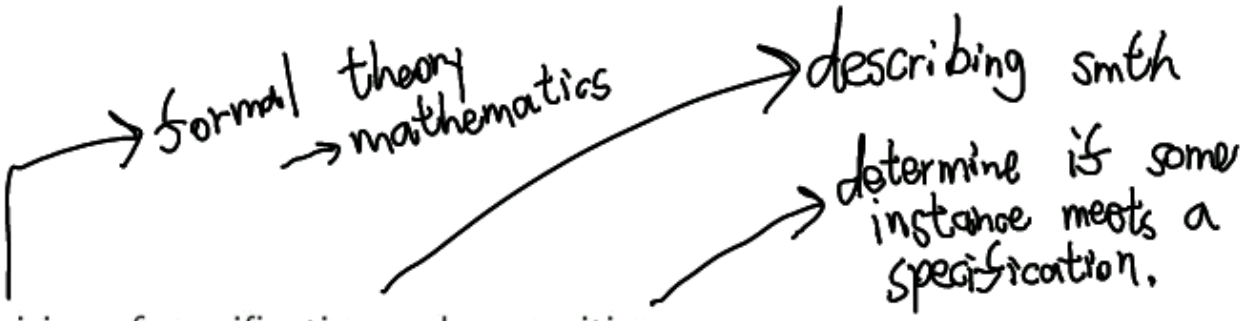
Lecture 9

Introduction to Formal Languages

CS 241: Foundations of Sequential Programs
Fall 2014

Troy Vasiga et al
University of Waterloo

Motivation



- precision of specification and recognition
- importance of this: every assignment from this point forward
- benefits of theory:
  - means of communication
  - determine to power and limits of our communication
  - guides us in practice
    - software we make

Terminology

- rooted in set theory ← [ ] ← contains unique objects
- alphabet: a finite set of symbols

$\Sigma_1 = \{a, b, c\}$ $\Sigma_2 = \{\square, \blacktriangle, \text{  }, 173, blue\}$

↑ Sigma

- word (aka string, sentence): finite sequence of symbols from the alphabet

word w_1 over Σ_1 : $w_1 = baab$

w_2 over Σ_1 : $w_2 = ccc$

w_3 over Σ_1 : $w_3 = \varepsilon$ ← the empty word

↑ epsilon

Terminology (continued)

- language: set of words

↑ possibly infinite

L_1 over Σ_1: $L_1 = \{ba, baa, baaa\}$

L_2 over Σ_1: $L_2 = \{a, aa, aaa, \ldots\}$

- $|W|$: the size of W (where W is a word or language)

$|L_1| = 3$ $\quad$ $|L_2| = \infty$

$|w_1| = |baab| = 4$

$|\varepsilon| = 0$

Uses of Formal Languages: Specification

A statement of what is in the language

Ideally, we have 3 goals of a specification:

mandatory → (1) precise

(2) easy-to-use

(3) automatable

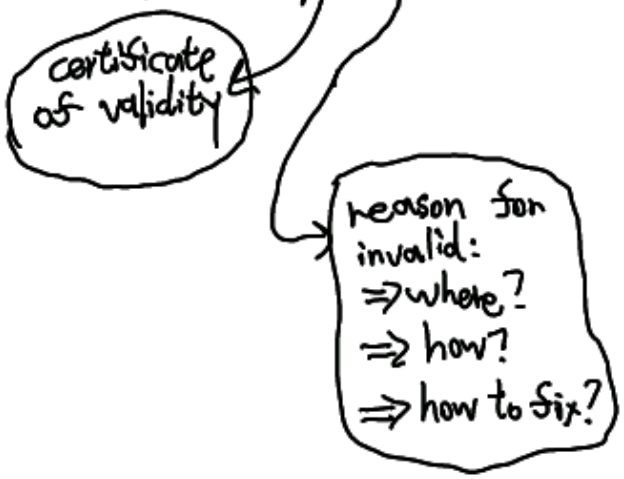
Uses of Formal Languages: Recognition

determine if a given word meets the specification.

