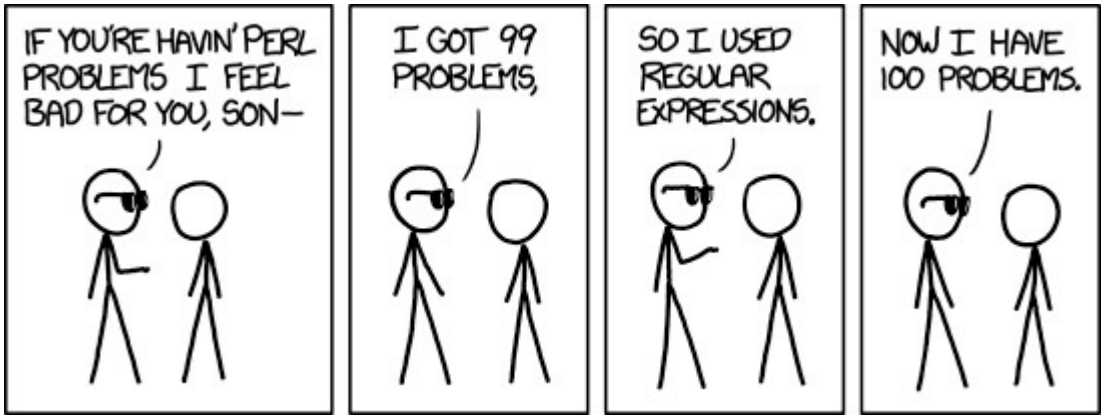
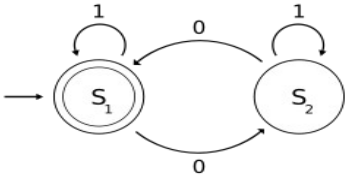
CS 480
Fall 2015

