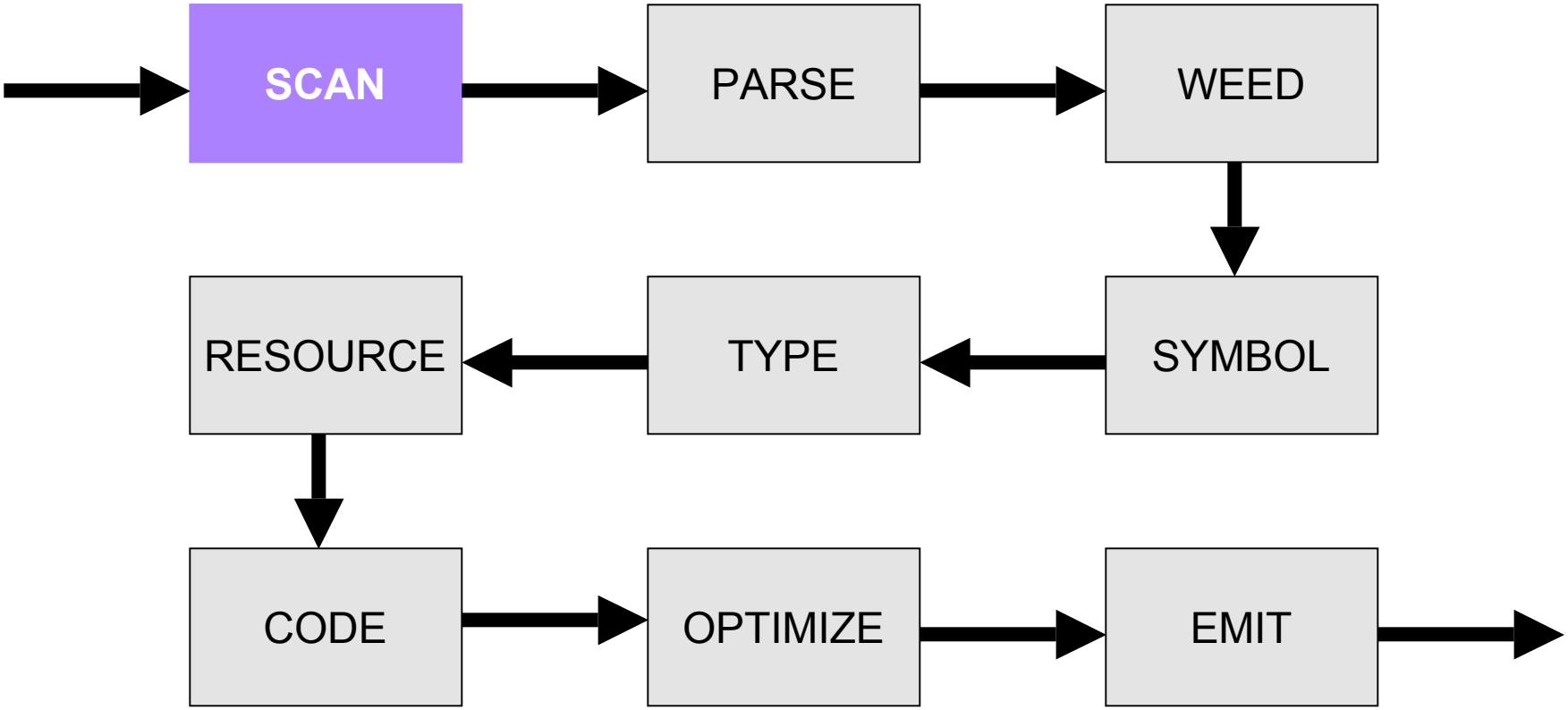
Scanning

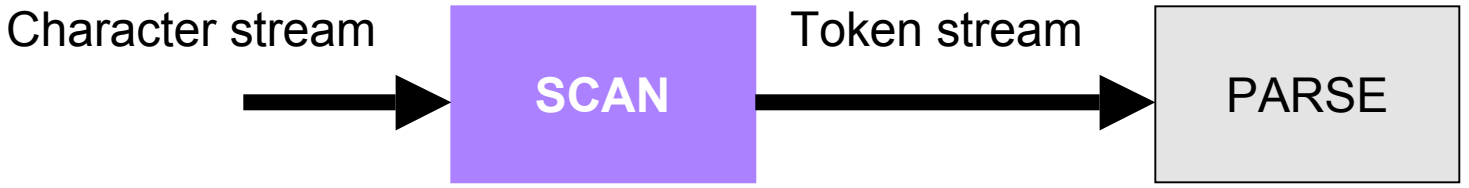
Matthew Dwyer
365 Avery Hall
http://www.cse.unl.edu/~dwyer

