BU CS 332 – Theory of Computation

Lecture 2:

- Parts of a Theory of Computation
- Sets, Strings, and Languages

Reading:

Sipser Ch 0

Mark Bun

January 27, 2021

What makes a good theory?

- General ideas that apply to many different systems
- Expressed simply, abstractly, and precisely

Parts of a Theory of Computation

- Models for machines (computational devices)
- Models for the problems machines can be used to solve
- Theorems about what kinds of machines can solve what kinds of problems, and at what cost

What is a (Computational) Problem?

For us: A problem will be the task of recognizing whether a *string* is in a *language*

- **Alphabet**: A finite set Σ Ex. $\Sigma = \{a, b\}$
- **String:** A finite concatenation of alphabet symbols
  Ex. $bba, ababb$
  ε denotes empty string, length 0
  Σ^* = set of all strings using symbols from Σ
  Ex. $\{a, b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, \dots\}$
- **Language:** A set $L \subseteq \Sigma^*$ of strings

Examples of Languages

Parity: Given a string consisting of a's and b's, does it contain an even number of a's?

$\Sigma = \{a, b\}$ $L = \{x \in \{a,b\}^* \mid x \text{ has an even \# of a's}\}$

Primality: Given a natural number x (represented in binary), is x prime?

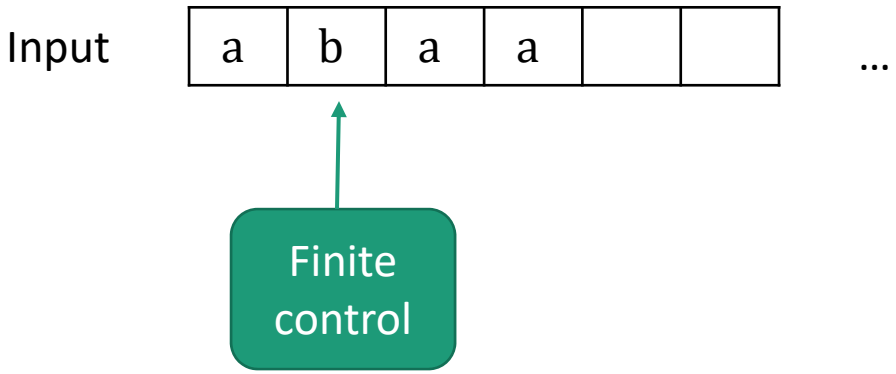
$\Sigma = \{0, 1\}$ $L = \{x \in \{0,1\}^* \mid x \text{ represents a prime}\}$

Halting Problem: Given a C program, can it ever get stuck in an infinite loop?

$\Sigma =$ ASCII $L = \{P \in \Sigma'^* \mid P \text{ never gets stuck in a loop}\}$

Machine Models

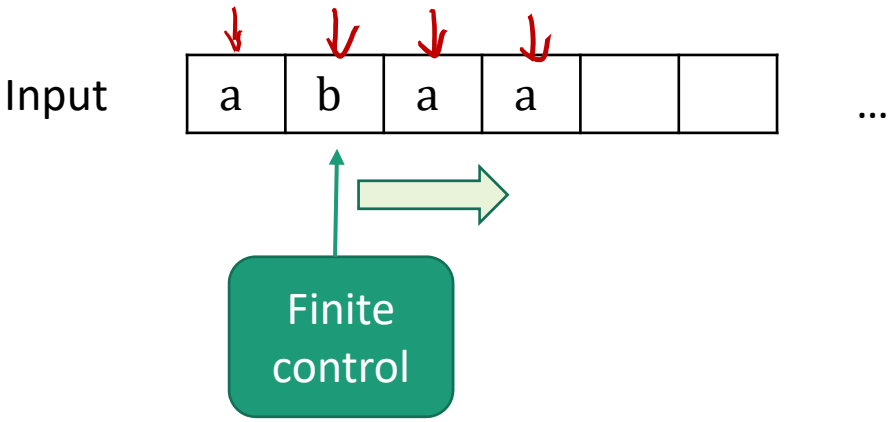
Computation is the processing of information by the **unlimited application** of a **finite set** of operations or rules



Abstraction: We don't care how the control is implemented. We just require it to have a finite number of states, and to transition between states using fixed rules.

