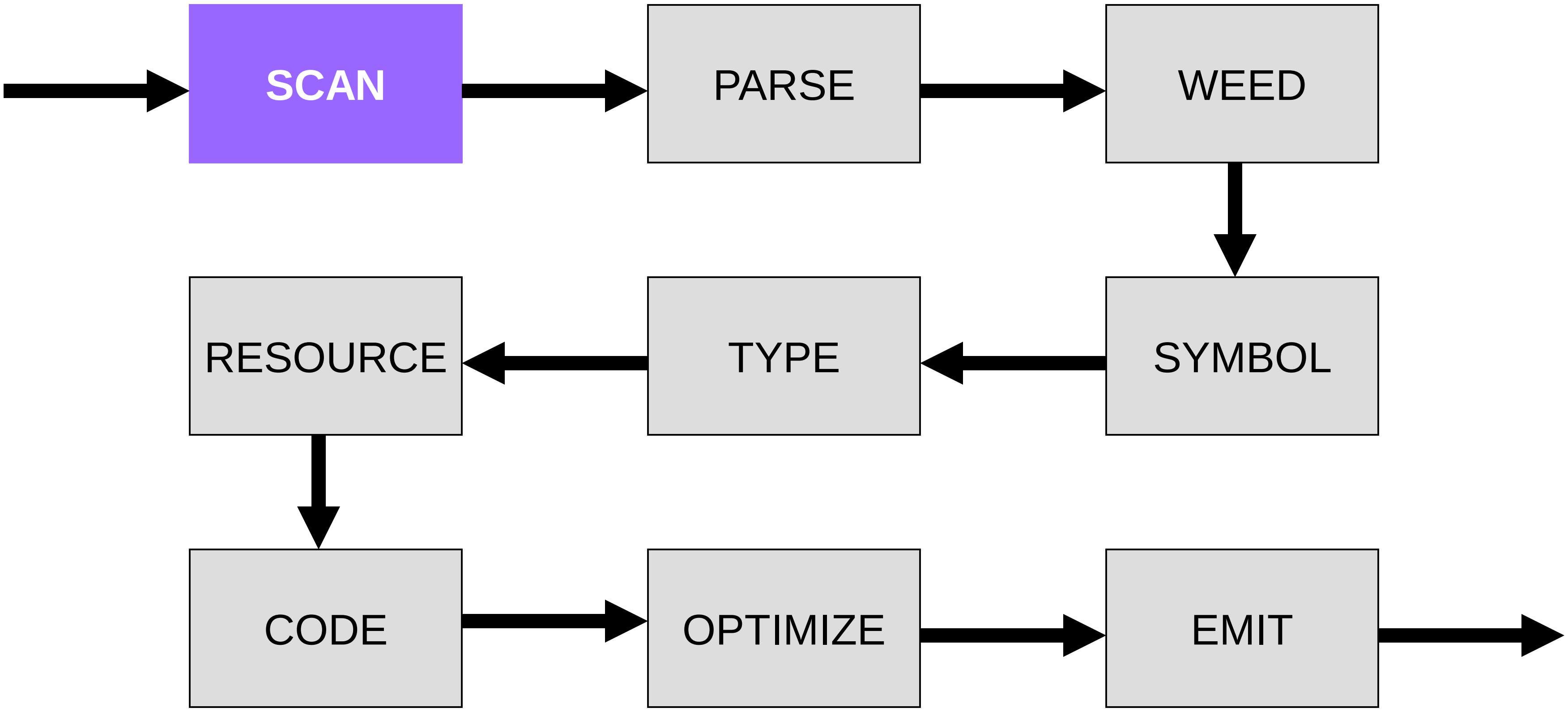
Compiler

Scanning

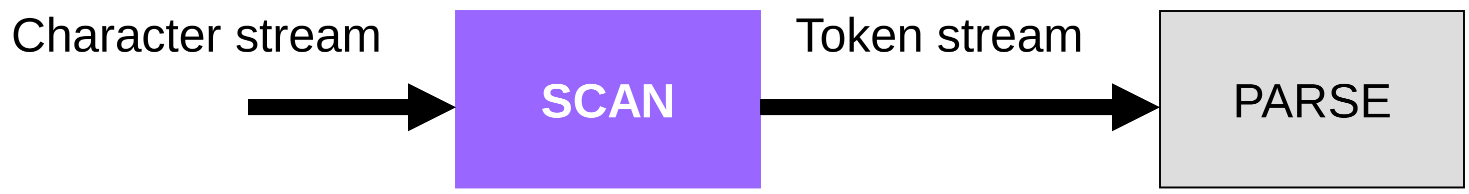
© Copyright 2005, Matthew B. Dwyer and Robby. The syllabus and lectures for this course are copyrighted materials and may not be used in other course settings outside University of Nebraska-Lincoln and Kansas State University in their current form or modified form without express written permission of one of the copyright holders. During this course, students are prohibited from selling notes to or being paid for taking notes by any person or commercial firm without the express written permission of one of the copyright holders.

