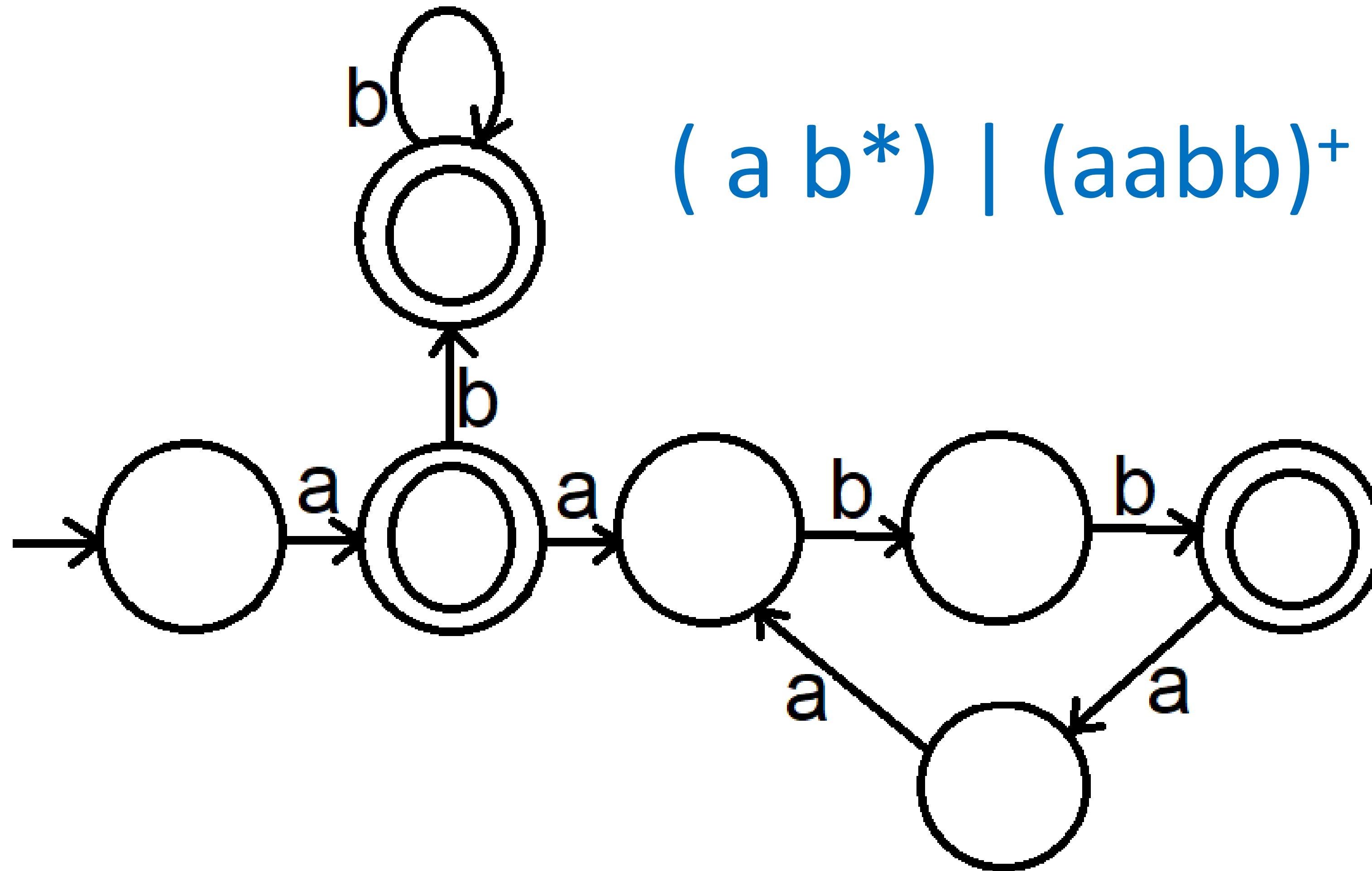


(aabb)$^+$



a b* (aabb)$^+$

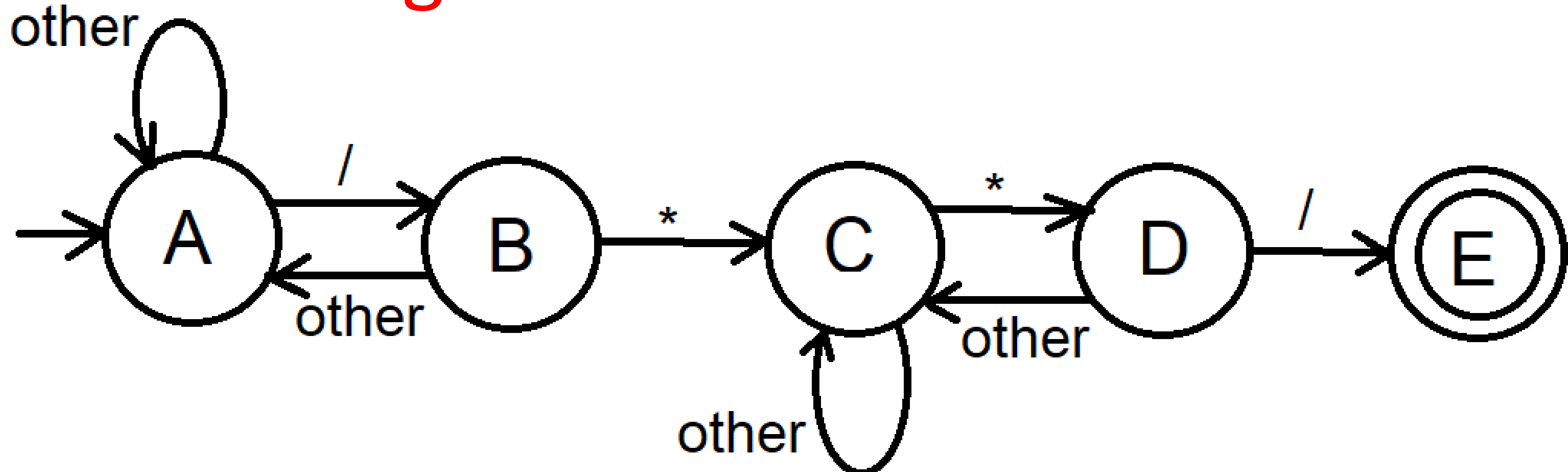
Merging as Alternates

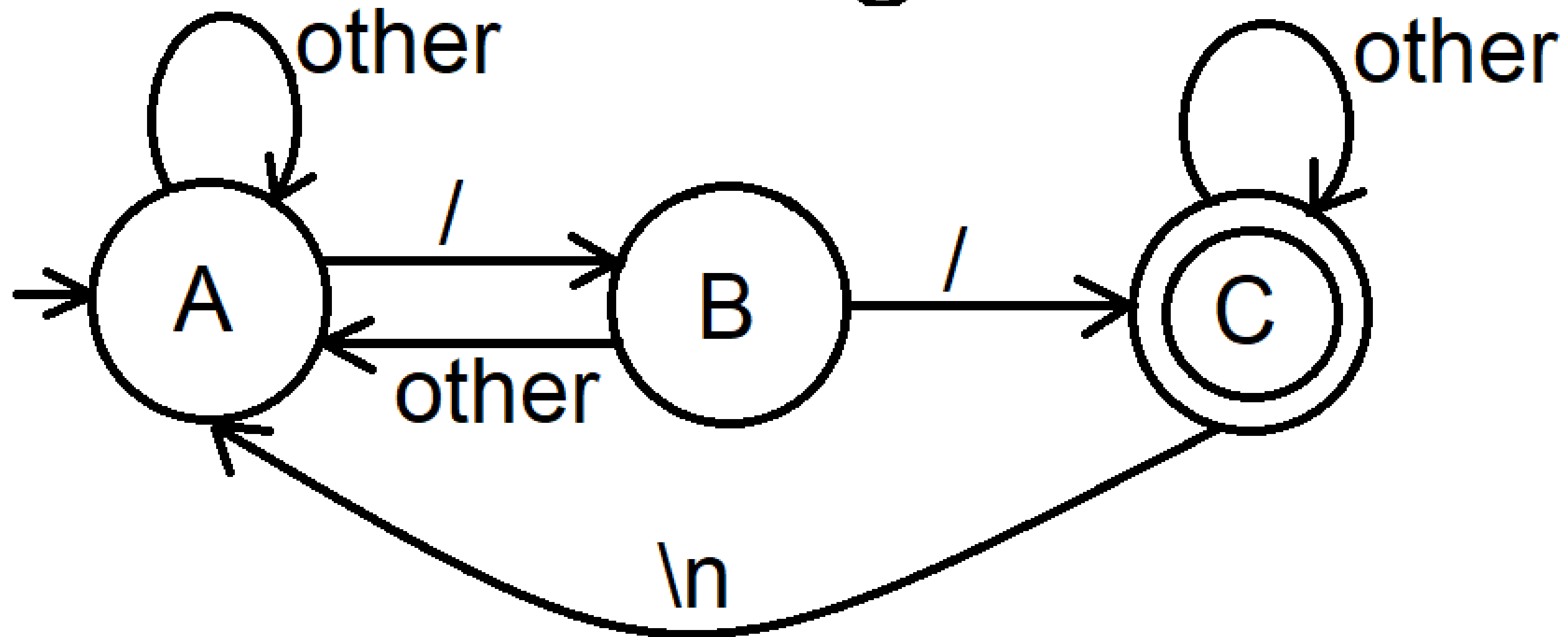


Hints for Creating a DFA

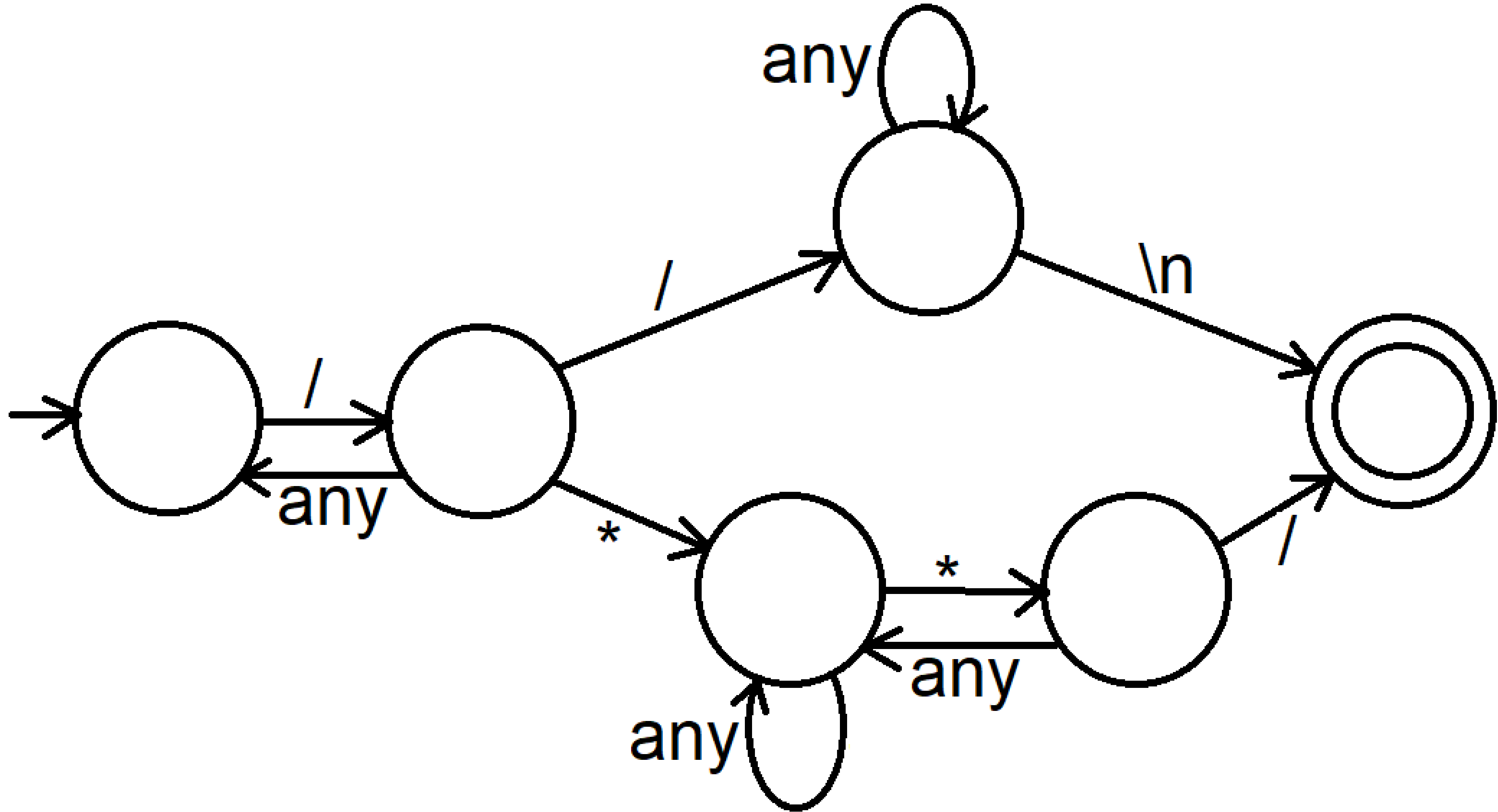
- It usually helps to think of the input one character at a time
- First consider the characters that successfully find the item you are looking for
- You can make a DFA for a subset of the language and then merge the results

Merge the two comment FSA





Possible Solution



Stages of a Compiler

- Source preprocessing
- Lexical Analysis (scanning)
- Syntactic Analysis (parsing)
- Semantic Analysis
- Optimization
- Code Generation
- Link to libraries

Compiling with a FSA

- The first step of a compiler is to parse the input source
  - discard comments
  - remove white space
  - create a list of tokens
- A token is a unit of the language
- Keywords, numbers, strings and variable names become tokens instead of just individual letters
  - “while” is a token and not “w”, “h”, “i”, “l”, “e”

Theoretical Machines

- A Finite State Automata (FSA) can be used to scan a programming language and create a list of tokens
- A Push Down Automata can be used to parse the language
- The input to the Push Down Automata is a list of tokens created by the FSA

Stages of an Interpreter

- Some languages (e.g. JavaScript and Python) are not compiled into executables
- The source code is executed directly
    - Lexical Analysis (scanning)
    - Syntactic Analysis (parsing)
    - Semantic Analysis
    - Intermediate Execution

Parsing with a FSA

- While most programming languages are context-free, many command line statements can be defined as a regular language
- If you write a program that accepts text commands, you can usually parse them with a FSA
- Assembler syntax can usually be recognized with a FSA

Long Range Plan for COMP360

- One of the goals for COMP360 is to create the first parts of a compiler
- We will create a compiler for a silly language I dreamed up
- You will need to create an FSA for our compiler's scanner
- You will need to parse the language