Formally: Given a language L and a word w over alphabet Σ, is $w \in L$?

↳ w in L

In practice: We want more than just "yes/no"



WLP4

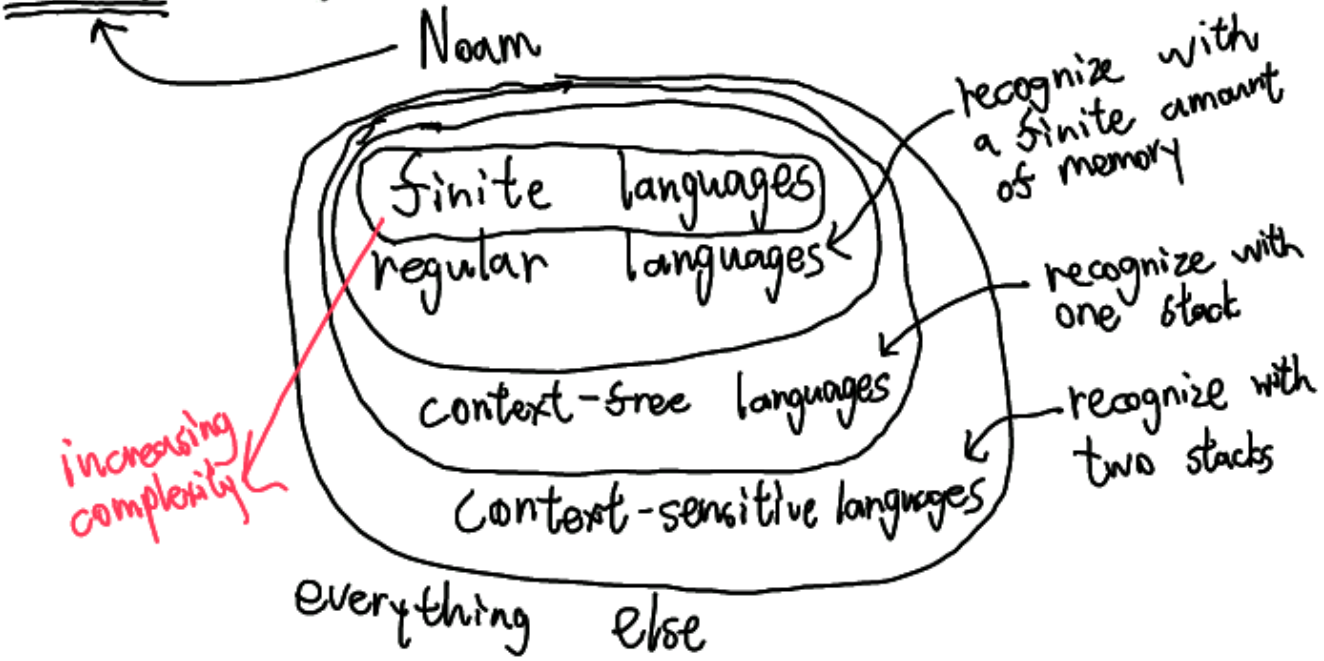
Let's try to formally specify WLP4.

