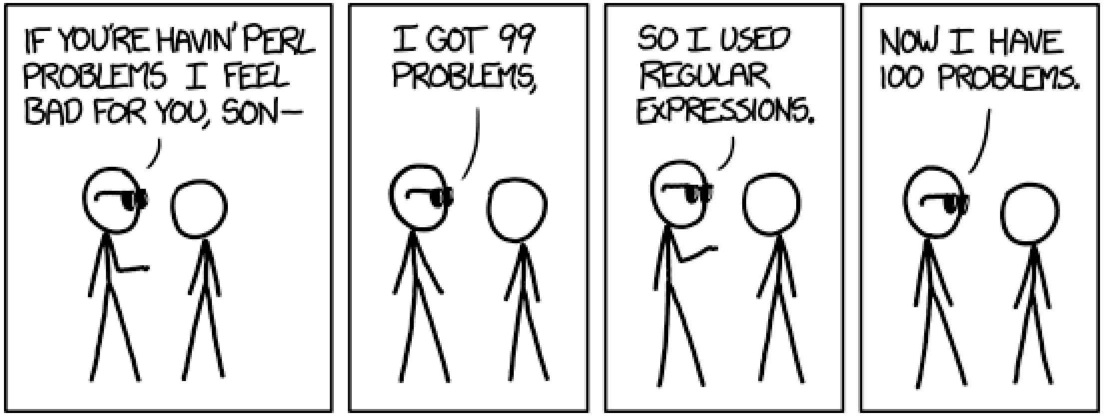
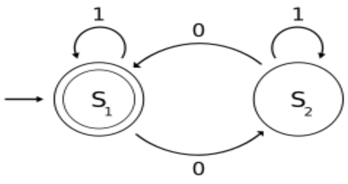
CS 432
Fall 2016