Compiler Architecture



Compiler Architecture

Source Code



Scanner : Overview

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

Token Definitions

Described in language specification:

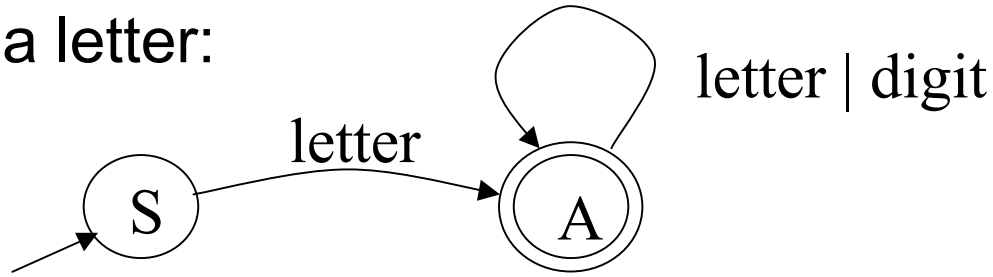
"An *identifier* is an unlimited-length sequence of *Java letters* and *Java digits*, the first of which must be a Java letter.

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://java.sun.com/docs/books/jls/html/3.doc.html#40625

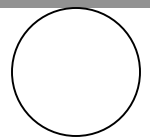
Finite State Machine

- A FSM is similar to a compiler in that:
  - A compiler recognizes legal *programs* in some (source) language.
  - A finite-state machine recognizes legal *strings* in some language.
- Example: Pascal Identifiers
  - sequences of one or more letters or digits, starting with a letter:

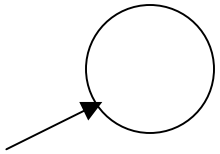


FSM Graphs

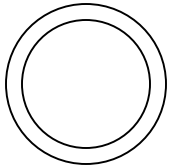
- A state



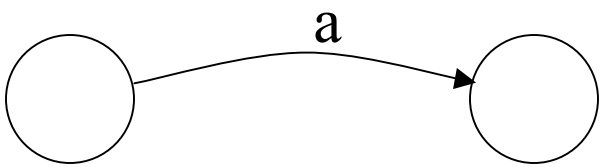
- The start state



- An accepting state



- A transition



FSM Interpretation

- Transition: $s_1 \rightarrow^a s_2$
- 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 a 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.

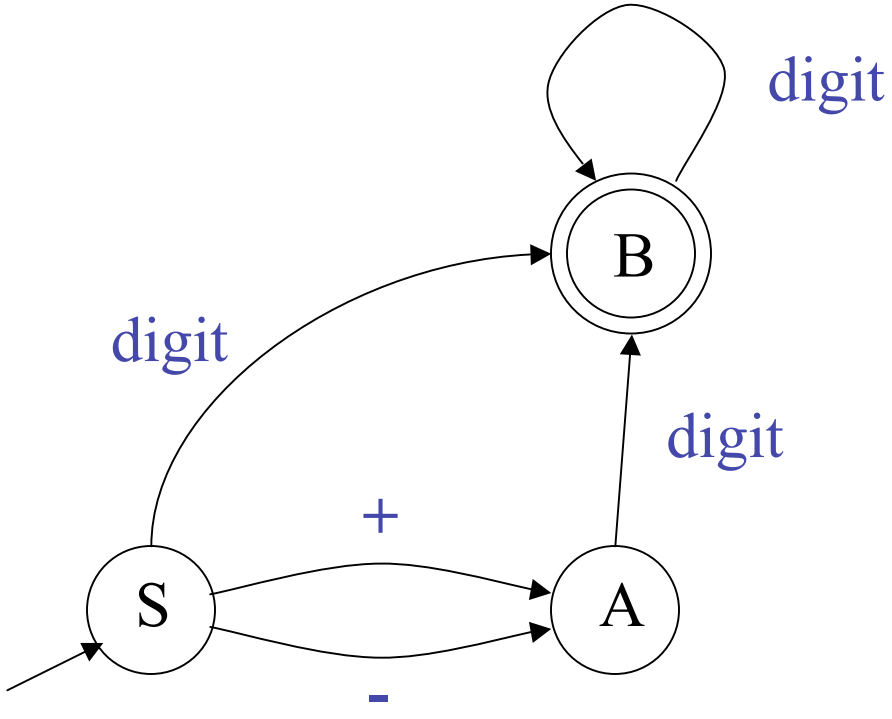
For You To Do

- Write an automaton that accepts Java identifiers
  - one or more letters, digits, or underscores, starting with a letter or an underscore.