Waterloo Language Plus Pointers Plus Procedures

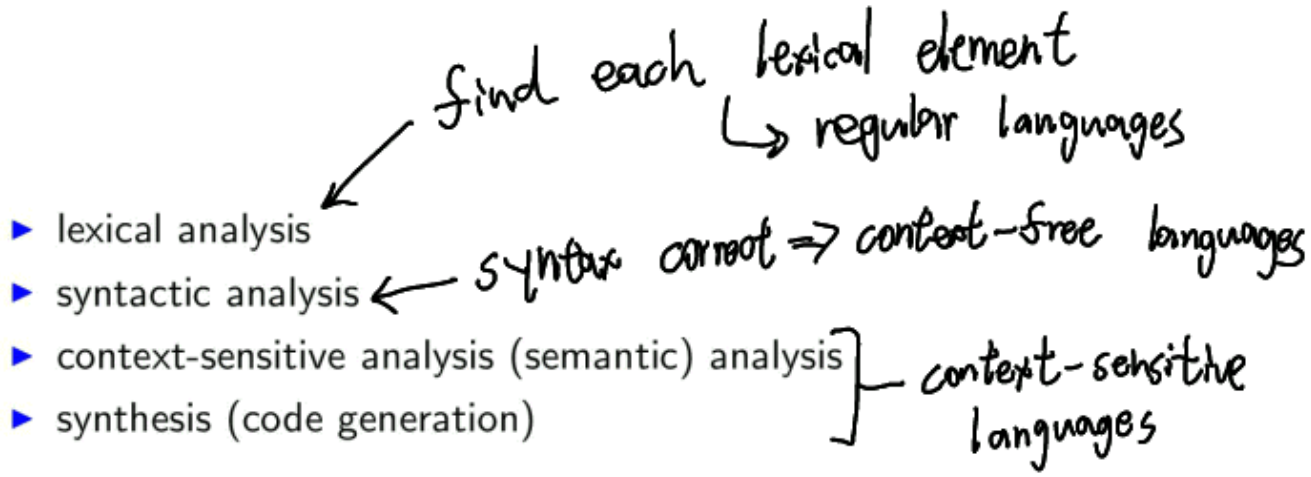
- alphabet:
  Valid WLP4 tokens
  <IF>, <WHILE>, ...
- word:
  sequence of WLP4 tokens
  $\Rightarrow$ WLP4 program
- language:
  {valid WLP4 programs}

Language Classes

- a set of languages may share common characteristics
- Chomsky Hierarchy



Uses of Formal Languages: Organization of Compilation



- lexical analysis
- syntactic analysis
- context-sensitive analysis (semantic) analysis
- synthesis (code generation)

Which language level?

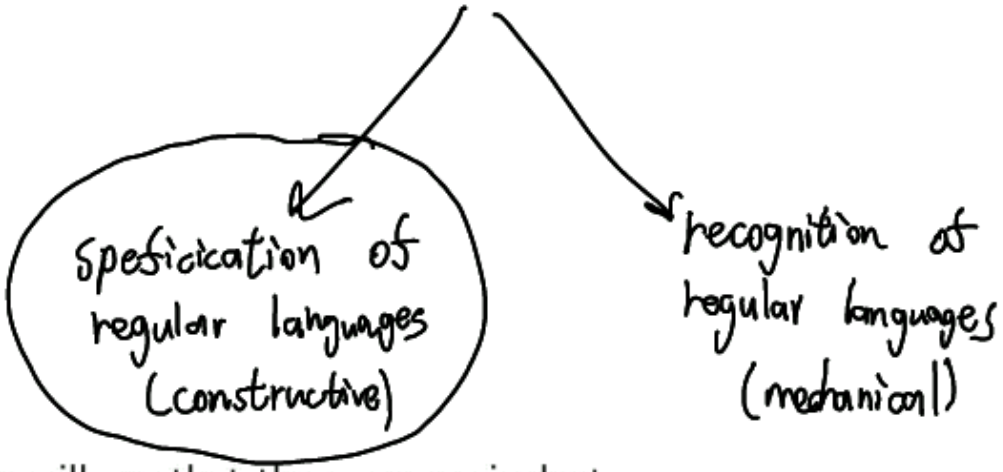
Let Σ ={ASCII characters} - {CR}.

- L_1 = all valid assembly language programs
  - regular: binary
  - context-sensitive: assembly
- L_2 = {\$0, \$1, \$2, ,,, \$31 }
  ⇒ finite language
- L_3 = valid labels in MIPS
  ⇒ regular language
- L_4 = valid load word (lw) offsets
  ⇒ finite

  lw \$3, ____ (\$7)
  ↑ 16-bit integer
- L_5 = valid line of AL for A3P3
  .word int ← 4 billion
  .word hexint ← 4 billion
  ⇒ finite language
- L_6 = valid line of AL for A3P4
  label: .word blat ⇒ context-sensitive language

Regular Languages

- Defining regular languages: two approaches



- We will see that these are equivalent

Specifying Regular Languages

Basic building blocks

1. finite languages
2. union
3. concatenation
4. repetition

Union

- Definition: If T_1 & T_2 are regular languages, then
  $T_1 \cup T_2 = \{x : x \in T_1 \text{ or } x \in T_2\}$
  is a regular language.

