Let's get started

We want you to be successful.

We will work together to build an environment in CSE 105 that supports your learning in a way that respects your perspectives, experiences, and identities (including race, ethnicity, heritage, gender, sex, class, sexuality, religion, ability, age, educational background, etc.). Our goal is for you to engage with interesting and challenging concepts and feel comfortable exploring, asking questions, and thriving.

If you or someone you know is suffering from food and/or housing insecurities there are UCSD resources here to help:

Basic Needs Office: https://basicneeds.ucsd.edu/

Triton Food Pantry (in the old Student Center) is free and anonymous, and includes produce:

https://www.facebook.com/tritonfoodpantry/

Mutual Aid UCSD: https://mutualaiducsd.wordpress.com/

Financial aid resources, the possibility of emergency grant funding, and off-campus housing referral resources are available: see your College Dean of Student Affairs.

If you find yourself in an uncomfortable situation, ask for help. We are committed to upholding University policies regarding nondiscrimination, sexual violence and sexual harassment. Here are some campus contacts that could provide this help: Counseling and Psychological Services (CAPS) at 858 534-3755 or http://caps.ucsd.edu; OPHD at 858 534-8298 or ophd@ucsd.edu , http://ophd.ucsd.edu; CARE at Sexual Assault Resource Center at 858 534-5793 or sarc@ucsd.edu , http://care.ucsd.edu.

Please reach out (minnes@ucsd.edu) if you need support with extenuating circumstances affecting CSE 105.

Introductions

Class website on Canvas https://canvas.ucsd.edu

Instructor: Prof. Mia Minnes "Minnes" rhymes with Guinness, minnes@ucsd.edu, http://cseweb.ucsd.edu/ minnes

Our team: One instructor + two TAs and eight tutors + all of you

Fill in contact info for students around you, if you'd like:

CC BY-NC-SA 2.0 Version September 25, 2024 (1)

Welcome to CSE 105: Introduction to Theory of Computation in Fall 2024!

CSE 105's Big Questions

- What problems are computers capable of solving?
- What resources are needed to solve a problem?
- Are some problems harder than others?

In this context, a **problem** is defined as: "Making a decision or computing a value based on some input"

Consider the following problems:

- Find a file on your computer
- Determine if your code will compile
- Find a run-time error in your code
- Certify that your system is un-hackable

file system organized, name.

Which of these is hardest?

Definition of problem
- what are we starting from?

Approach to solution
- eg. finding bugs might require more than running test case.

* Running solution

In Computer Science, we operationalize "hardest" as "requires most resources", where resources might be memory, time, parallelism, randomness, power, etc.

To be able to compare "hardness" of problems, we use a consistent description of problems

Input: String

Output: Yes/ No, where Yes means that the input string matches the pattern or property described by the problem.

CC BY-NC-SA 2.0 Version September 25, 2024 (2)

Weeks 0 and 1 at a glance

Textbook reading: Chapter 0, Sections 1.3, 1.1

Before Monday, review class syllabus on Canvas (https://canvas.ucsd.edu/) and read Example 1.51 and Definition 1.52 (definition of regular expressions) on page 64. *Notice*: we are jumping to Section 1.3 and then will come back to Section 1.1 on Wednesday.

Before Wednesday, read Figure 1.4 and Definition 1.5 (definition of finite automata) on pages 34-35.

Before Friday, read the definition of the union, concatenation, and star operations for languages is given as Definition 1.23 on page 44 and a useful example is Example 1.24.

For Week 2 Monday: Pages 41-43 (Figures 1.18, 1.19, 1.20) (examples of automata and languages).

Textbook references: Within a chapter, each item is numbered consecutively. Figure 1.22 is the twenty-second numbered item in chapter one; it comes right after Example 1.21 and right before Definition 1.23.