Implementing a FSA

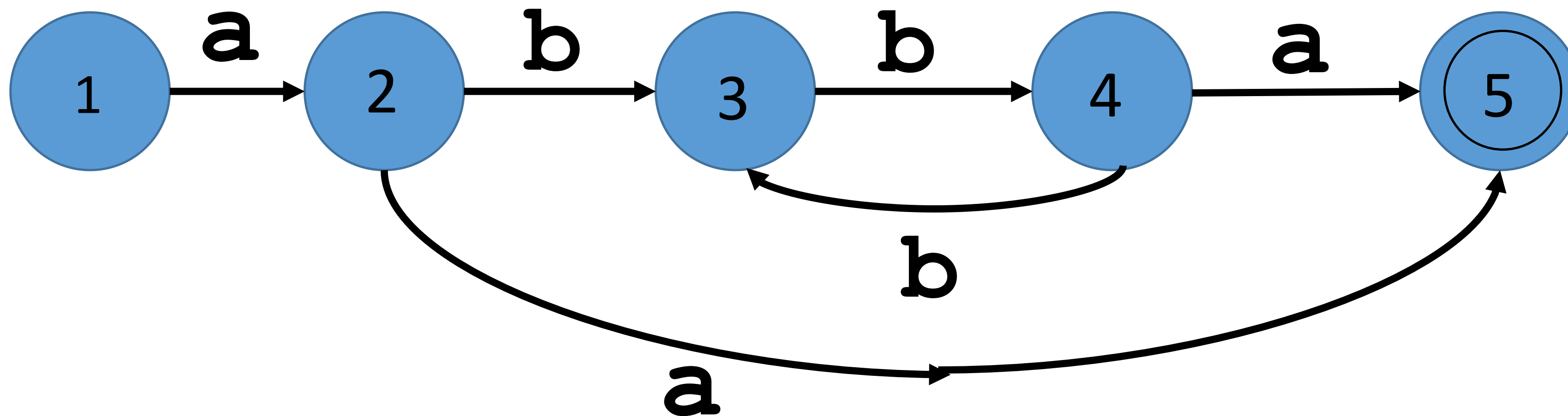
- A FSA graph can be converted to a table
- The table has cells for each state and each input symbol
- In the cell goes the next state if the DFA is in that state and receives that input symbol
- You can consider the state table to be an adjacency table for the graph

Converting an Example FSA

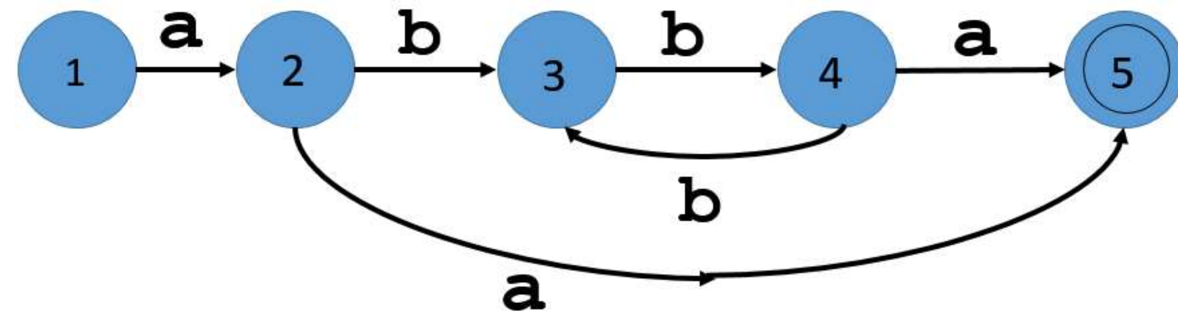
- Consider the regular expression that begins and ends with an **a** and can have an even number of **b**'s between them

a (bb)* a

- It can be recognized by the FSA



Convert the Graph to a Table



	1	2	3	4	5
a	2	5	0	5	0
b	0	3	4	3	0

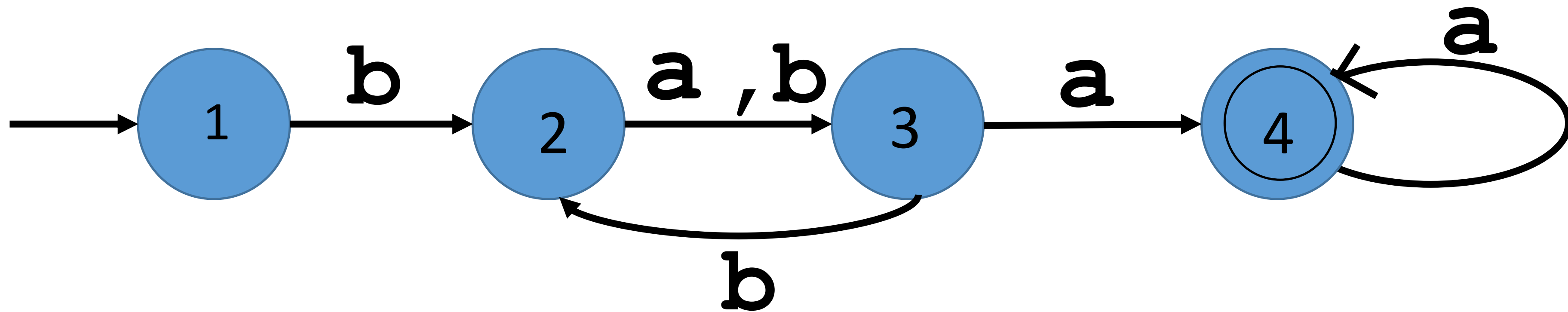
- The states are listed along the top
- The input symbols are along the side
- For that symbol while in that state, the DFA will go to the new state given in the table
- State zero represents a final error state