- Examples:

  $T_1 = \{dog, cat\}$

  $T_2 = \{red, blue, cat\}$

  $T_1 \cup T_2 = \{dog, cat, red, blue\}$

Concatenation

- Definition: If T_1 & T_2 are regular languages, then

$$T_1 \cdot T_2 = \{ xy : x \in T_1 \text{ and } y \in T_2 \}$$

is regular.

- Examples:

$T_1 = \{dog, cat\}$

$T_2 = \{fish, \varepsilon\}$

$w \cdot \varepsilon = w$

$T_1 \cdot T_2 = \{dogfish, catfish, cat, dog\}$

Repetition

- Definition: If T is a reg. lang., then
  $T^* = \{\varepsilon\} \cup \{xy : x \in T, y \in T^*\}$
  is regular.
- Alternate Definition:
  $T^0 = \{\varepsilon\}$ $T^1 = T$ $T^2 = T \cdot T = \{xy : x \in T, y \in T\}$
  $T^3 = T \cdot T^2, \ldots,$ $\boxed{T^* = \bigcup_{k=0}^{\infty} T^k}$
- Examples:

$*$ ← Kleene Closure

$T = \{man, dog\}$

$T^* = \{\varepsilon, man, dog, mandog, dogman, dogmandog, dogdog, manman, \ldots\}$

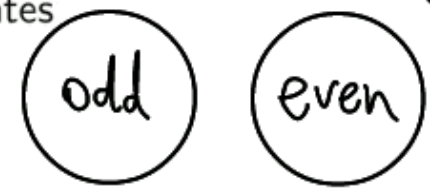
Recognizers: Finite Automata

machines

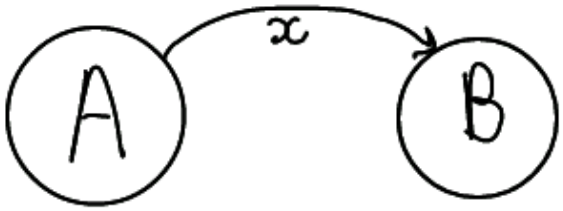
Regular languages can be recognized by *finite automata*.
We begin with *deterministic finite automata*, also called DFAs.