We will be learning and practicing to:

- Clearly and unambiguously communicate computational ideas using appropriate formalism. Translate across levels of abstraction.
  - Translate a decision problem to a set of strings coding the problem.
    - **Distinguish between alphabet, language, sets, and strings**
  - Use regular expressions and relate them to languages and automata.
    - **Write and debug regular expressions using correct syntax**
    - **Determine if a given string is in the language described by a regular expression**
  - Use precise notation to formally define the state diagram of finite automata.
  - Use clear English to describe computations of finite automata TM informally.
    - **State the formal definition of (deterministic) finite automata**
    - **Trace the computation of a finite automaton on a given string using its state diagram**
    - **Translate between a state diagram and a formal definition**
    - **Determine if a given string is in the language recognized by a finite automaton**

TODO:

#FinAid Assignment on Canvas (complete as soon as possible) and read syllabus on Canvas

Schedule your Test 1 Attempt 1, Test 2 Attempt 1, Test 1 Attempt 2, and Test 2 Attempt 2 times at PrairieTest (http://us.prairietest.com)

Review Quiz on PrairieLearn (http://us.prairielearn.com), complete by Sunday 10/6/2024

Create a homework group, possibly by using the Piazza (https://piazza.com/) find-a-teammate tool

Homework 1 submitted via Gradescope (https://www.gradescope.com/), due Tuesday 10/8/2024

CC BY-NC-SA 2.0 Version September 25, 2024 (3)

Week 0 Friday: Terminology and Notation

The CSE 105 vocabulary and notation build on discrete math and introduction to proofs classes. Some of the conventions may be a bit different from what you saw before so we'll draw your attention to them.

For consistency, we will use the notation from this class' textbook[1].

These definitions are on pages 3, 4, 6, 13, 14, 53.

Term	**Typical symbol** or **Notation**	**Meaning**
	Sigma Gamma	
Alphabet	Σ, Γ	A non-empty finite set
Symbol over Σ	σ, b, x	An element of the alphabet Σ
String over Σ	u, v, w	A finite list of symbols from Σ
(The) empty string	ε	The (only) string of length 0
The set of all strings over Σ	Σ^*	The collection of all possible strings formed from symbols from Σ
(Some) language over Σ	L	(Some) set of strings over Σ
(The) empty language	$\emptyset$	The empty set, i.e. the set that has no strings (and no other elements either)
The power set of a set X	$\mathcal{P}(X)$	The set of all subsets of X
(The set of) natural numbers	$\mathcal{N}$	The set of positive integers
(Some) finite set		The empty set or a set whose distinct elements can be counted by a natural number
(Some) infinite set		A set that is not finite.
Reverse of a string w	$w^{\mathcal{R}}$	write w in the opposite order, if $w = w_1 \cdots w_n$ then $w^{\mathcal{R}} = w_n \cdots w_1$. Note: $\varepsilon^{\mathcal{R}} = \varepsilon$
Concatenating strings x and y	xy	take $x = x_1 \cdots x_m$, $y = y_1 \cdots y_n$ and form $xy = x_1 \cdots x_m y_1 \cdots y_n$
String z is a substring of string w		there are strings u, v such that $w = uzv$
String x is a prefix of string y		there is a string z such that $y = xz$
String x is a proper prefix of string y		x is a prefix of y and $x \neq y$
Shortlex order, also known as string order over alphabet Σ		Order strings over Σ first by length and then according to the dictionary order, assuming symbols in Σ have an ordering

[1] Page references are to the 3rd edition of Sipser's Introduction to the Theory of Computation, available through various sources for approximately $30. You may be able to opt in to purchase a digital copy through Canvas. Copies of the book are also available for those who can't access the book to borrow from the course instructor, while supplies last (minnes@ucsd.edu)

CC BY-NC-SA 2.0 Version September 25, 2024 (4)

Write out in words the meaning of the symbols below:

$$\{a, b, c\}$$

The set whose three elements are a and b and c

$$|\{a, b, a\}| = 2$$

There are (exactly) 2 distinct elements in the set $\{a, b, a\}$.

$$|aba| = 3$$

The string aba has length 3.

Circle the correct choice:

A **string** over an alphabet Σ is an element of Σ^* ~~OR a subset of Σ^*~~.

A **language** over an alphabet Σ is ~~an element of Σ^* OR~~ a subset of Σ^*.