Compiler Architecture



Compiler Architecture

Source Code



Scanner: Overview

- A scanner transforms a string of characters into a string of *symbols/tokens*:
  - it corresponds to a *finite-state machine* (FSM);
  - plus some code to make it work;
  - FSM can be generated from specification.
- Symbols (*a.k.a.* tokens, lexemes) are the indivisible units of a languages syntax
  - key words, punctuation symbols, identifiers …
- A FSM recognizes the structure of a symbol
  - that structure is specified as a regular expression

Token Definitions

Described in language specification, e.g.:

“An *identifier* is an unlimited-length sequence of *Java letters* and *Java digits*, the first of which must be a Java letter.
An identifier cannot have the same spelling (Unicode character sequence) as a keyword (§3.9), Boolean literal (§3.10.3), or the null literal (§3.10.7).”

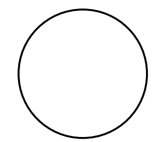
http://docs.oracle.com/javase/specs/

Finite State Machine (FSM) or Finite State Automaton (FSA)

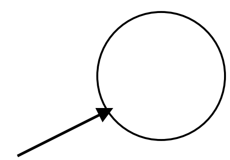
- a quintuple $(\boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{s_0}, \boldsymbol{\delta}, \boldsymbol{F})$, where
  - Σ, is a finite non-empty set of symbols (input alphabets)
  - S, is a finite non-empty set of states
  - $s_0 \in S$, is the initial state
  - $\delta: S \times \Sigma \to S$, is the state transition function
  - $F \subseteq S$, is the (possibly empty) set of accepting (~~final~~) states

FSM Graphs

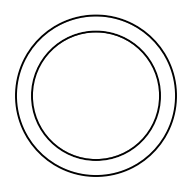
- A state



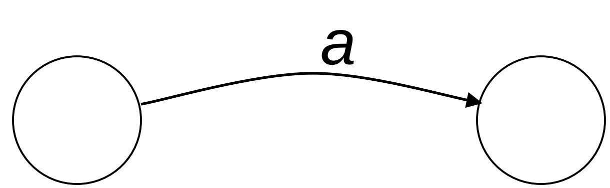
- The start state



- An accepting state



- A transition *a*

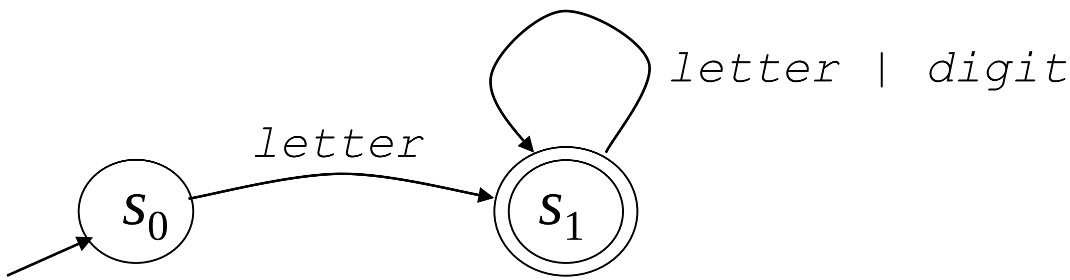


FSM Interpretation

- Transition: $s_1 \xrightarrow{a} s_2$
- Is read: in state s_1 on input a go to state s_2
- At end of input
  - if in accepting state => *accept*
  - otherwise => *reject*
- If no transition possible => reject