number of states

- states

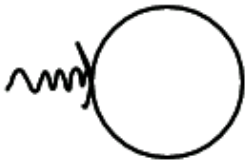


- transitions

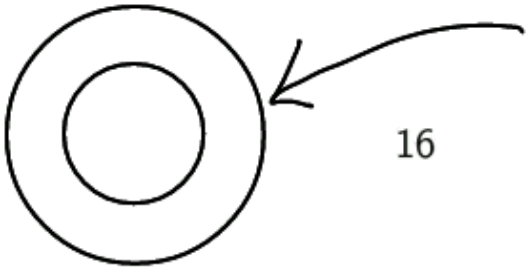


input x moves from state A to state B

- start state



- final states



where I want to be at the end

Finite Automata Example 1

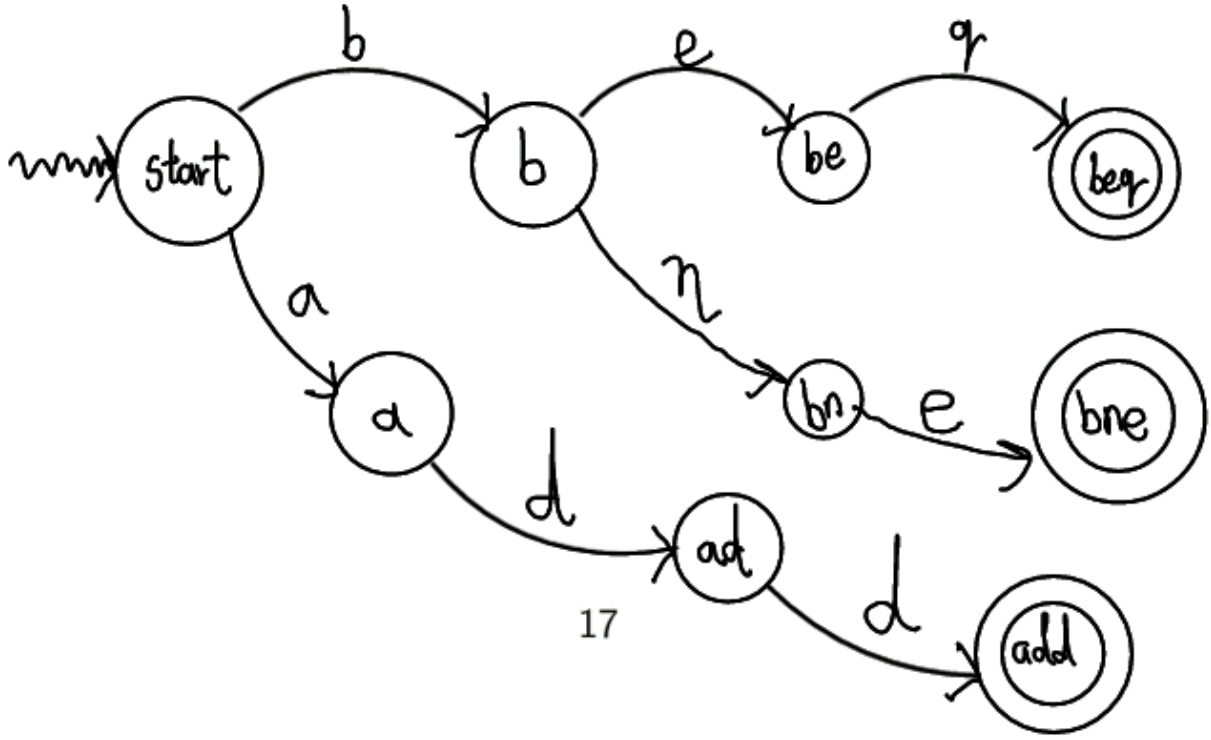
Example: selected opcodes from MIPS assembly language, where alphabet is the ASCII characters.

$L = \{beq, bne, add\}$

ACCEPT: beq, bne, add

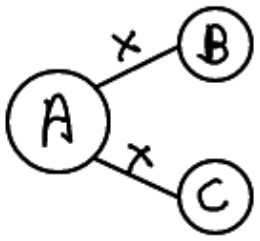
Reject: bn, beqz

M:



Observations About Finite Automata

- ability to trace
  ↳ Remember which state I am in.
- transitions out of a state are unique
  ↳ deterministic

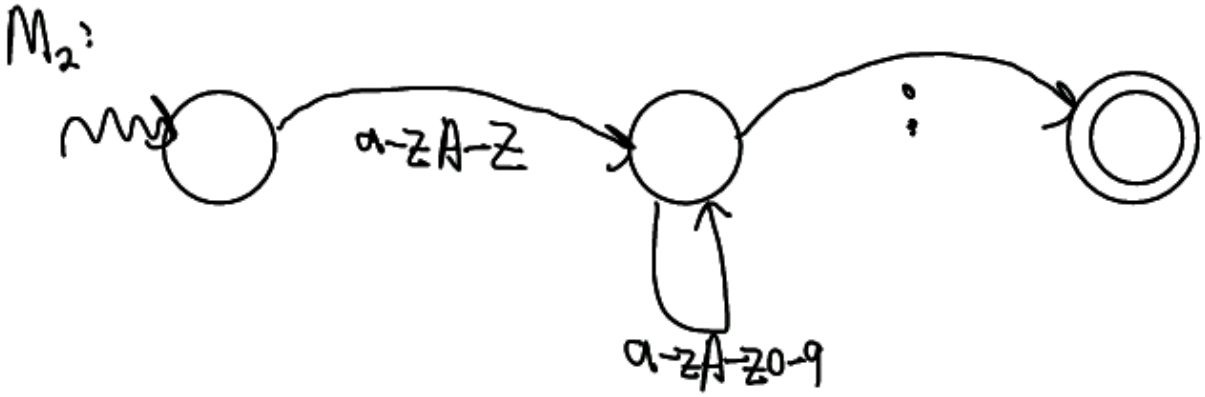


Non-deterministic

- errors
  ↳ implicit error state ⇒ non-accepting, never leave it.
- size of this language
  ∞
- DFA M and language L(M)
  ↑ language accepted by M
  L(M) = {beq, bne, add}