With $\Sigma_1 = \{0, 1\}$ and $\Sigma_2 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$ and $\Gamma = \{0, 1, x, y, z\}$

~~**True** or~~ **False**: $\varepsilon \in \Sigma_1$

True ~~or **False**~~: ε is a string over Σ_1

~~**True** or~~ **False**: ε is a language over Σ_1

True ~~or **False**~~: ε is a prefix of some string over Σ_1

~~**True** or~~ **False**: There is a string over Σ_1 that is a proper prefix of ε

The first five strings over Σ_1 in string order, using the ordering $0 < 1$:

ε, 0, 1, 00, 01

The first five strings over Σ_2 in string order, using the usual alphabetical ordering for single letters:

ε, a, b, c, d

CC BY-NC-SA 2.0

Week 1 Monday: Regular expressions

Our motivation in studying sets of strings is that they can be used to encode problems. To calibrate how difficult a problem is to solve, we describe how complicated the set of strings that encodes it is. How do we define sets of strings?

Set notation · roster $\{-, -, -, -\}$ finite sets only

Set builder notation $\{ x \mid P(x) \}$ or $\{ f(x) \mid \ldots x \ldots - \}$

The collection of all values for which the property $P(\cdot)$ holds

Recursive definition
- Basis step
- recursive step

Named sets Greek alphabet, $\mathbb{N}$, $\emptyset$, $\mathbb{R}$
subsets of these might be defined with set operations

How would you describe the language that has no elements at all?

↳ set of strings over a given alphabet

$\emptyset$ $\{\}$ $\{ x \in \Sigma^* \mid x \neq x \}$

set of all strings over alphabet Σ

How would you describe the language that has all strings over $\{0, 1\}$ as its elements?

Not roster definitions because infinite

Set operation: Kleene star

for any set A $A^* = \{ w_1 \cdots w_k \mid k \geq 0, w_i \in A \}$

Set of strings over $\{0,1\}$ is $\{0,1\}^*$

$\{ x \in \Sigma^* \mid x = x \}$

Recursively: (Basis step) $\varepsilon \in \{0,1\}^*$

(Recursive step) For $w \in \{0,1\}^*$ $w0 \in \{0,1\}^*$ and $w1 \in \{0,1\}^*$

CC BY-NC-SA 2.0 Version September 25, 2024 (6)

This definition was in the pre-class reading **Definition 1.52**: A **regular expression** over alphabet Σ is a syntactic expression that can describe a language over Σ. The collection of all regular expressions over Σ is defined recursively:

Basis steps of recursive definition

a is a regular expression, for $a \in \Sigma$
ε is a regular expression
$\emptyset$ is a regular expression

Recursive steps of recursive definition

$(R_1 \cup R_2)$ is a regular expression when R_1, R_2 are regular expressions
$(R_1 \circ R_2)$ is a regular expression when R_1, R_2 are regular expressions
(R_1^*) is a regular expression when R_1 is a regular expression

The *semantics* (or meaning) of the syntactic regular expression is the **language described by the regular expression**. The function that assigns a language to a regular expression over Σ is defined recursively, using familiar set operations:

Not a language: $\{\emptyset\}$ — the set that contains the empty set
$\neq$ Language: $\{\varepsilon\}$ — The set that contains the empty string
$\neq$ Language: $\emptyset$ — The set that has no elements

Basis steps of recursive definition

The language described by a, for $a \in \Sigma$, is $\{a\}$ and we write $L(a) = \{a\}$
The language described by ε is $\{\varepsilon\}$ and we write $L(\varepsilon) = \{\varepsilon\}$
The language described by $\emptyset$ is $\{\}$ and we write $L(\emptyset) = \emptyset$.