- Write a finite-state machine that accepts only Java identifiers that do not end with an underscore

Example: Integer Literals

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



Two kinds of FSM

Deterministic (DFA):

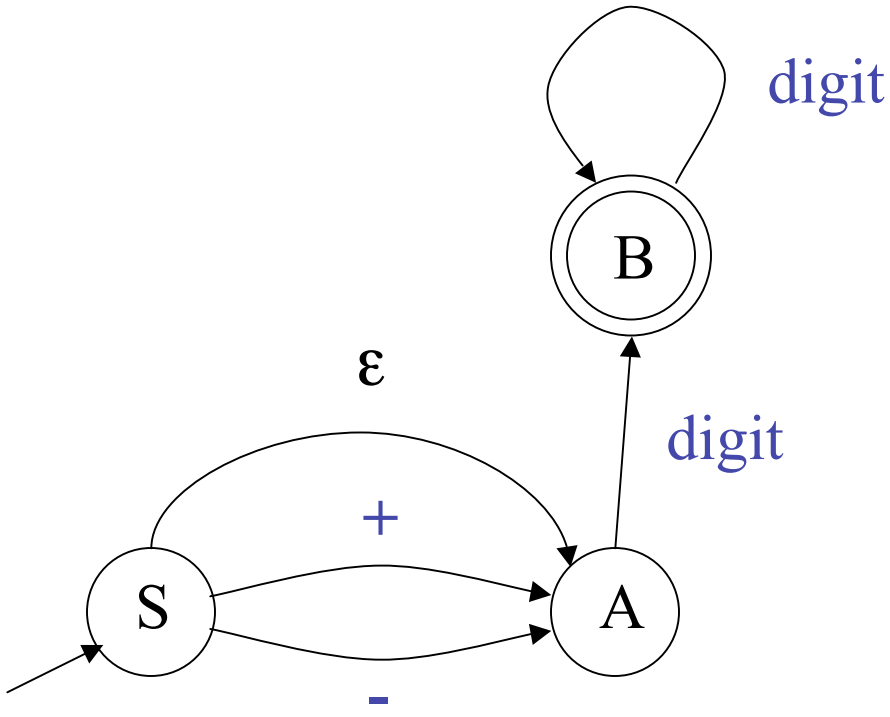
- No state has more than one outgoing edge with the same label.

Non-Deterministic (NFA):

- States *may* have more than one outgoing edge with same label.
- Edges may be labeled with ε (epsilon), the empty string.
- The automaton can take an ε epsilon 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 a final state,
  - that consumes the entire 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

Regular Expressions (RE)

- Automaton is a good “visual” aid
  - but is not suitable as a specification
- regular expressions are a suitable *specification*
  - a compact 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 l digit*

letter (letter l 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:

  (+|-|ε) (digit digit*)

- Convenience operators

a+	is the same as	a (a)*
a?	is the same as	(a | ε)

(+|a)? digit+

Language Defined by RE

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

Regular Exp.	**Corresponding Set of Strings**
ε	{""}
a	{"a"}
a b c	{"abc"}
a | b | c	{"a", "b", "c"}
(a | b | c)*	{"", "a", "b", "c", "aa", "ab", ..., "bccabb" ...}

The Role of Regular Expressions

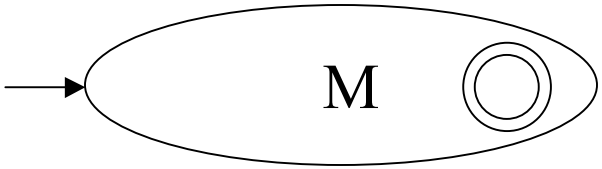
- Theorem:
  - for every regular expression, there is a deterministic finite-state machine that defines the same language, and vice versa.

