Lexical Analysis

- Readings
  - Sections 2.1, 2.6
  - Chapter 3
- Topics
  - Scanners
  - Finite Automata
  - Regular Expressions
  - Conversion Processes
  - Automating an Automaton

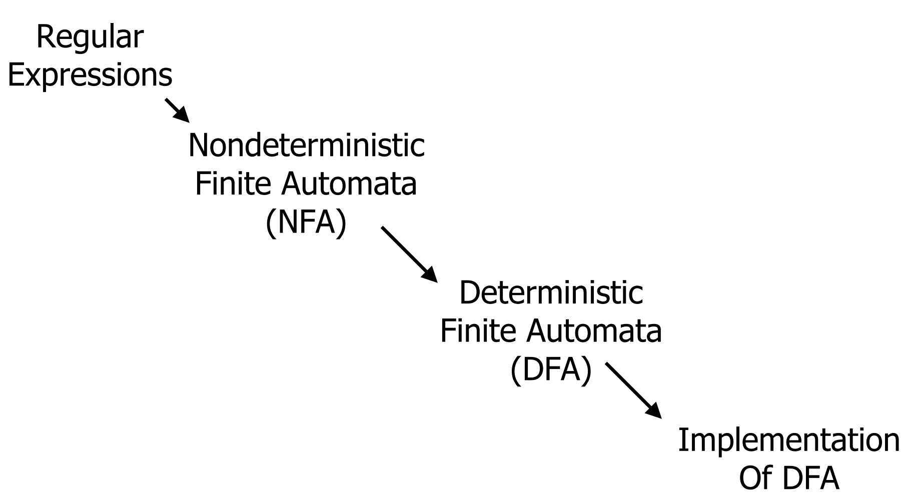
Scanner

- Translate a sequence of characters into a corresponding sequence of *tokens*
  - Group characters into lexemes (sequences of characters that go together)
  - Determine token lexeme corresponds to
- Deciding how to break the characters into groups is based on the language
  - *"An identifier is any letter followed by 0 or more letters or digits"*

Scanners in a Compiler

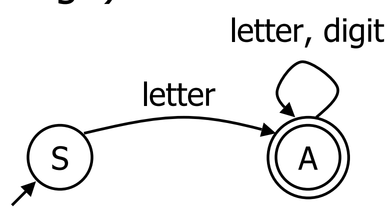
- Scanners are generally called by the parser (supply the next token from the file)
- Written either from scratch or using a scanner generator:
  - lex or flex (C)
  - Jlex (Java)
- Scanner generators:
  - Generally take regular expressions as input
  - Produce a finite state machine (FSM) implementation as output

Generating a Scanner

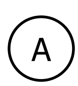


Finite State Machine (FSM)

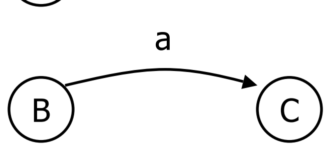
- A *finite state machine* (or finite auotmaton) recognizes *legal* strings from a language
- Example: identifiers (letter followed by letter or digit)



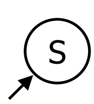
FSM Components



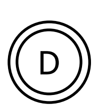
State



Transition (from state B to state C on input “a”)



Start state



A final, halt or accepting state

String Processing with a FSM

- Set the current state to the start state
- While there is still more input
  - Look for a transition from the current state based on the current input character
    - Set the current state to the resulting state from the transition
    - If no transition stop (reject the string)
- Accept the string if the current state is a final state (reject the string otherwise)

- Q: what if there is more than one transition?

Example FSM

- A number consists of one or more digits with an optional sign (+ or -) plus an optional decimal point

Formal Definition of a FSM

- A finite automaton is a 5-tuple (Σ, Q, Δ, s, F) where:
  - An input alphabet Σ
  - A set of states Q
  - A start state s
  - A set of accepting states F $\subseteq$ Q
  - Δ is the state transition function: Q x Σ → Q (i.e., encodes transitions state $\rightarrow^{input}$ state)

Types of Finite State Machines

- Deterministic (also called DFAs for Deterministic Finite Automata)
  - 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), which stands for the empty string (some use λ instead)
    - The FSM can follow an ε edge without considering the current input character

Why Use NFAs?

- Often simpler than DFA
- Easier to string together expressions that cover different types of strings
- Processing in an NFA
  - Current states represents the *set* of possible current states
  - An NFA accepts a string if there is a sequence of moves starting in the start state that consumes the entire string and leaves the machine in a final state

The Language of an FSM

- The language defined by a FSM is the set of strings accepted by FSM.
- For FSM M we write L(M) for the language defined by M.
- Two FSMs M and N are equivalent if L(M) = L(N)
- Theorem: for every NFA M, there exists an equivalent DFA A.