Recursive steps of recursive definition

When R_1, R_2 are regular expressions, the language described by the regular expression $(R_1 \cup R_2)$ is the union of the languages described by R_1 and R_2, and we write

$$L(\ (R_1 \cup R_2)\ ) = L(R_1) \cup L(R_2) = \{w \mid w \in L(R_1) \vee w \in L(R_2)\}$$

When R_1, R_2 are regular expressions, the language described by the regular expression $(R_1 \circ R_2)$ is the concatenation of the languages described by R_1 and R_2, and we write

$$L(\ (R_1 \circ R_2)\ ) = L(R_1) \circ L(R_2) = \{uv \mid u \in L(R_1) \wedge v \in L(R_2)\}$$

When R_1 is a regular expression, the language described by the regular expression (R_1^*) is the **Kleene star** of the language described by R_1 and we write

$$L(\ (R_1^*)\ ) = (\ L(R_1)\ )^* = \{w_1 \cdots w_k \mid k \geq 0 \text{ and each } w_i \in L(R_1)\}$$

CC BY-NC-SA 2.0

For the following examples assume the alphabet is $\Sigma_1 = \{0, 1\}$:

The language described by the regular expression 0 is $L(0) = \{0\}$

The language described by the regular expression 1 is $L(1) = \{1\}$

The language described by the regular expression ε is $L(\varepsilon) = \{\varepsilon\}$

The language described by the regular expression $\emptyset$ is $L(\emptyset) = \emptyset$

The language described by the regular expression $(\Sigma_1\Sigma_1\Sigma_1)^*$ is $L(\ (\Sigma_1\Sigma_1\Sigma_1)^*\ ) =$

Σ_1 is the regular expression $(0 \cup 1)$

symbol for concatenation

$$L\Big(\ \big((0\cup1)\circ(0\cup1)\circ(0\cup1)\big)^*\ \Big)$$

$$= \Big(L\big((0\cup1)\circ(0\cup1)\circ(0\cup1)\big)\Big)^*$$

$$= \{\text{length 3 strings}\}^*$$

$$= \{\ \underbrace{_\ _\ \cdots\ _}_{k \text{ many slots}} \mid w_i \text{ is a length 3 string}\}.$$

$$= \{\text{strings over } \{0,1\} \text{ whose length is a multiple of 3}\}$$

$L((0\cup1)) =$

$L(0) \cup L(1) =$

$\{0\} \cup \{1\} = \{0,1\}$

$\{0,1\}\cdot\{0,1\}\cdot\{0,1\}$

$= \{uvw \mid u \in \{0,1\} \text{ and } v \in \{0,1\} \text{ and } w \in \{0,1\}\}$

The language described by the regular expression $1^* \circ 1$ is $L(1^* \circ 1) =$

(Added after class)

symbol

$$L(1^*\circ1) = L(1^*)\circ L(1)$$

Symbol, set operation

$$= L(1)^* \circ L(1)$$

set operation

$$= \{1\}^* \circ \{1\}$$

by basis step of semantics of regular expressions

$$= \{1^i \mid i \in \mathbb{Z},\ i \geq 0\} \circ \{1\}$$

$$= \{1^{i+1} \mid i \in \mathbb{Z},\ i \geq 0\}$$

= The set of nonempty strings of all 1s.

: regular expressions

: sets

CC BY-NC-SA 2.0 Version September 25, 2024 (8)

Recall: two sets are equal means agreement on members
i.e. $A=B$ means $\forall x (x \in A \leftrightarrow x \in B)$ i.e. $A \subseteq B$ and $B \subseteq A$.

Week 1 Wednesday: Regular expressions conventions

Review: Determine whether each statement below about regular expressions over the alphabet $\{a, b, c\}$ is true or false:

characters glued together

True or ~~False~~: $ab \in L((a \cup b)^*)$ "ab is an element of the language described by the regular expression $(a \cup b)^*$"

~~True~~ or False: $ba \in L(a^*b^*)$

~~True~~ or False: $\varepsilon \in L(a \cup b \cup c)$

True or ~~False~~: $\varepsilon \in L((a \cup b)^*)$

~~True~~ or False: $\varepsilon \in L(aa^* \cup bb^*)$

$L((a \cup b)^*) = \{a, b\}^* = \{\text{strings over } \{a,b\}\}$ $L((a \cup b)^*) = \{a, b\}^*$