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

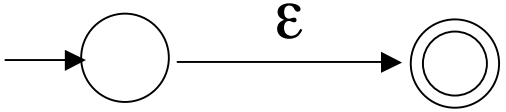
- Q: Is this theorem sufficient for *automatic* scanner generation?

Regular Expressions to NFA (1)

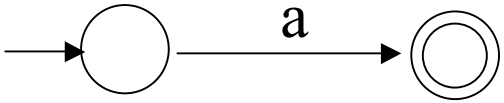
- For each kind of RE, define an NFA
  - Notation: NFA for RE M



ε

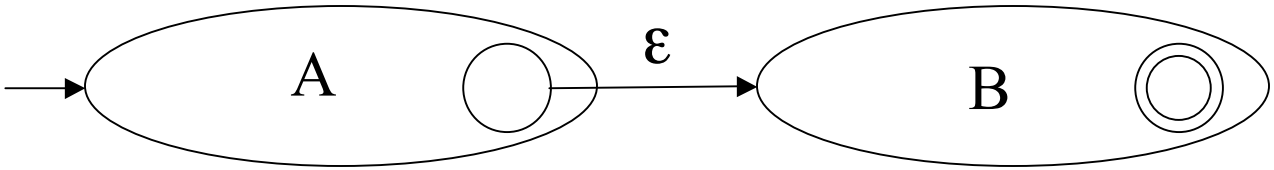


a

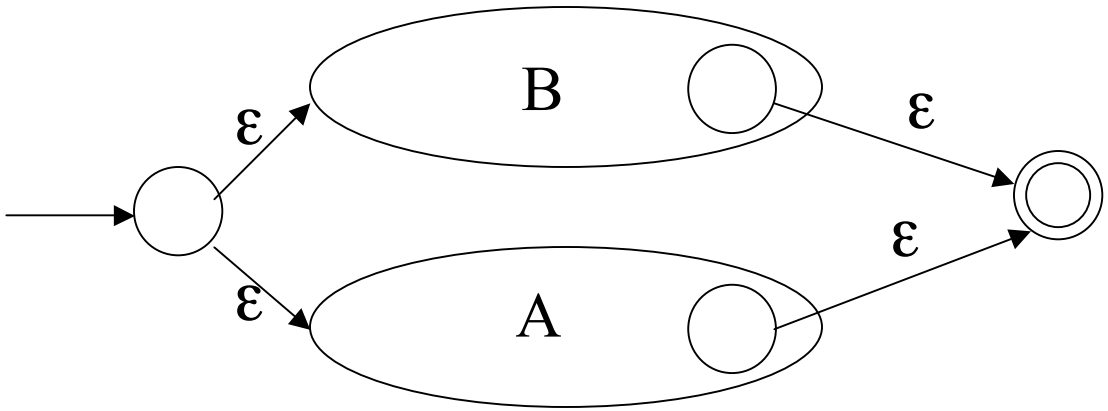


Regular Expressions to NFA (2)

A B

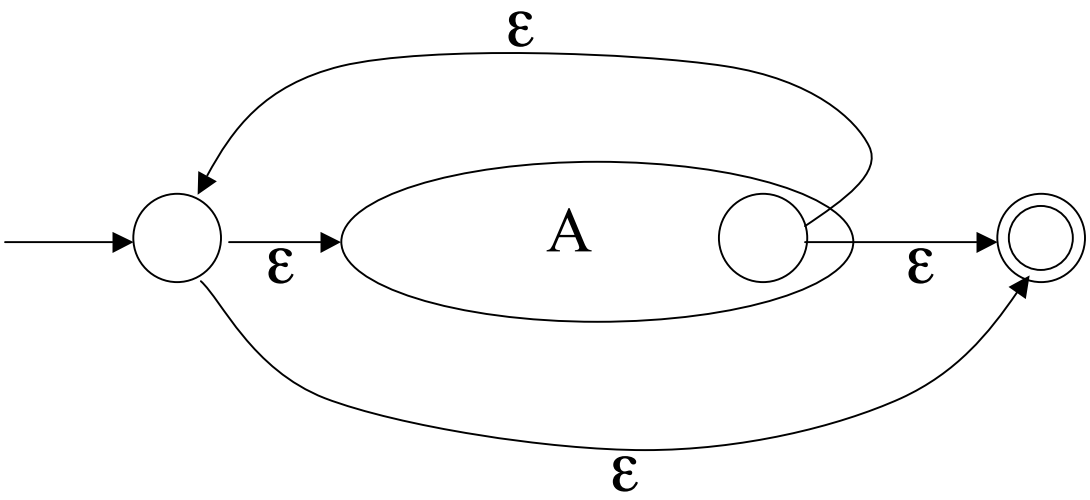


A | B



Regular Expressions to NFA (3)

