

Foundations of Computer Science

Lecture 23

Languages: What is Computation?

A Formal Model of a Computing Problem

Decision Problems and Languages

Describing a Language: Regular Expressions

Complexity of a Computing Problem



“Say what’s on your mind, Harris—the language of dance has always eluded me.”

- ① Comparing infinite sets.
- ② Countable.
 - ▶ $\mathbb{N}_0, E, \mathbb{Z}, \mathbb{Q}$ are countable.
 - ▶ Finite binary strings \mathcal{B} is countable.
- ③ Uncountable
 - ▶ *Infinite* binary strings are uncountable.
 - ▶ Reals are uncountable.
- ④ Infinity and computing.
 - ▶ Programs are finite binary strings (countable).
 - ▶ Functions we might like to compute are infinite binary strings (uncountable).
 - ▶ Conclusion: there are **MANY** functions which *cannot* be computed by programs.

Today: Languages: What is Computation?

1 Decision problems.

2 Languages.

- Describing a language.

3 Complexity of a computing problem.

What is a Computing Problem?

Decide **YES** or **NO** whether a given integer $n \in \mathbb{N}$ is prime.

List the primes in increasing order (primes are countable),

$$\text{primes} = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, \dots\}$$

Given $n \in \mathbb{N}$, walk through this list.

- 1: If you come to n output **YES**.
- 2: If you come to a number bigger than n , output **NO**.

Not the smartest approach to primality testing, but gets to the heart of computing

LANGUAGES

Decision Problems

$$\mathcal{L}_{\text{prime}} = \{10, 11, 101, 111, 1011, 1101, 10001, 10011, 10111, 11101, \dots\}. \quad (\text{primes in binary})$$

9 is prime \leftrightarrow the *string* 1001 is in $\mathcal{L}_{\text{prime}}$.

The light is off. Every push toggles between on and off.
Given the number of pushes, decide whether the light is on or off.
Encode number of pushes by a binary string, e.g. 101 means 5 pushes.



$$\mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \dots\}.$$

The light is on for 1010 pushes, if and only if $1010 \in \mathcal{L}_{\text{push}}$.

The door should open if a person is on the mat.
Walk on (1) or off (0). E.g. 10110 means on, off, on, on, off \rightarrow open.



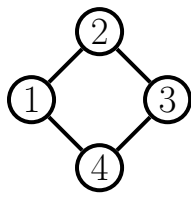
$$\mathcal{L}_{\text{door}} = \{1, 11, 101, 110, 111, 1011, 1101, 1110, 1111, \dots\}.$$

Given input w , e.g. $w = 1011$, the door is open if and only if $w \in \mathcal{L}_{\text{door}}$.

Decision problems can be formulated as testing membership in a set of strings

A Decision Problem on Graphs

- (a) **[Optimization]** What's distance between nodes ① and ③? Answer: **2**
- (b) **[Decision]** Is there a path between ① and ③ of length at most 3? **YES**.

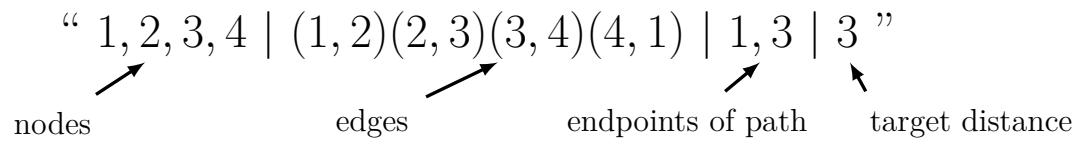


(a) is harder than (b): (a)'s answer gives (b)'s answer instantly.

Let's *encode* (b) as a string identifying the graph, nodes of interest and target distance.