Draw a DFA for this Regular Language

$$(\texttt{bb} \mid \texttt{ba})\ \texttt{a}^{+}$$

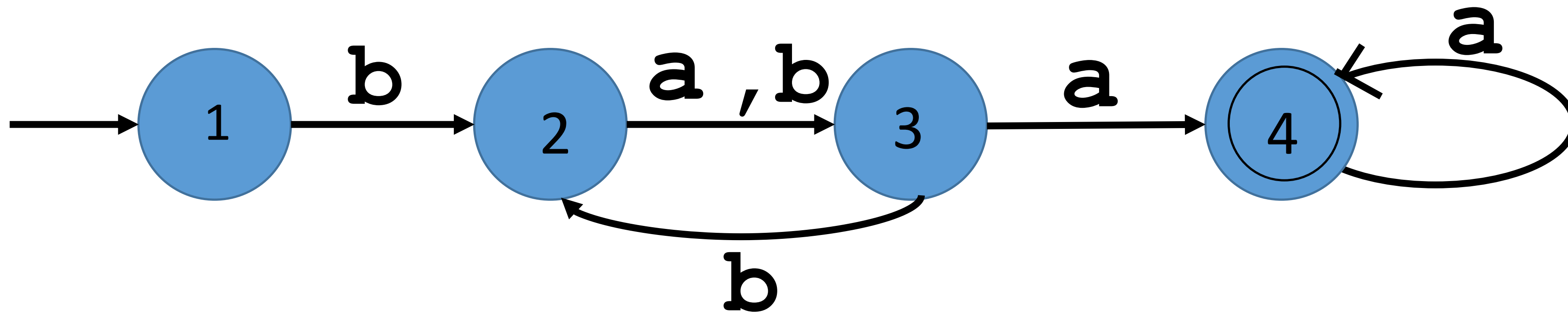
Possible Solution

(bb | ba) a⁺

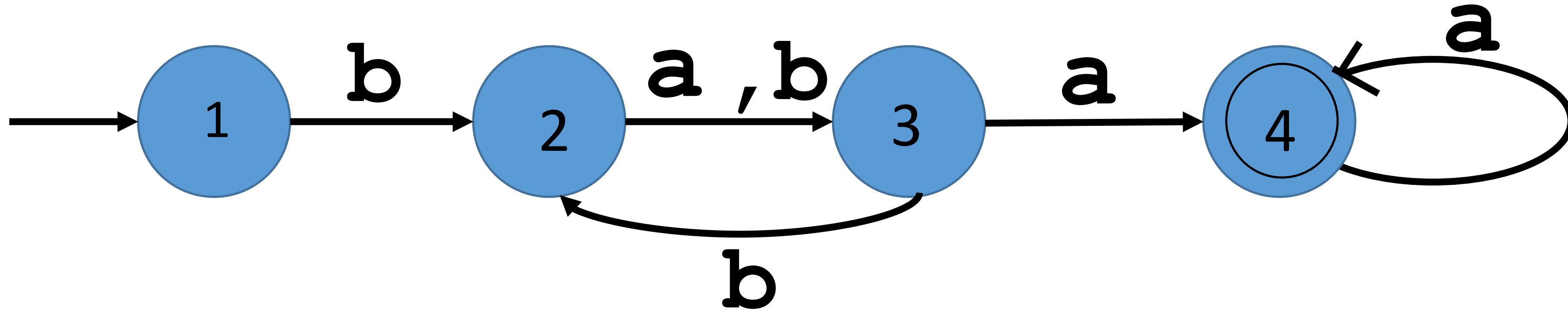


Create a state table for the FSA

$$(\mathtt{bb} \mid \mathtt{ba})\ \mathtt{a}^{+}$$

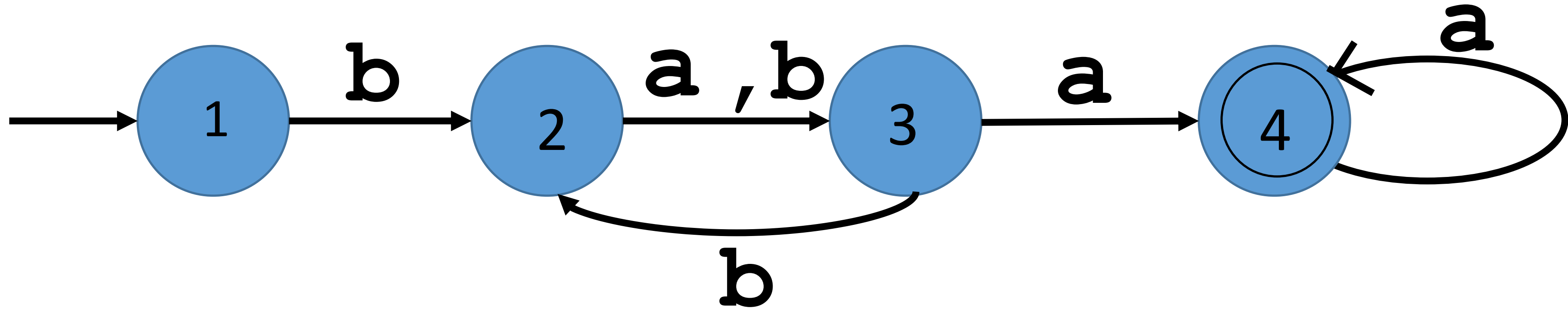


Complete the State Table



	1	2	3	4
a				
b				

Possible Solution



	1	2	3	4
a	0	3	4	4
b	2	3	2	0

Lexical Analysis

- Lexical Analysis or scanning reads the source code (or expanded source code)
- It removes all comments and white space
- The output of the scanner is a stream of tokens
- Tokens can be words, symbols or character strings
- A scanner can be a finite state automata (FSA)

Syntactic Analysis

- Syntactic Analysis or parsing reads the stream of tokens created by the scanner
- It checks that the language syntax is correct
- The output of the Syntactic Analyzer is a parse tree
- The parser can be implemented by a context free grammar stack machine

Simple Program

```
/*  This is an example program */
bull = (cow + cat)    *    dog;
goat = “quote”;
```

After Lexical Scan

bull	=	(	cow	+	cat	)
*	dog	;	goat	=	quote	;

- The lexical scanner creates a list of all tokens in the program
- Comments, line divisions and whitespace have been removed
- Words, numbers and quote strings are single tokens instead of individual characters
- The list could be an array, a linked list or an ArrayList

What is the correct order for a compiler?