A*

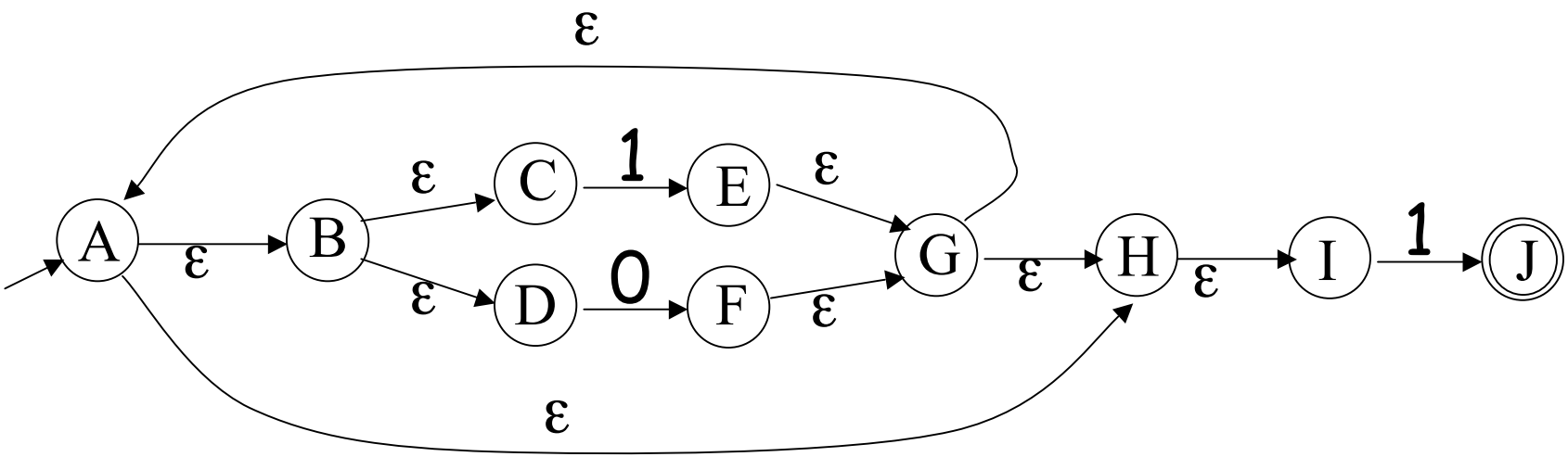


Example : RE to NFA

- 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 or operator
- Associate patterns and actions
- Integrate matching of all possible patterns

Example : Expressions

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

parentheses: "(", ")"