Language defined by FSM

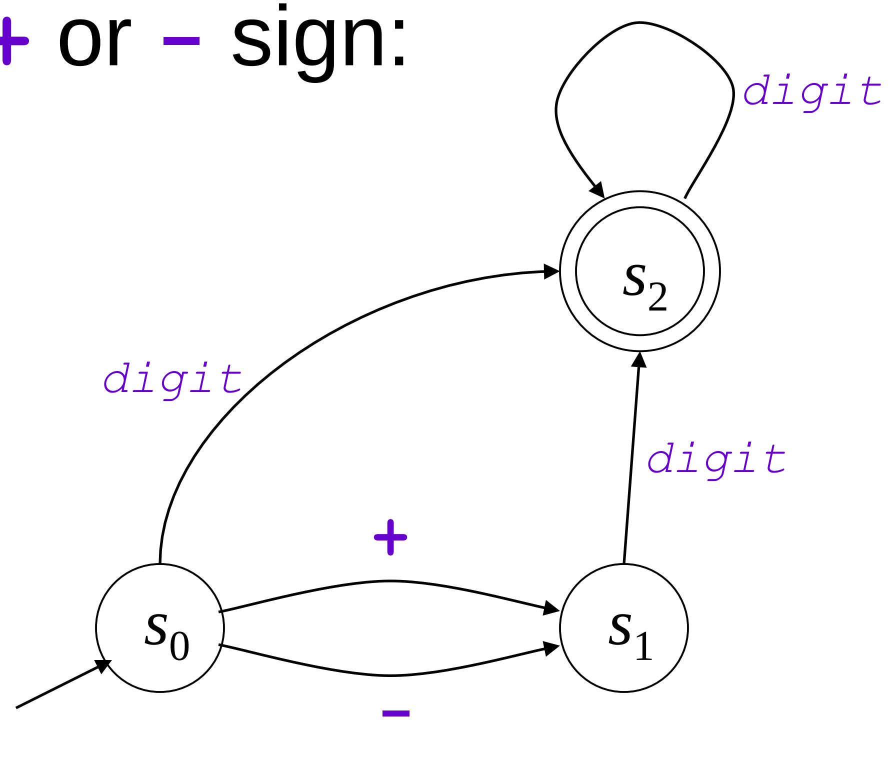
- The *language defined by an FSM* is the set of strings accepted by the FSM.



  - in the language of the FSM shown above:
    - **`x`**, **`tmp2`**, **`XyZzy`**, **`position27`**.
  - *not* in the language of the FSM shown above:
    - **`123`**, **`a?`**, **`13apples`**.

Example: Integer Literals

- FSM that accepts integer literals with an optional **+** or **–** sign:



Two kinds of FSM

Deterministic (DFA):

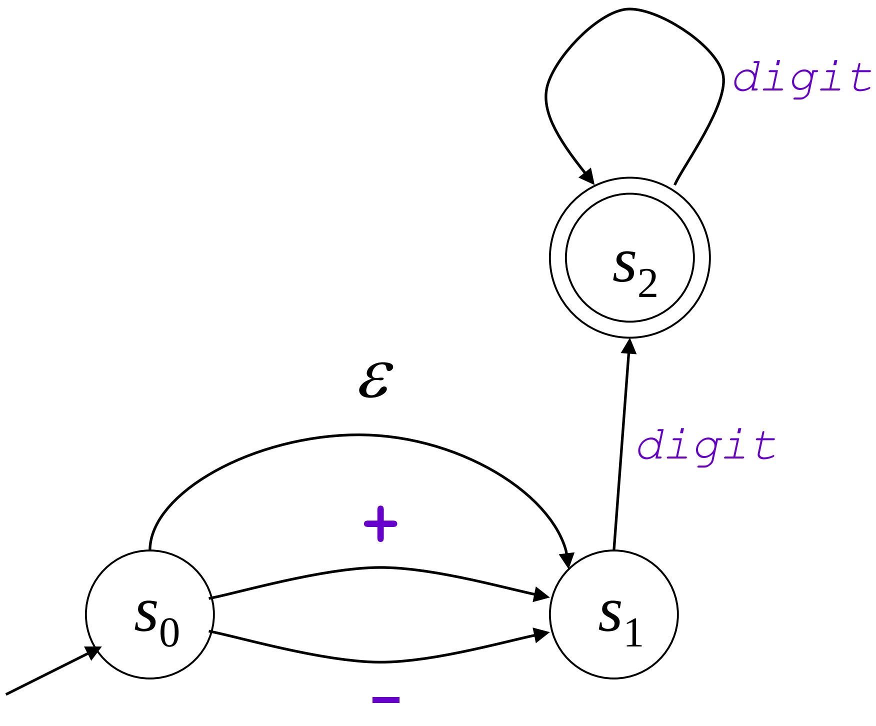
- No state has more than one outgoing edge with the same label.