A	B	C
parser	scanner	scanner
scanner	parser	semantic analysis
semantic analysis	semantic analysis	parser
code generation	code generation	code generation

Token Object

A token created by the lexical scanner should contain:

- String value; // text of the token
- int type; // type of value
  - name
  - number
  - punctuation
  - string
- int line number, column // position in source code

Useful Methods

- The token object might include useful methods such as:

boolean isVariable() – true of this token is a variable

boolean isConst() – true if this token is a constant

boolean isVarCont() – true if this is a variable or constant

- These methods will probably not be used by the lexical scanner, but will probably be useful to the parser

Machines of a Compiler

- Lexical Analysis – Finite State Automata
- Syntactic Analysis – Push Down Automata

- Since these simple theoretical machines can recognize the grammars we are using, compilers are based on these simple models

The tokens from the scanner are best stored in

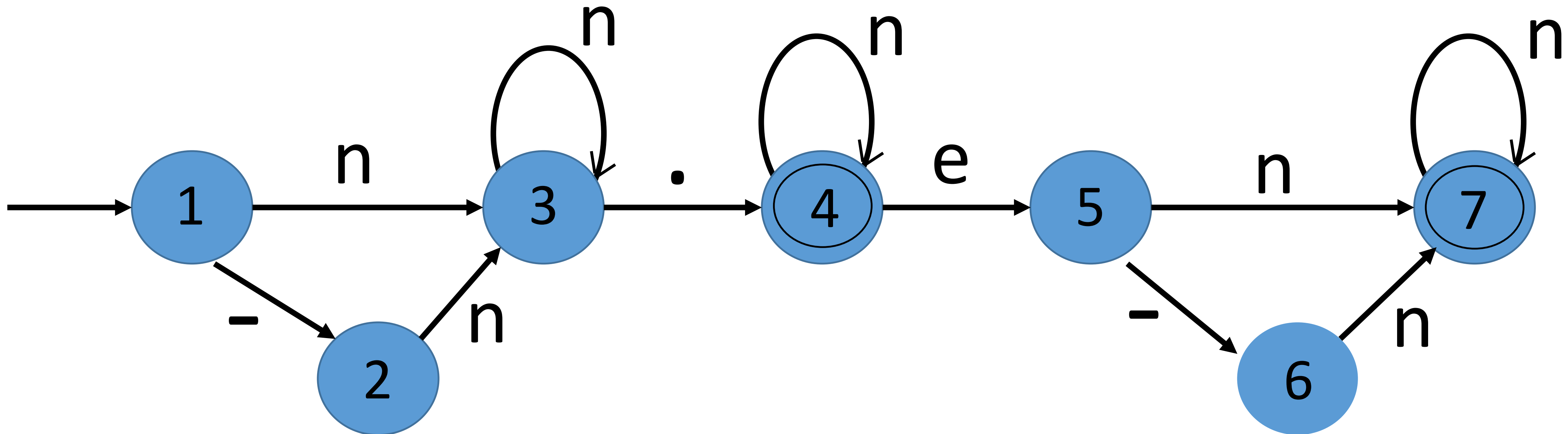
A. hash table of strings
B. hash table of objects
C. array of strings
D. array of objects
E. something even better

State Tables

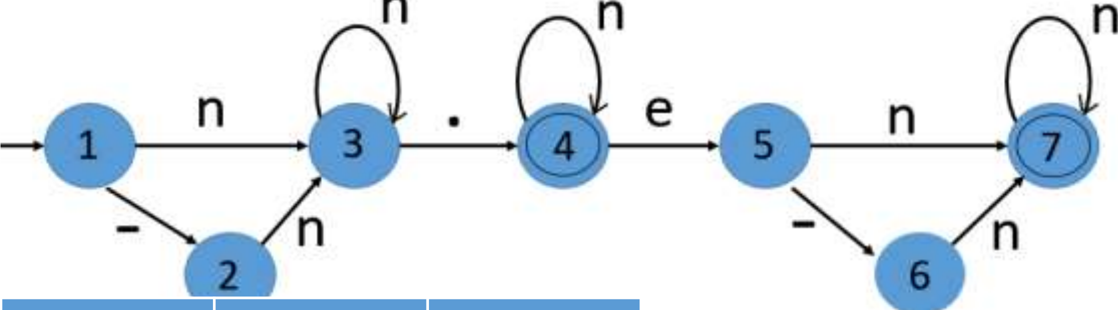
- An FSA graph can be converted to a table
- The table has cells for each state and each input symbol
- In the cell goes the next state if the DFA is in that state and receives that input symbol
- You can consider the state table to be an adjacency table for the graph