$L(a \cup b \cup c) = \{a\} \cup \{b\} \cup \{c\} = \{a, b, c\}$

$a/b\ a/b \ldots a/b$

$L(a^* \circ b^*) = \{a^i b^j \mid i, j \geq 0\}$

Shorthand and conventions (Sipser pages 63-65)

Assuming Σ is the alphabet, we use the following conventions

Σ	regular expression describing language consisting of all strings of length 1 over Σ
$*$ then $\circ$ then $\cup$	precedence order, unless parentheses are used to change it
R_1R_2	shorthand for $R_1 \circ R_2$ (concatenation symbol is implicit)
R^+	shorthand for $R^* \circ R$
R^k	shorthand for R concatenated with itself k times, where k is a (specific) natural number

$L(R^+) = L(R^*) \circ L(R)$

Σ^* regular expression

$L(\Sigma^*)$ = set of all strings over Σ.

Exercise: $L(R^* \circ R) = L(R \circ R^*)$

Caution: many programming languages that support regular expressions build in functionality that is more powerful than the "pure" definition of regular expressions given here.

Regular expressions are everywhere (once you start looking for them).

Software tools and languages often have built-in support for regular expressions to describe **patterns** that we want to match (e.g. Excel/ Sheets, grep, Perl, python, Java, Ruby).

Under the hood, the first phase of **compilers** is to transform the strings we write in code to tokens (keywords, operators, identifiers, literals). Compilers use regular expressions to describe the sets of strings that can be used for each token type.

Next time: we'll start to see how to build machines that decide whether strings match the pattern described by a regular expression.

CC BY-NC-SA 2.0

Practice with the regular expressions over $\{a, b\}$ below.

For example: Which regular expression(s) below describe a language that includes the string a as an element?

$L(a^*b^*) = \{ a^i b^j \mid i, j \geq 0 \text{ (integers)} \}$

a is in the set, choosing $i=1$ and $j=0$.

$a(ba)^*b$

strings in the language described by this regular expression start with a, end with b, and have some number of copies of ba in between.

Since a doesn't end in b, it is not in this set.

$L(a^* \cup b^*) = \{ a^i \mid i \geq 0 \text{ (integer)} \} \cup \{ b^j \mid j \geq 0 \text{ (integer)} \}$.

a is in the set, because it is in $\{a^i \mid i \geq 0\}$ (and using definition of union)

$L((aaa)^*) = \{ a^{3i} \mid i \geq 0 \text{ (integer)} \}$

a is not in the set because 1 is not an integer multiple of 3

$L((\varepsilon \cup a)b) = L(\varepsilon \cup a) \cdot L(b) = \{\varepsilon b, ab\} = \{b, ab\}$

a is not in this set.

CC BY-NC-SA 2.0 Version September 25, 2024 (10)

Deterministic

Week 1 Friday: Finite automata

This definition was in the pre-class reading A finite automaton (FA) is specified by $M = (Q, \Sigma, \delta, q_0, F)$. This 5-tuple is called the **formal definition** of the FA. The FA can also be represented by its state diagram: with nodes for the state, labelled edges specifying the transition function, and decorations on nodes denoting the start and accept states.

Finite set of states Q can be labelled by any collection of distinct names. Often we use default state labels $q0, q1, \ldots$

The alphabet Σ determines the possible inputs to the automaton. Each input to the automaton is a string over Σ, and the automaton "processes" the input one symbol (or character) at a time.

The transition function δ gives the next state of the automaton based on the current state of the machine and on the next input symbol.

The start state q_0 is an element of Q. Each computation of the machine starts at the start state.

The accept (final) states F form a subset of the states of the automaton, $F \subseteq Q$. These states are used to flag if the machine accepts or rejects an input string.

The computation of a machine on an input string is a sequence of states in the machine, starting with the start state, determined by transitions of the machine as it reads successive input symbols.

The finite automaton M accepts the given input string exactly when the computation of M on the input string ends in an accept state. M rejects the given input string exactly when the computation of M on the input string ends in a nonaccept state, that is, a state that is not in F.