Non-Deterministic (NFA):

- States *may* have more than one outgoing edge with the same label.
- Edges may be labeled with ε (epsilon), the empty string.
- The automaton can take an ε transition *without* looking at the current input character.

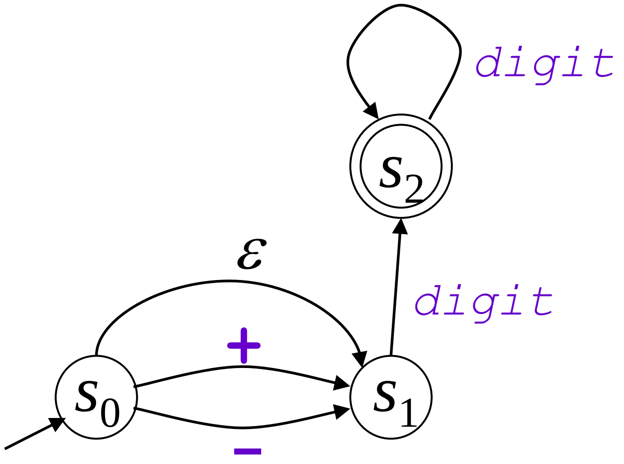
Example of NFA

- integer-literal example:



NFA

- sometimes simpler than DFA
- can be in multiple states at the same time
- NFA accepts a string if
  - there exists a sequence of moves
  - starting in the start state,
  - ending in an accepting state,
  - that consumes the <u>entire</u> string.



- Example:
  - the integer-literal NFA on input "**+75**"

Equivalence of DFA and NFA

- Theorem:
  - For every non-deterministic finite-state machine *M*, there exists a *deterministic* machine *M'* such that *M* and *M'* accept the *same* language.

- DFA are easy to implement
- NFA are easy to generate from specifications
- Algorithms exist to convert NFA to DFA (*see assignments*)

Regular Expressions (RE)

- Automaton is a good “visual” aid
  - but is not suitable as a specification
- Regular expressions are a suitable *specification*
  - a <u>compact</u> way to define a language that can be accepted by an automaton.
- used as the input to a scanner generator
  - define each token, and also
  - define white-space, comments, etc.
    - these do not correspond to tokens, but must be recognized and ignored.

Example: Pascal Identifier

- Lexical specification (in English):
  - a letter, followed by zero or more letters or digits.
- Lexical specification (as a regular expression):
  - *letter (letter | digit)**

|	means "or"
	means "followed by"
*	means zero or more instances of
( )	are used for grouping

Operands of RE Operators

- Operands are same as labels on the edges of an FSM
  - single characters or ε
- *`letter`* is a shorthand for
  - **`a | b | c | ... | z | A | ... | Z`**
- *`digit`* is a shorthand for
  - **`0 | 1 | … | 9`**
- sometimes we put the characters in quotes
  - necessary when denoting `| *`

Operator Precedence

Regular Expression Operator	Analogous Arithmetic Operator	Precedence
|	plus	lowest
	times	middle
*	exponentiation	highest

Consider regular expressions:

```
letter letter | digit*
letter (letter | digit)*
```

For You To Do

- Describe (in English) the language defined by each of the following regular expressions:

letter (*letter* | *digit**)

*digit digit** "." *digit digit**

Example: Integer Literals

- An integer literal with an optional sign can be defined in English as:

  "(nothing or + or -) followed by one or more digits"

- The corresponding regular expression is:

$$(+ \mid - \mid \varepsilon)\ (\mathit{digit}\ \mathit{digit}^*)$$

- Convenience operators

$a+$	is the same as	$a\ (a)^*$
$a?$	is the same as	$(a \mid \varepsilon)$

$$(\text{"+"} \mid -)?\ \mathit{digit}+$$

Language Defined by RE

- Recall: language = set of strings
- Language defined by an automaton
  - the set of strings accepted by the automaton
- Language defined by a regular expression
  - the set of strings that match the expression.