Example FSA

- Consider a FSA to recognize Java or C++ double constants of the form $-^{0..1}\ n^{+}\ .\ \ n^{*}\ (e\ -^{0..1}\ n^{+})^{0..1}$
- Examples 12.34 1. -1.23e-4 0.5



Convert the Graph to a Table



	1	**2**	**3**	**4**	**5**	**6**	**7**
n	3	3	3	4	7	7	7
-	2	0	0	0	6	0	0
.	0	0	4	0	0	0	0
e	0	0	0	5	0	0	0

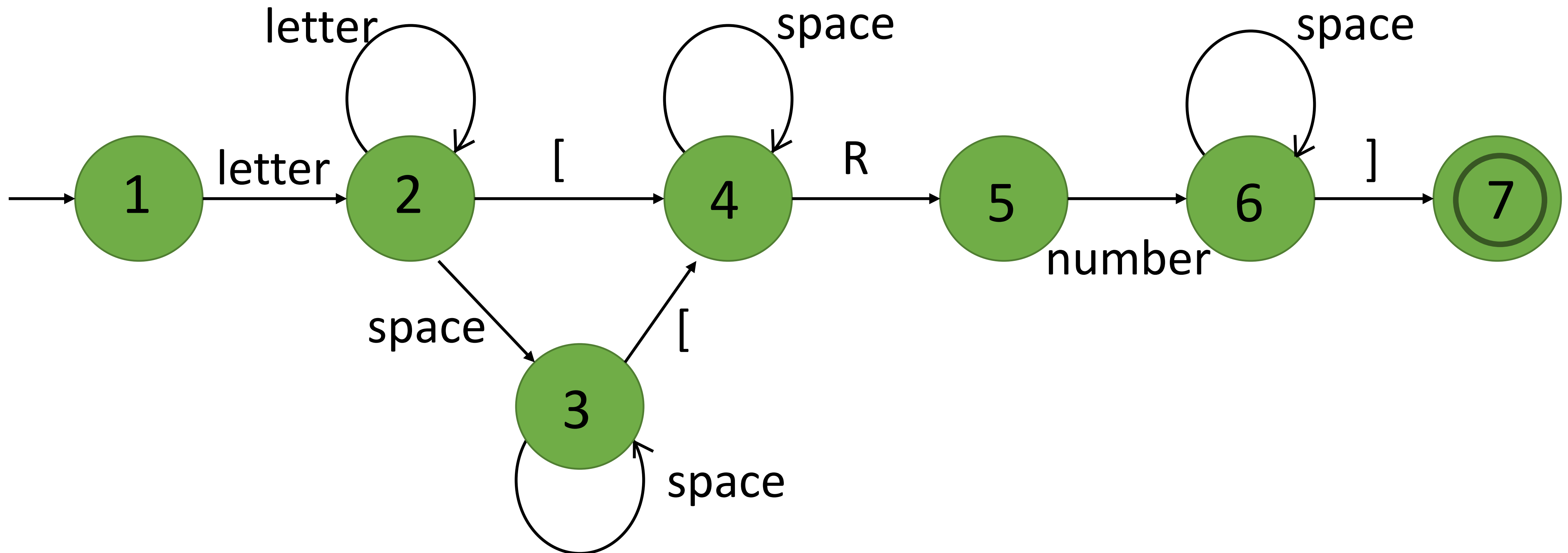
- The states are listed along the top
- The input symbols are along the side
- For that symbol while in that state, the DFA will go to the new state given in the table
- State zero represents a final error state

Write an FSA

- Create an FSA to recognize a name followed by a register (R*number*) in square brackets, e.g. **`stuff[R8]`**

Possible Solution

- Create an FSA to recognize a name followed by a register (R*number*) in square brackets, e.g. **`stuff[R8]`**



Programming a FSA

- It is relatively simple to implement a Finite State Automata in a modern programming language
- This program can be used to recognize if a string conforms to a regular language

FSA Program

```
state = 1
while not end of file {
        symbol = next input character
        state = stateTable[ symbol, state ]
        if state = 0 then error
}
if state is a terminating state, success
```

Grouping Input Symbols

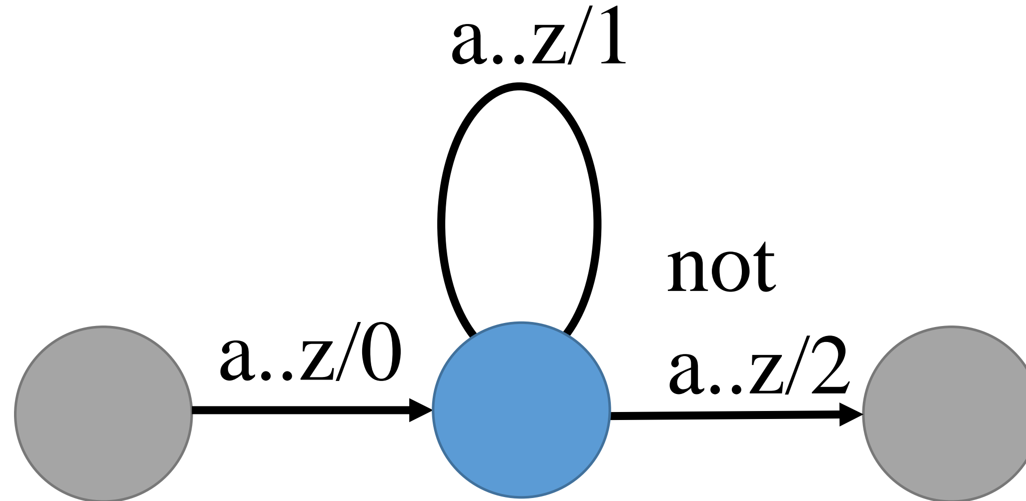
- For many FSA programs, it is useful to create an index value for the input symbols, i.e. a = 0, b = 1
- Often you can have groups of symbols use the same index value, i.e. all letters have the index 2 and all numbers have the index 3