Mike Lam, Professor

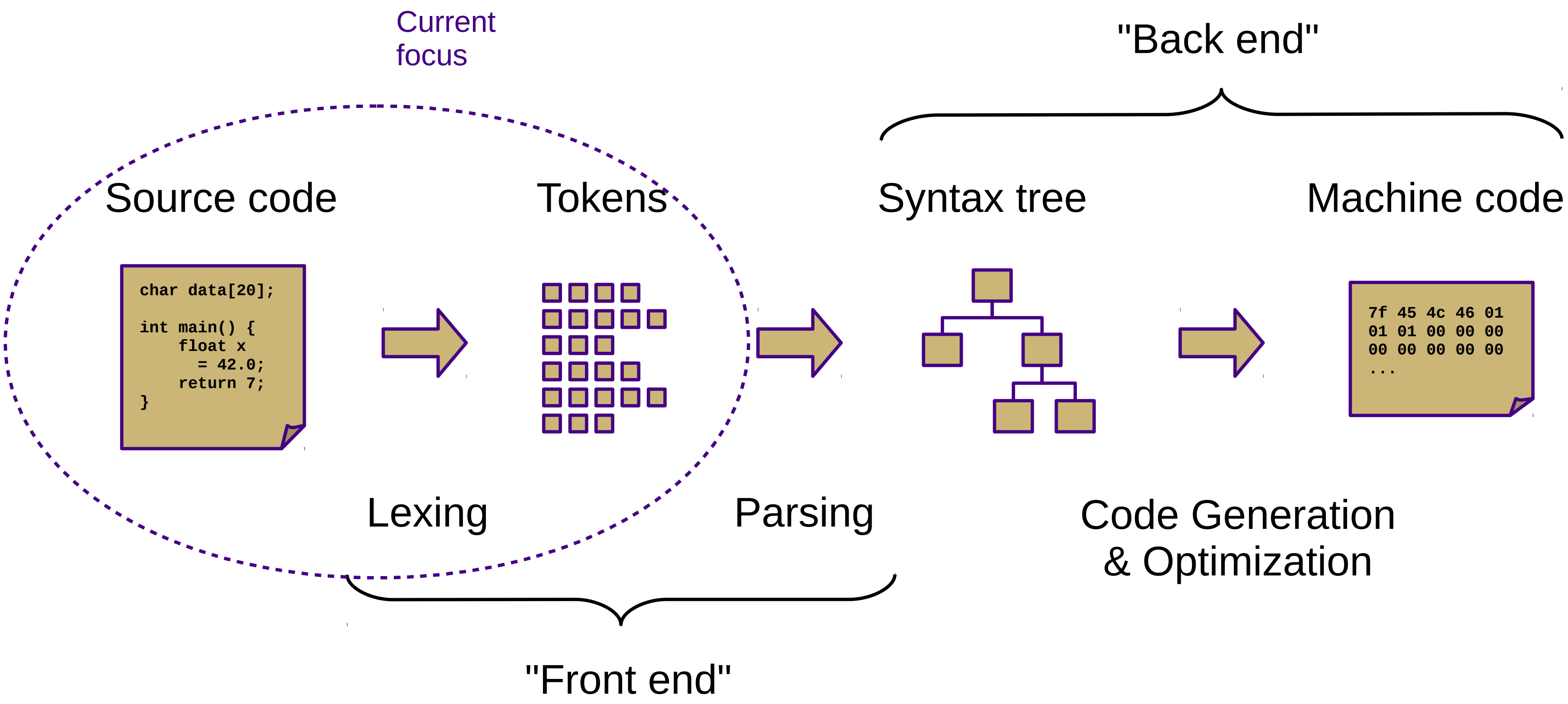


a|(bc)*



Regular Expressions
and
Finite Automata

Compilation



Lexical Analysis

- *Lexemes* or *tokens*: the smallest building blocks of a language's syntax
- *Lexing* or *scanning*: the process of separating a character stream into tokens

```
total = sum(vals) / n

total       identifier
=           equals_op
sum         identifier
(           left_paren
vals        identifier
)           right_paren
/           divide_op
n           identifier
```

```
char *str = "hi";

char        keyword
*           star_op
str         identifier
=           equals_op
"hi"        str_literal
;           semicolon
```

Discussion question

- What is a language?

Language

- A language is a (potentially infinite) set of strings over a finite alphabet

Discussion question

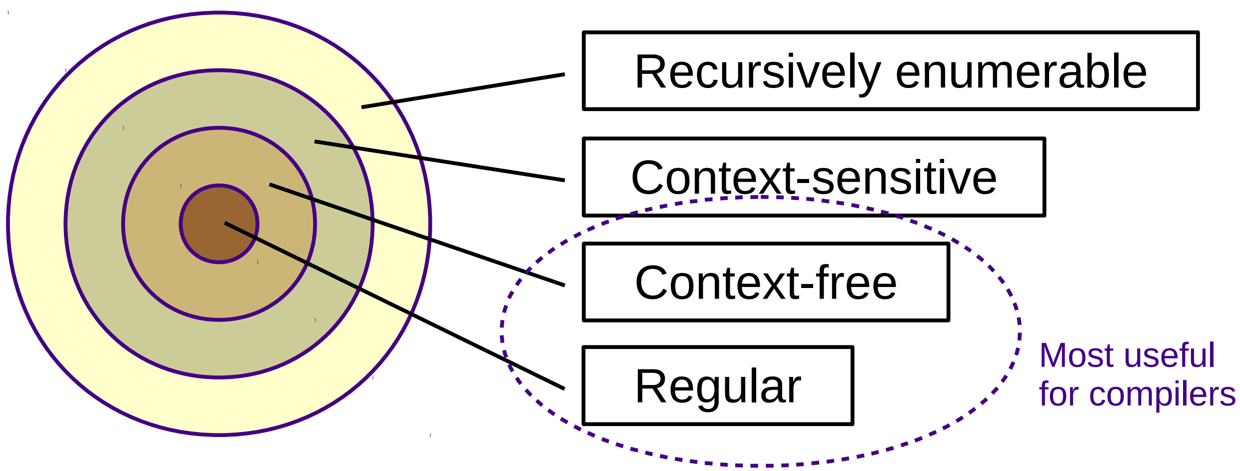
- How do we describe languages?

Language description

- Ways to describe languages
  - Ad-hoc prose
    - “A single ‘x’ followed by one or two ‘y’s followed by any number of ‘z’s”
  - Formal regular expressions (current focus)
    - x(y|yy)z*
  - Formal grammars
    - A → x B C
    - B → y | y y
    - C → z C | ε

Languages

Chomsky Hierarchy of Languages



- Alphabet:
  - Σ = { set of all characters }
- Language:
  - L = { set of sequences of characters from Σ }

Regular expressions

- Describe regular languages
  - Can be thought of as generalized search patterns
- Three basic operations
  - Alternation: **a|b**
  - Concatenation: **ab**
  - ("Kleene") Closure: **a***
- Extended constructs
  - Character sets: **[a-z]** or **[0-9]**
  - Grouping: **(a|b)c**
  - Positive closure: **a+**
    - **a+** == **aa***

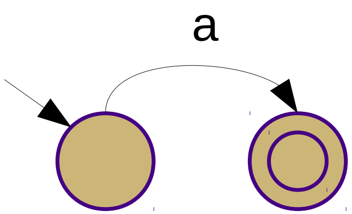
Discussion question

- How would you implement regular expressions?
  - Given a regular expression and a string, how would you tell whether the string belongs to the language described by the regular expression?