Regular Exp. r		Corresponding Set of Strings $L(r)$	
ε		{""}	
`a`		{"`a`"}	
`r s`	`a b c`	$L(r)\,L(s)$	{"`abc`"}
`r | s`	`a|b|c`	$L(r) \cup L(s)$	{"`a`", "`b`", "`c`"}
`r*`	`(a|b)*`	$L(r)^*$	{"", "`a`", "`b`", "`aa`", "`ab`", ..., "`bbab`“, ...}

The Role of Regular Expressions

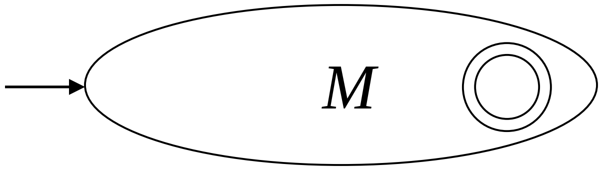
- Theorem:
  - For every regular expression, there exists a nondeterministic finite-state machine that defines the same language, and vice versa.

- Q: Why is the theorem important for *scanner generation*?

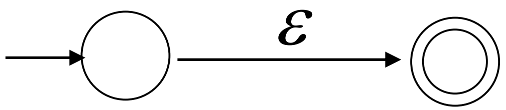
- Q: Theorem is not enough: what do we need for *automatic* scanner generation?

Regular Expressions to NFA (1)

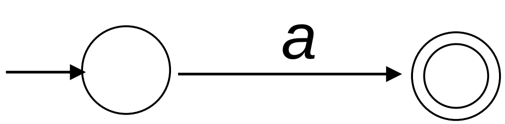
- For each kind of RE, define an NFA
  - Notation: NFA for RE *M*



ε

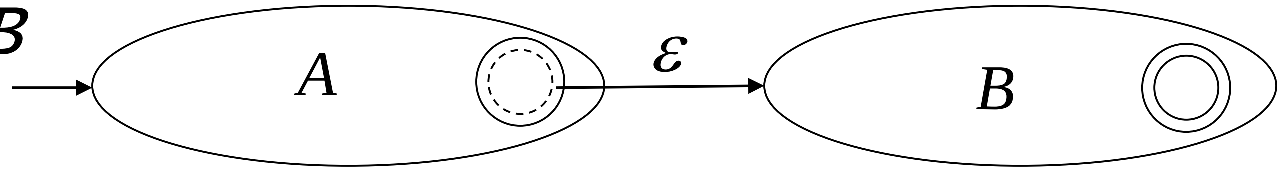


a

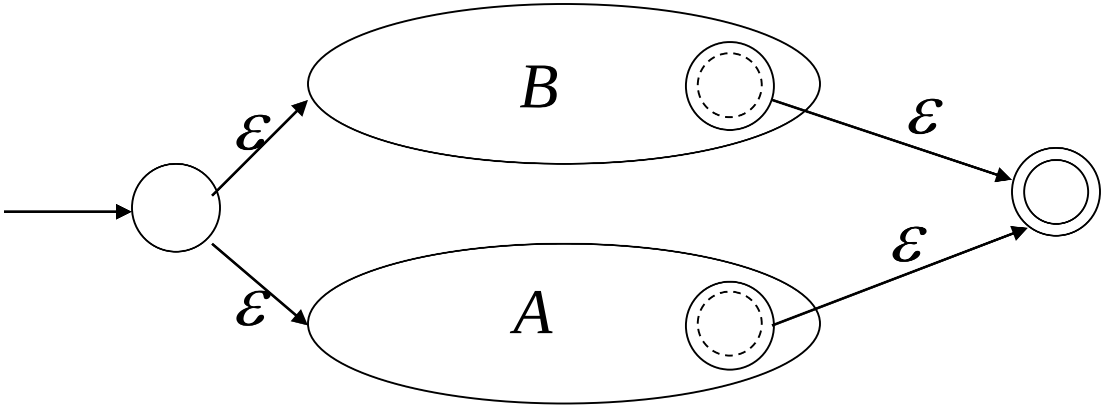


Regular Expressions to NFA (2)