Mike Lam, Professor

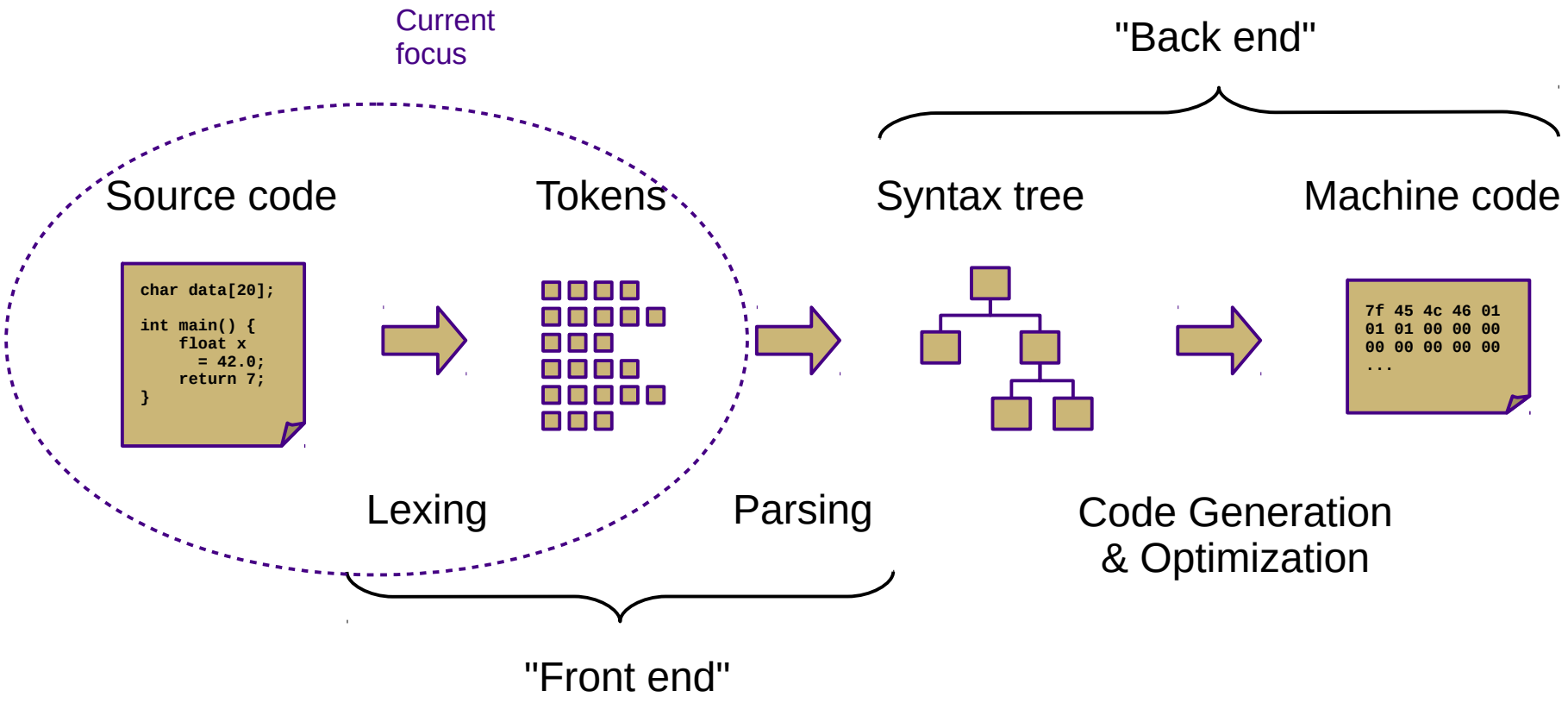


a|(bc)*



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

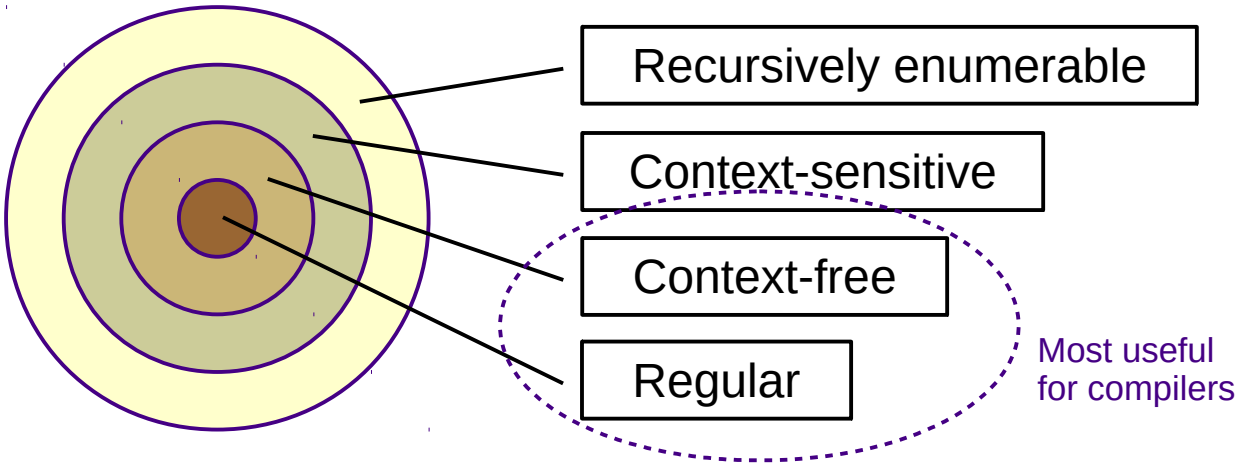
- How do we describe languages?

Language description

- Ways to describe languages
  - Ad-hoc prose
  - Formal regular expressions (current focus)
  - Formal grammars

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*b**
- Extended constructs
  - Character sets: **[a-z]** or **[0-9]**
  - Grouping: **(a|b)c**
  - Positive closure: **a+b**
    - **a+** == **aa***

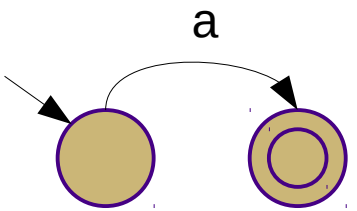
Discussion question

- 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: $(Q, \Sigma) \rightarrow Q$
  - $s_{0:}$ start state
  - S_A: accepting/final states
- Acceptance algorithm
  - $s := s_0$
  - for each input c:
  - $s := \delta(s,c)$
  - return $(s \in S_A)$

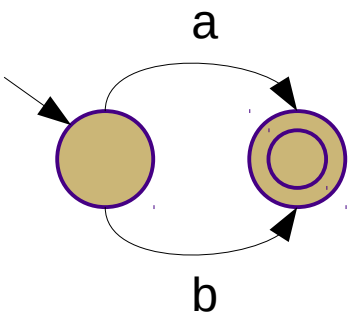
Non-deterministic finite automata

- Formal definition
  - DFA w/ multiple paths and ε-transitions
  - δ: $(Q, (\Sigma \cup \{\varepsilon\})) \rightarrow [Q]$
  - ε-closure(s): all states reachable from s via ε-transitions
- Acceptance algorithm
  - S := ε-closure(s_0)
  - for each input c:
  - $T := \{\}$
  - for each s in S:
  - $T := T \cup$ ε-closure($\delta(s,c)$)
  - $S = T$
  - return $|S \cap S_A| > 0$

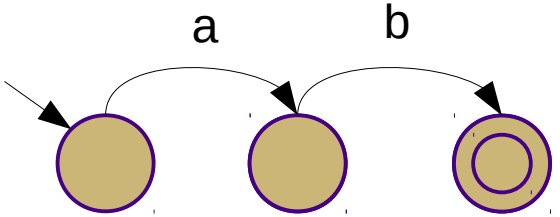
Lexical Analysis

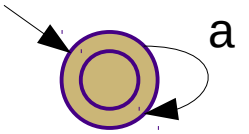
- Examples:

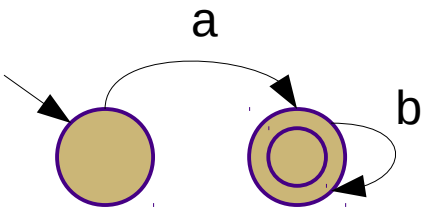
a|b



ab



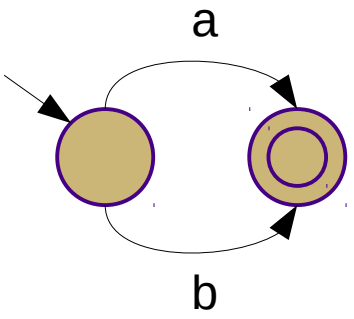




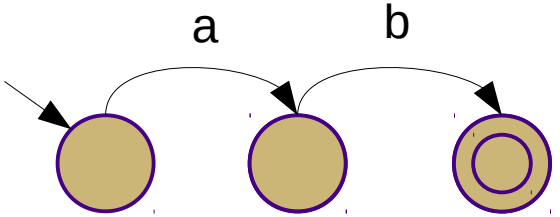
Lexical Analysis

- Examples:

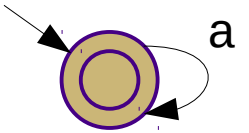
a|b



ab

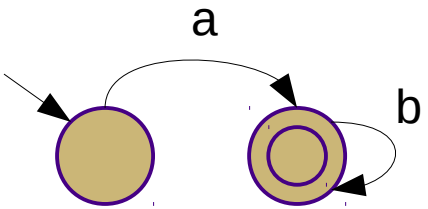


a*



aa*|b

ab*

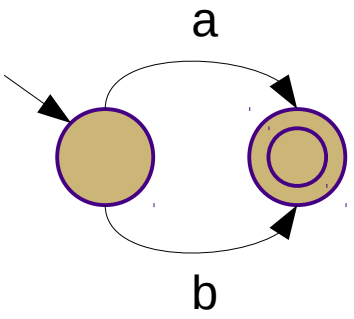


a(bc|c*)

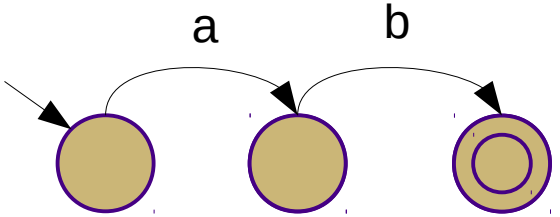
Lexical Analysis

- Examples:

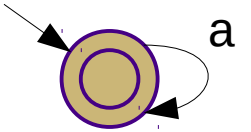
a|b



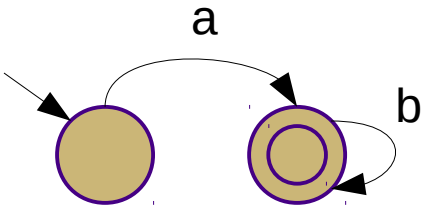
ab



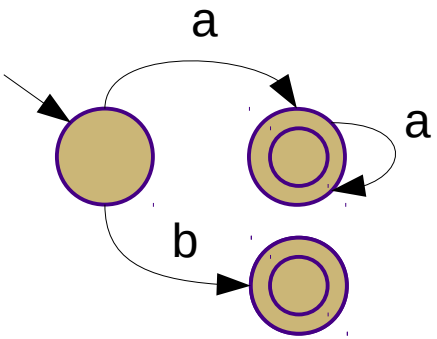
a*



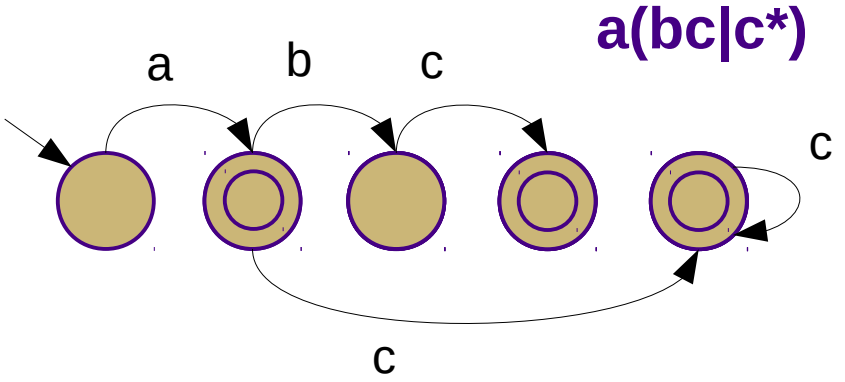
ab*



aa*|b



a(bc|c*)



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

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