Mealy and Moore Machines

- The FSA we have discussed so far simply determine if the input is valid for the specified language
- An FSA can also produce an output
- A Mealy machine has an output or function associated with each transition or edge of the graph
- A Moore machine has an output or function associated with each state or node of the graph

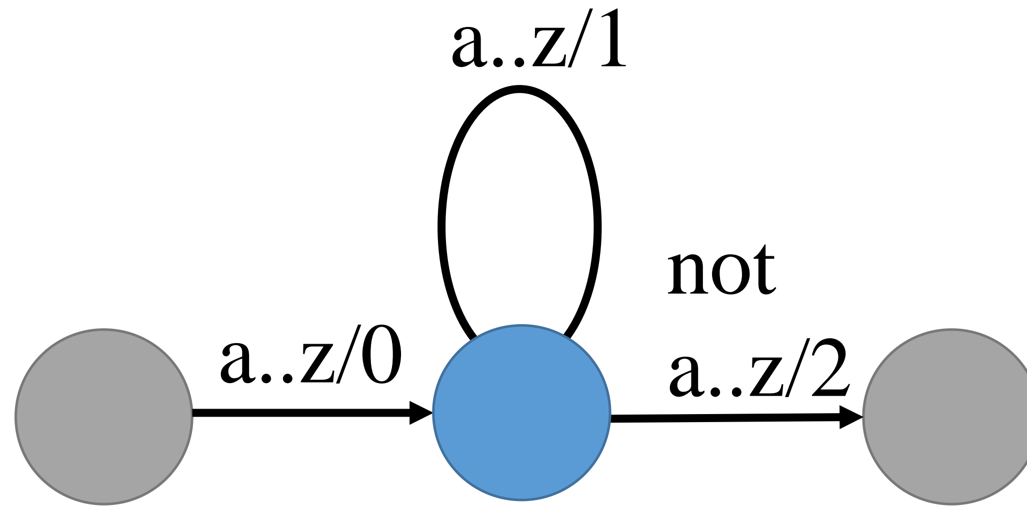
Using Mealy Machines to Create Tokens

- Consider an FSA reading text and creating token of words separated by spaces or punctuation



- 0 = Save character as first letter in a string
- 1 = Save character as next letter in a string
- 2 = Save the string as a token, handle other symbol

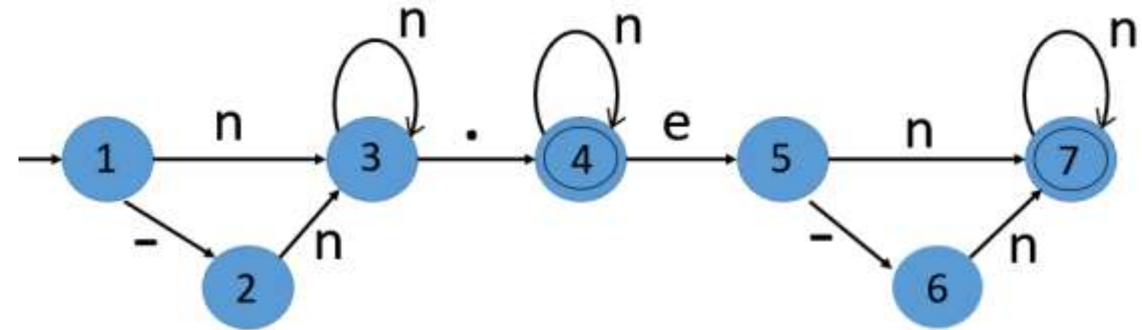
Mealy Machine State Table



	1	2	3
a .. z	2/0	2/1	2/0
not a .. z	1/x	3/2	3/x

New state / Output function

Creating decimal number token



	1	2	3	4	5	6	7
n	3/1	3/2	3/2	4/2	7/2	7/2	7/2
-	2/1	0	0	0	6/2	0	0
.	0	0	4/2	0	0	0	0
e	0	0	0	5/2	0	0	0

Actions

0 = do nothing

1 = create new token with first character

2 = append character to end of existing token

Lexical Analysis with a Mealy Machine

- Compilers can use a Mealy machine to scan the source code
- The FSA recognizes and discards comments and white space
- Names, numbers, strings and punctuation are each output as a list of tokens
- A token is an object that contains one unit of the input specifying the value and type

Assembler Assignment

- The assembler program due today
- Upload the source code of your assembler to Blackboard before midnight

FSA Assignment

- The second assignment in COMP360 has been posted to Blackboard
- You have to write some Regular Expressions and draw some FSA
- Due Monday, February 3, 2020 at 2:00pm
- You can hand in paper at the beginning of class or submit your answers to Blackboard in any readable format