A B

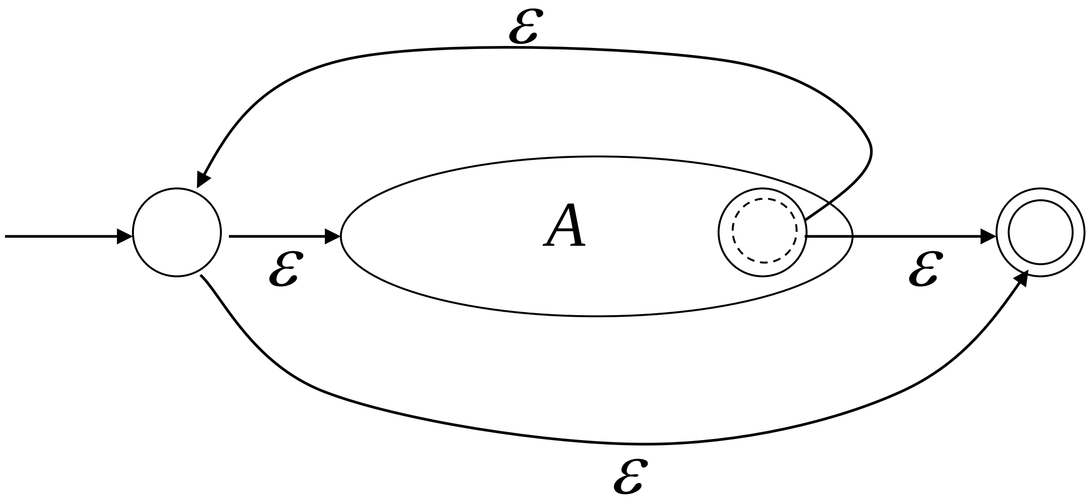


A | *B*



Regular Expressions to NFA (3)

A^*

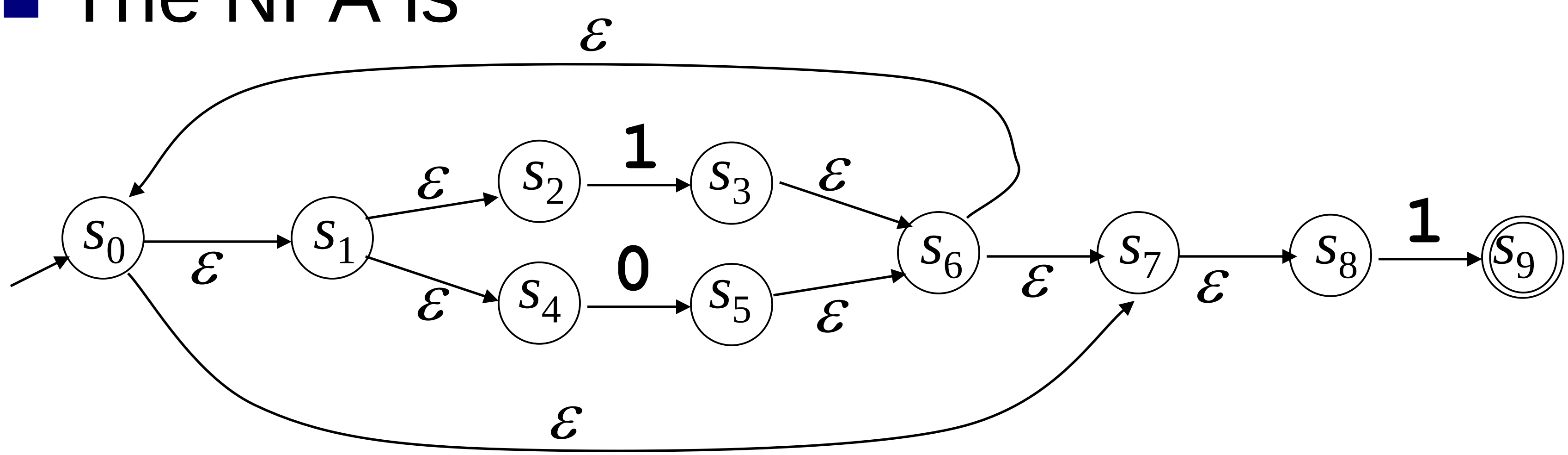


Example : RE to NFA

- Consider the regular expression

  (1|0)*1

- The NFA is



Putting It All Together

- Specify regular expression for each token
  - Generate NFA and convert to DFA
- Define appropriate action for each token
  - *ignore* comments and whitespace
  - *return string* for identifier or numeric constant
  - *indicate* keyword, operator, punctuations, …
- Associate patterns and actions
- Integrate matching of all possible patterns