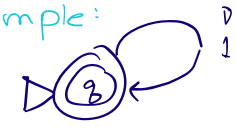
The language of M, $L(M)$, is defined as the set of all strings that are each accepted by the machine M. Each string that is rejected by M is not in $L(M)$. The language of M is also called the language recognized by M.

What is **finite** about all finite automata? (Select all that apply)

- ☑ The size of the machine (number of states, number of arrows)
- ☑ The length of each computation of the machine
- ☐ The number of strings that are accepted by the machine

Counterexample:

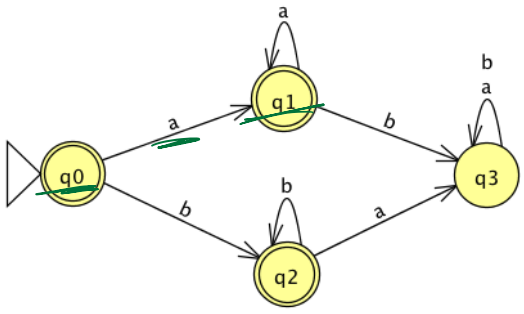
M is



$\Sigma = \{0, 1\}$

$L(M) = \Sigma^*$

CC BY-NC-SA 2.0 Version September 25, 2024 (11)



The formal definition of this FA is $(\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_0, q_1, q_2\})$

where $\delta: \{q_0, q_1, q_2, q_3\} \times \{a, b\} \to \{q_0, q_1, q_2, q_3\}$ is given by

Input	Output
(q_0, a)	q_1
(q_0, b)	
(q_1, a)	
⋮	

	a	b
q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_3	q_2
q_3	q_3	q_3

$L(\ldots) = ?$

Classify each string $a, aa, ab, ba, bb, \varepsilon$ as accepted by the FA or rejected by the FA.

Why are these the only two options?

For each string, there is exactly one computation of the FA on that string and that computation ends in a state. If this state is in F then the machine accepts this string. If this state is not in F then the machine rejects this string.

The language recognized by this automaton is

$L(a^* \cup b^*)$ regex

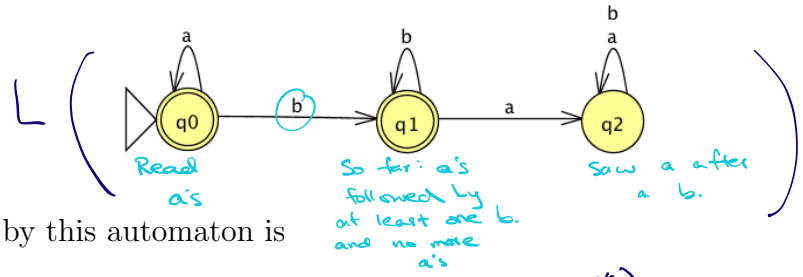
$L(\varepsilon \cup aa^* \cup bb^*)$ regex

$\{a^i \mid i \geq 0, i \text{ integer}\} \cup \{b^j \mid j \geq 0, j \text{ integer}\}$

$a^0 = \varepsilon$

Concatenation of a with itself zero times

CC BY-NC-SA 2.0 Version September 25, 2024 (12)

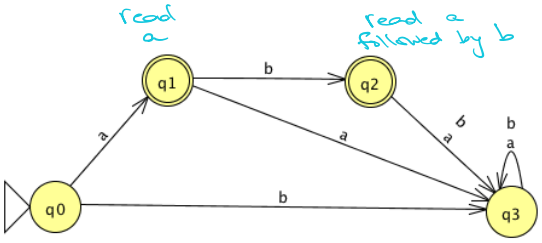


The language recognized by this automaton is

$L(a^* \cup a^*bb^*)$ regex

$= L(a^*b^*)$ regex

$= \{a^i b^j \mid i,j \geq 0 \text{ integers}\}$



The language recognized by this automaton is $\{a, ab\} = L(a \cup ab)$ regex

CC BY-NC-SA 2.0 Version September 25, 2024 (13)