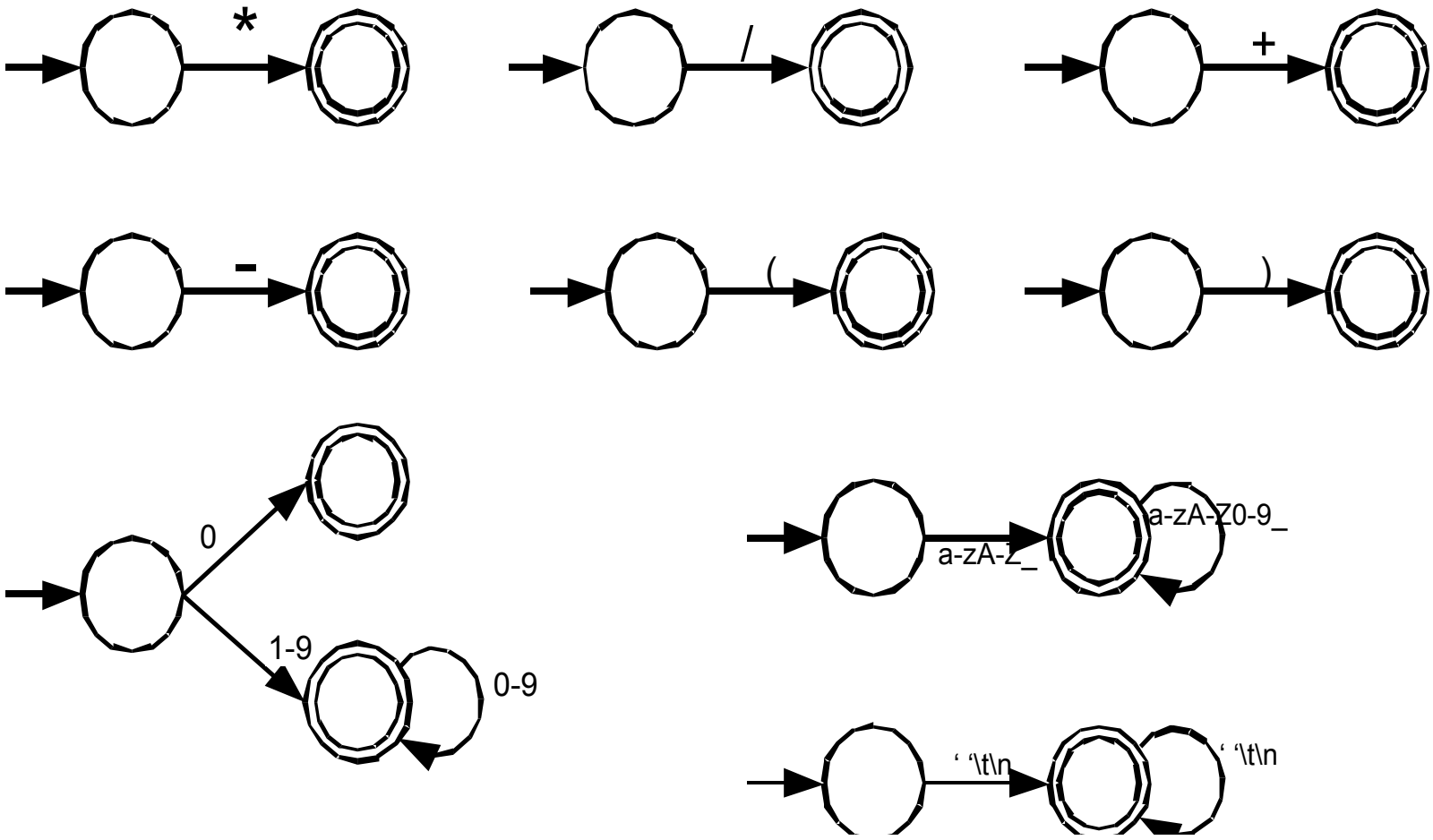
integer constants: 0|([1-9][0-9]*)

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
        s_i := the longest prefix that D_i accepts;
        k := max{ |s_i| };
        if k > 0 then
                j := min  { i: |s_i|=k };
                remove s_j from input;
                perform the j^th action
        else
                move one character from input to output
        end
end
```

For You To Do

- What if more than one string matches a pattern?
  - Which string is used?
- What if a string matches more than one pattern?
  - Which pattern is used?
- What happens if a string matches no patterns?
  - Are there “implicit” patterns?

ANTLR Scanner for StaticJava

```
class SJLexer extends Lexer; // declares an ANTLR lexer named SJLexer


LPAREN:  '('; // declares a character token named LPAREN
RPAREN:  ')';
LBRACK:  '[';
RBRACK:  ']';
LCURLY:  '{';
RCURLY:  '}';
COMMA:   ',';
DOT:     '.';
ASSIGN:  '=';
NOT:     '!';
DIV:     '/';
PLUS:    '+';
MINUS:   '-';
STAR:    '*';
MOD:     '%';
GT:      '>';
LT:      '<';
```

ANTLR Scanner for StaticJava

```
SEMI:     ';';
EQUAL:    "=="; // declares a string token named EQUAL
LE:       "<=";
NOT_EQUAL: "!=";
GE:       ">=";
LAND:     "&&";
LOR:      "||";

IDENT:    ( 'a'..'z' | 'A'..'Z' | '_' | '$' )
          ( 'a'..'z' | 'A'..'Z' | '_' | '0'..'9' | '$' )*;

NUM_INT: ( '0' |('1'..'9') ('0'..'9')* );

WS:       ( ' ' | '\t' | '\f'
          | ( options {generateAmbigWarnings=false;}:
              "\r\n" | '\r' | '\n' )
            { newline(); } // tell ANTLR to increment line & reset column
          )+
          { _ttype = Token.SKIP; }; // tell ANTLR to skip this WS token
```