Example : Expressions

operators: `"*"`, `"/"`, `"+"`, `"-"`

parentheses: `"("`, `")"`

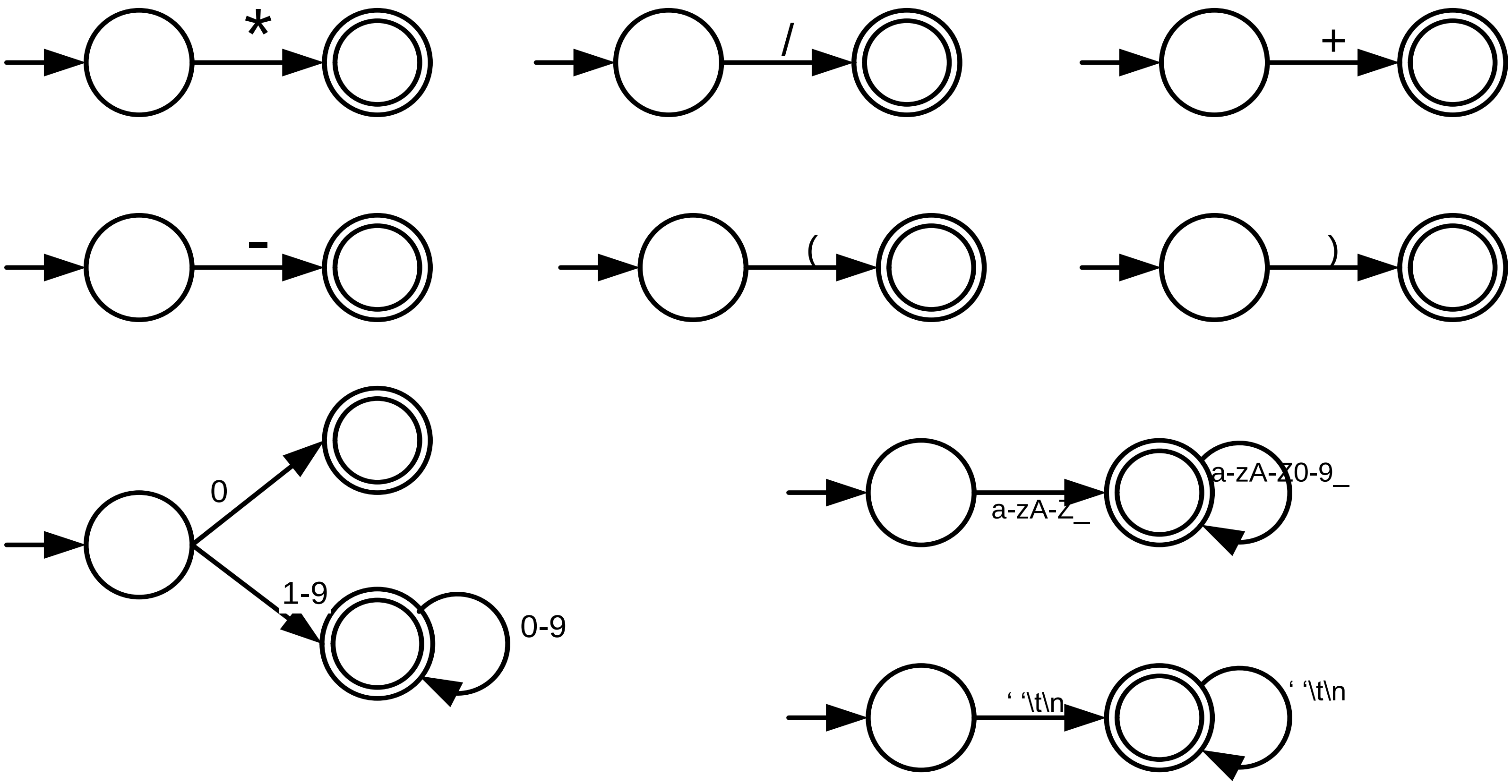
integer constants: `0|([1-9] [0-9]*)`

identifiers: `[a-zA-Z_][a-zA-Z0-9_]*`

white space: `[ \t\n]+`

where: `[abc] = (a|b|c)`

Symbol DFAs



Scanner Algorithm

Given DFA $D_1, \ldots, D_n$

```
while input is not empty do
      foreach i ∈ {1,..,n} do
            s_i := the longest prefix that D_i accepts;
      k := max{|s_i|  |  i ∈ {1,..,n} };
      if k > 0 then
            remove k characters from input;
            j := min{i | |s_i| = k };
            perform the j^th action
      else
            move one character from input to output
            // alternatively produce scan error
```

For You To Do

- What if more than one prefix matches a pattern?
  - Which prefix is used?
- What if a prefix matches more than one pattern?
  - Which pattern is used?
- What happens if a string matches no patterns?