Machine Models

- Finite Automata (FAs): Machine with a finite amount of unstructured memory

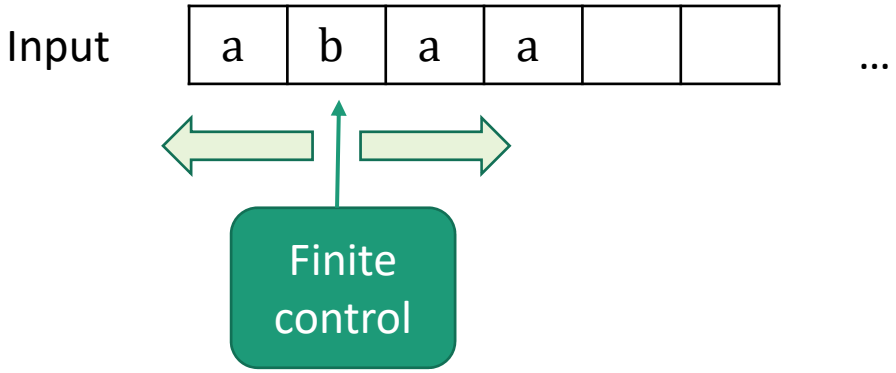


Control scans left-to-right
Can check simple patterns
Can't perform unlimited counting

Useful for modeling chips, simple control systems, choose-your-own adventure games…

Machine Models

- Turing Machines (TMs): Machine with unbounded, unstructured memory



Control can scan in both directions
Control can both read and write

Model for general sequential computation
Church-Turing Thesis: Everything we intuitively think of as "computable" is computable by a Turing Machine

What theorems would we like to prove?

We will define classes of languages based on which machines can recognize them

Inclusion: Every language recognizable by a FA is also recognizable by a TM

Non-inclusion: There exist languages recognizable by TMs which are not recognizable by FAs

Completeness: Identify a “hardest” language in a class

Robustness: Alternative definitions of the same class

Ex. Languages recognizable by FAs = regular expressions

Why study theory of computation?

- You'll learn how to formally reason about computation
- You'll learn the technology-independent foundations of CS

Philosophically interesting questions:

- Are there well-defined problems which cannot be solved by computers?
- Can we always find the solution to a puzzle faster than trying all possibilities?
- Can we say what it means for one problem to be "harder" than another?

Why study theory of computation?

- You'll learn how to formally reason about computation
- You'll learn the technology-independent foundations of CS

Connections to other parts of science:

- Finite automata arise in compilers, AI, coding, chemistry

https://cstheory.stackexchange.com/a/14818

- Hard problems are essential to cryptography
- Computation occurs in cells/DNA, the brain, economic systems, physical systems, social networks, etc.

What appeals to you about the theory of computation?

1. I want to learn new ways of thinking about computation
2. I like math and want to see how it's used in computer science
3. I'm excited about the philosophical questions about computation
4. I want to practice problem solving and algorithmic thinking
5. I want to develop a "computational perspective" on other areas of math/science
6. I actually wanted to take CS 320 or 350 but they were full

Why study theory of computation?

Practical knowledge for developers

"Boss, I can't find an efficient algorithm.
I guess I'm just too dumb."

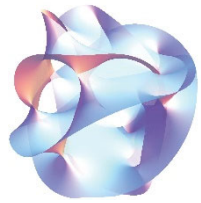
"Boss, I can't find an efficient algorithm
because no such algorithm exists."

Will you be asked about this material on job interviews?