Lexical Analysis

- Performed automatically by state machines (*finite state automata)*
  - Set of states with a single *start state*
  - Transitions between states on inputs (+ implicit *dead states*)
  - Some states are *final* or *accepting*
- Deterministic vs. non-deterministic
  - Non-deterministic: multiple possible states for given sentence
  - One edge from each state per character (deterministic)
  - Multiple edges from each state per character (non-deterministic)
  - Empty or ε-transitions (non-deterministic)

Regex: a



Deterministic finite automata

- Formal definition
  - S: set of states
  - Σ: alphabet (set of characters)
  - δ: transition function: (S, Σ) → S
  - $s_{0:}$ start state
  - S_A: accepting/final states
- Acceptance algorithm
  - $s := s_0$
  - ***for each input*** *c:*
  - $\quad s := \delta(s,c)$
  - ***return*** $(s \in S_A)$

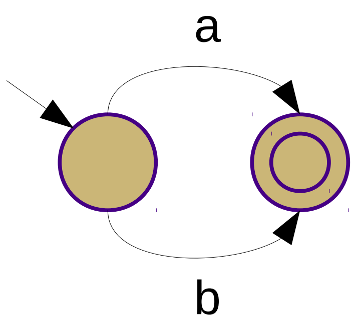
Non-deterministic finite automata

- Formal definition
  - DFA w/ multiple paths and ε-transitions
  - δ: (S, (Σ ∪ {ε})) -> [S]
  - ε-closure(s): all states reachable from s via ε-transitions
- Acceptance algorithm
  - *T := ε-closure(s_0)*
  - ***for each input** c:*
  - *N := {}*
  - ***for each** s **in** T:*
  - *N := N ∪ ε-closure(δ(s,c))*
  - *T = N*
  - ***return** |T ∩ S_A| > 0*

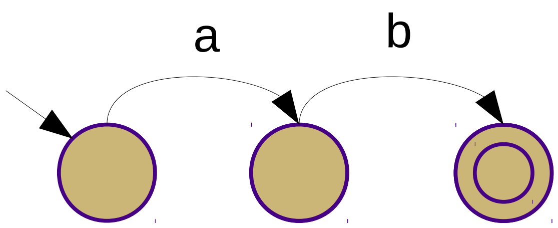
Lexical Analysis

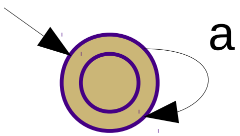
- Examples:

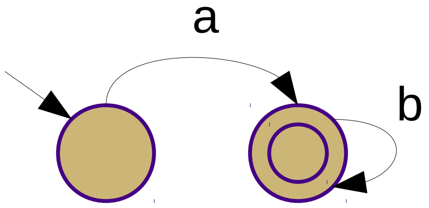
a|b



ab



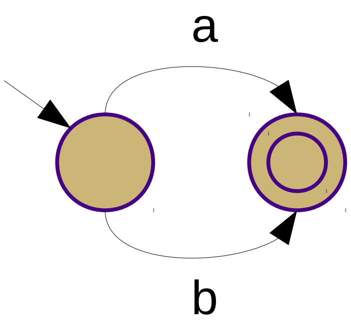




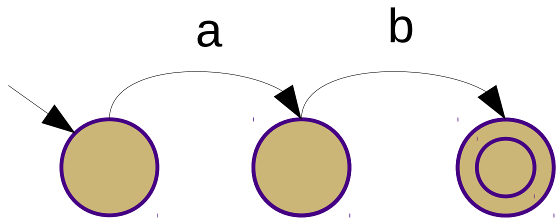
Lexical Analysis

- Examples:

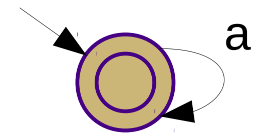
a|b



ab

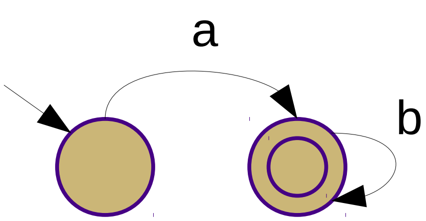


a*



aa*|b

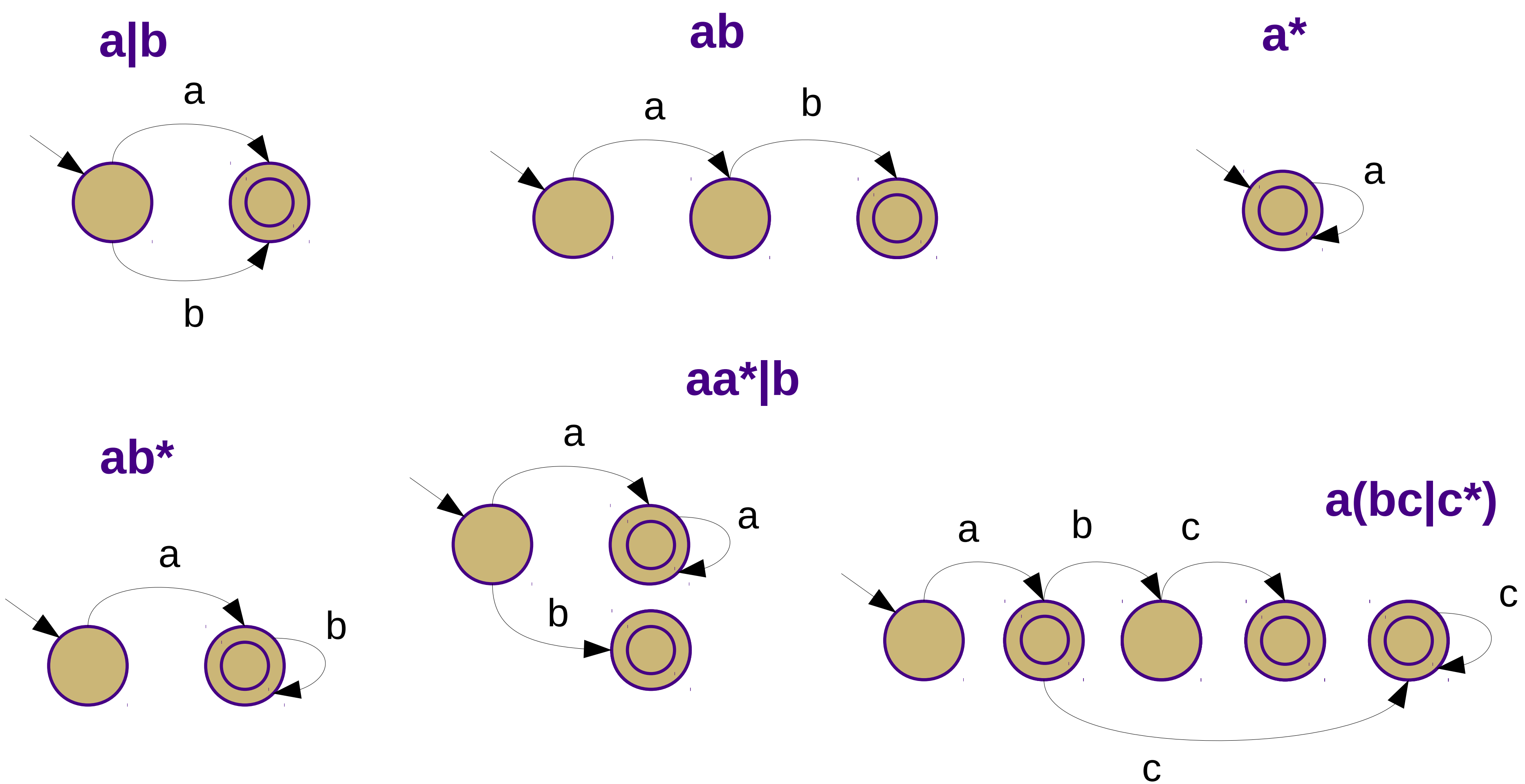
ab*



a(bc|c*)

Lexical Analysis

- Examples:



Equivalence

- Regular expressions, NFAs, and DFAs all describe the same set of languages
  - "Regular languages" from Chomsky hierarchy
- Next week, we will learn how to convert between them

Application

- PA2: Use Java regular expressions to tokenize Decaf files
  - Process the input one line at a time
  - Generally: one regex per token type
  - Each regex begins with “^” (only match from beginning)
  - Prioritize regexes and try each of them in turn
  - When you find a match, extract the matching text
  - Repeat until no match is found or input is consumed
  - Less efficient than an auto-generated lexer
  - However, it is simpler to understand
  - (Our approach to PA3 will be similar)

Activity

- Construct state machines for the following regular expressions:

x*yz*　　**1(1|0)***　　**1(10)***　　**(a|b|c)(ab|bc)**

(dd*.d*)|(d*.dd*)　← ε-transitions may make this one slightly easier