Finite Automata Example 2

Example: MIPS labels, where the alphabet is ASCII characters.

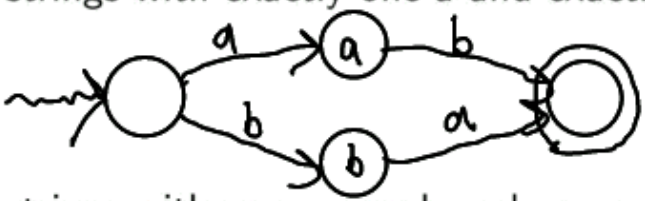


$$|L(M_2)| = \infty$$

More Finite Automata Examples

Let $\Sigma = \{a, b, c\}$.

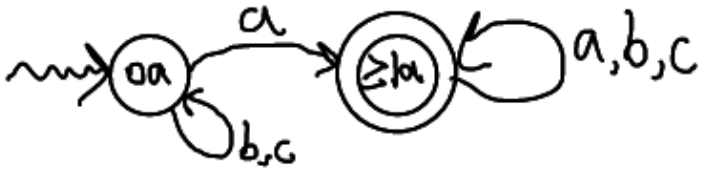
- strings with exactly one a and exactly one b and no c's



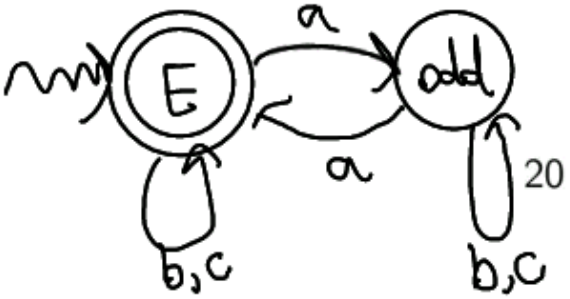
- strings with one a, one b and one c

ex:

- strings with at least one a



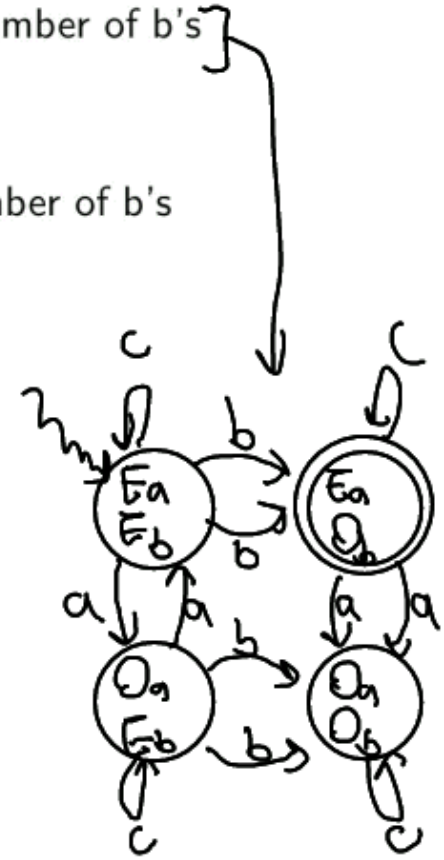
- string with an even number of a's



More Finite Automata Examples

Let $\Sigma = \{a, b, c\}$.

- strings with an even number a's and odd number of b's
- strings with an even number a's or odd number of b's
- valid MIPS identifiers
- valid register designations in MIPS
- valid lines of MIPS assembly language



Killer app for Finite Automata

Scanner: see `asm.*`