No promises, but a true story…

More about strings and languages

String Theory



- **Symbol:** Ex. $a, b, 0, 1$
- **Alphabet**: A finite set Σ Ex. $\Sigma = \{a, b\}$
- **String:** A finite concatenation of alphabet symbols

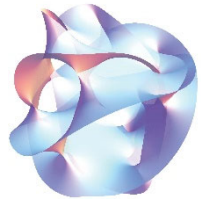
  Ex. $bba, ababb$

  ε denotes empty string, length 0

  Σ^* = set of all strings using symbols from Σ

  Ex. $\{a, b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, \ldots\}$
- **Language:** A set $L \subseteq \Sigma^*$ of strings

String Theory



- **Length** of a string, written $|x|$, is the number of symbols

Ex. $|\text{abba}| =$ 4 $\qquad |\varepsilon| =$ 0

- **Concatenation** of strings x and y, written xy, is the symbols from x followed by the symbols from y

Ex. $x = \text{ab}, y = \text{ba} \quad \Rightarrow \quad xy =$ abba

$x = \text{ab}, y = \varepsilon \quad \Rightarrow \quad xy =$ ab $\qquad x\varepsilon = x$

- **Reversal** of string x, written x^R, consists of the symbols of x written backwards

Ex. $x = \text{aab} \quad \Rightarrow \quad x^R =$ baa

$x = x_1 x_2 \ldots x_n \Rightarrow x^R = x_n x_{n-1} \ldots x_2 x_1$

Fun with String Operations

What is $(xy)^R$?

Ex. $x = \text{aba}, y = \text{bba}$ $\Rightarrow$ $xy =$ abbabba

$\Rightarrow$ $(xy)^R =$ abbaaba

1. $x^R y^R$
2. $y^R x^R$
3. $(yx)^R$
4. xy^R

Reverse has higher precedence than concat

$xy^R \neq (xy)^R$

Fun Proofs with String Operations

Claim: $(xy)^R = y^R x^R$ (each x_i is a symbol)

Proof: Let $x = x_1 x_2 \dots x_n$ and $y = y_1 y_2 \dots y_m$

Then $(xy)^R = (x_1 x_2 \dots x_n y_1 y_2 \dots y_m)^R$

$= y_m y_{m-1} \dots y_1 \; x_n \dots x_1$

$= y^R x^R$

Not even the most formal way to do this:

1. Define string length recursively
2. Prove by induction on $|y|$

not necessary for us

Languages

A language L is a set of strings over an alphabet Σ

i.e., $L \subseteq \Sigma^*$

Languages = computational (decision) problems

Input: String $x \in \Sigma^*$

Output: Is $x \in L$? (YES or NO?)

Accept or reject

Some Simple Languages

	$\Sigma = \{0, 1\}$	$\Sigma = \{a, b, c\}$
$\emptyset$ (Empty set)	$\{\ \}$	$\{\ \}$
Σ^* (All strings)	$\{\epsilon, 0, 1, 00, 01, 10, 11, \ldots\}$	$\{\epsilon, a, b, c, aa, ab, ac, ba, bb, \ldots\}$
$\Sigma^n = \{x \in \Sigma^* \mid \lvert x\rvert = n\}$ (All strings of length n)	$n=2$: $\{0,1\}^2 = \{00, 01, 10, 11\}$	$n=2$: $\{a,b,c\}^2 = \{aa, ab, ac, ba, \ldots\}$

Some More Interesting Languages

- L_1 = The set of strings $x \in \{a, b\}^*$ that have an equal number of a's and b's

  aabbab $\in L_1$ baa $\notin L_1$

- L_2 = The set of strings $x \in \{a, b\}^*$ that start with (0 or more) a's and are followed by an equal number of b's

  aabb $\in L_2$ abba $\notin L_2$ abab $\notin L_2$

  $= \{a^n b^n \mid n \geq 0\}$

- L_3 = The set of strings $x \in \{0, 1\}^*$ that contain the substring '0100'

  11**0100**01 $\in L_3$ 01110 $\notin L_3$

  $= \{x0100y \mid x, y \in \{0,1\}^*\}$

Some More Interesting Languages

- L_4 = The set of strings $x \in \{a, b\}^*$ of length at most 4

  $ab \in L_4$ $\quad$ $abbab \notin L_4$

  $= \{ x \in \{a,b\}^* \mid |x| \leq 4 \}$

- L_5 = The set of strings $x \in \{a, b\}^*$ that contain at least two a's

  $aab \in L_5$ $\quad$ $ababa \in L_5$ $\quad$ $bba \notin L_5$

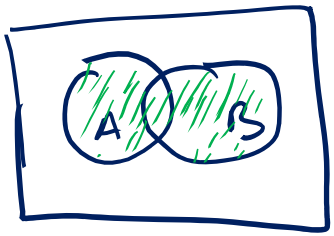
  $= \{ xayaz \mid x,y,z \in \{a,b\}^* \}$

New Languages from Old

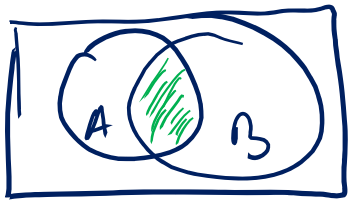
$L_6 =$ The set of strings $x \in \{a, b\}^*$ that have an equal number of a's and b's and length greater than 4

Since languages are just sets of strings, can build them using set operations:

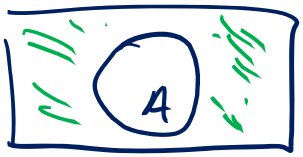
$A \cup B$ "union"



$A \cap B$ "intersection"



$\bar{A}$ "complement"



New Languages from Old

$L_6 =$ The set of strings $x \in \{a, b\}^*$ that have an equal number of a's and b's and have length greater than 4

- $L_1 =$ The set of strings $x \in \{a, b\}^*$ that have an equal number of a's and b's
- $L_4 =$ The set of strings $x \in \{a, b\}^*$ of length at most 4

$\overline{L_4} = \{ x \in \{a,b\}^* \mid |x| > 4 \}$

$\Rightarrow L_6 = L_1 \cap \overline{L_4}$

Operations Specific to Languages

- **Reverse**: $L^R = \{x^R | \, x \in L\}$

  Ex. $L = \{\varepsilon, \mathrm{a}, \mathrm{ab}, \mathrm{aab}\}$ $\Rightarrow L^R = \{\varepsilon, \mathrm{a}, \mathrm{ba}, \mathrm{baa}\}$

- **Concatenation:** $L_1 \circ L_2 = \{xy \,| x \in L_1, y \in L_2\}$

  Ex. $L_1 = \{\mathrm{ab}, \mathrm{aab}\}$ $L_2 = \{\varepsilon, \mathrm{b}, \mathrm{bb}\}$

  $\Rightarrow L_1 \circ L_2 = \{\mathrm{ab}, \mathrm{aab}, \mathrm{abb}, \mathrm{aabb}, \mathrm{abbb}, \mathrm{aabbb}\}$

  $|L_1 \circ L_2| \leq |L_1| \cdot |L_2|$

  (could be duplicates)

A Few "Traps"

String, language, or something else?

ε (empty) string

$\emptyset$ empty language

$\{\varepsilon\}$ language consisting only of empty string

$\{\emptyset\}$ neither (could be a set of languages)

Languages

Languages = computational (decision) problems

Input: String $x \in \Sigma^*$

Output: Is $x \in L$? (YES or NO?)

The language **recognized** by a program is the set of strings $x \in \Sigma^*$ that it *accepts*

What Language Does This Program Recognize?

Alphabet $\Sigma = \{a, b\}$

On input $x = x_1 x_2 \ldots x_n$:

count = 0

  For $i = 1, \ldots, n$:

    If $x_i = a$:

      count = count + 1

  If count ≤ 4: accept

  Else: reject

1. $\{x \in \Sigma^* \mid |x| > 4\}$
2. $\{x \in \Sigma^* \mid |x| \leq 4\}$
3. $\{x \in \Sigma^* \mid |x| = 4\}$
4. $\{x \in \Sigma^* \mid x \text{ has more than 4 a's}\}$
5. $\{x \in \Sigma^* \mid x \text{ has at most 4 a's}\}$
6. $\{x \in \Sigma^* \mid x \text{ has exactly 4 a's}\}$