“Is there a path of length at most 3 between nodes ① and ③ in the graph above.”

becomes



The graph problem can be encoded as a binary string using ASCII

```
0011000100101100001100100010110000110011001011000011010001111100001010000011000100101100001100100010100100101000001100100010110000110011
0010100100101000001100110010110000110100001010010010100000110100001011000011000100101001011111000011000100101100001100110111110000110011
```

$$\mathcal{L}_{\text{path}} = \left\{ \begin{array}{l} \text{All strings of the form "nodes | edges | endpoints of path | target distance" for which} \\ \text{the distance between the endpoints in the graph is at most the target distance.} \end{array} \right\}$$

Pop Quiz. **YES** or **NO**: “ 1, 2, 3, 4, 5 | (1, 2)(2, 3)(3, 5)(3, 4) | 1, 5 | 2 ”

Is Optimization Really Harder than Decision?

If you can solve the decision problem, you can solve the optimization problem.

Is there a path in the graph between nodes x and y of length at most **1**? NO

Is there a path in the graph between nodes x and y of length at most **2**? NO

Is there a path in the graph between nodes x and y of length at most **3**? NO

Is there a path in the graph between nodes x and y of length at most **4**? YES

You ask the decision question until the answer is YES.

The minimum-pathlength between x and y is 4.

It can take long, but it works.

Decision and optimization are “equivalent” when it comes to *solvability*.

A computing problem is a decision problem.

Standard formulation of a decision problem:

Problem: GRAPH-DISTANCE- D

Input: Finite graph G ; nodes x, y ; target distance D .

Question: Is there an (x,y) -path in G of length at most D .

Every decision problem has a **YES**-set, which we usually don't explicitly list.

$$\begin{aligned}\text{YES-set} &= \{\text{input strings } w \text{ for which the answer is YES}\} \\ &= \{w_1, w_2, w_3, \dots\}.\end{aligned}$$

← A *language* is any set of finite binary strings

A computing problem is a **YES**-set, a set of *finite* binary strings.

Computing Problems Are Languages

Language: Set of finite binary strings.

Solving the problem

Give a “procedure” to tell if a general input w is in the language (YES-set).

Abstract, precise and general formulation of a computing problem.

	$\{\epsilon, 1, 10, 01\}$	← finite language
Σ^*	$\{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \dots\}$	← all finite strings
$\mathcal{L}_{\text{prime}}$	$\{10, 11, 101, 111, 1011, 1101, 10001, \dots\}$	
$\mathcal{L}_{\text{push}}$	$\{1, 01, 11, 001, 011, 101, 111, 0001, 0011, \dots\}$	
$\mathcal{L}_{\text{door}}$	$\{1, 11, 101, 110, 111, 1011, 1101, \dots\}$	
$\mathcal{L}_{\text{unary}}$	$\{\epsilon, 1, 11, 111, 1111, \dots\} = \{1^{\bullet n} \mid n \geq 0\}$	← strings of 1s
$\mathcal{L}_{(01)^n}$	$\{\epsilon, 01, 0101, 010101, \dots\} = \{(01)^{\bullet n} \mid n \geq 0\}$	
$\mathcal{L}_{0^n 1^n}$	$\{01, 0011, 000111, \dots\} = \{0^{\bullet n} 1^{\bullet n} \mid n \geq 0\}$	
\mathcal{L}_{pal}	$\{\epsilon, 0, 1, 00, 11, 000, 010, 101, 111, \dots\}$	← palindromes
$\mathcal{L}_{\text{repeated}}$	$\{\epsilon, 00, 11, 0000, 0101, 1010, 1111, \dots\}$	← repeated strings



Describing a Language: String Patterns and Variables

An example where there is a clear pattern,

$$\mathcal{L} = \{\varepsilon, 01, 0101, 010101, \dots\}.$$

Use a variable to formally define \mathcal{L} :

$$\mathcal{L} = \{w \mid w = (01)^{\bullet n}, \text{ where } n \geq 0\}. \quad (\text{informally } \{(01)^{\bullet n} \mid n \geq 0\})$$

More than one variable:

$$\{u \bullet v \mid u \in \Sigma^* \text{ and } v = u^R\} = \{\varepsilon, 00, 11, 0000, 0110, 1001, 1111, \dots\}. \quad \leftarrow \begin{array}{l} \text{even} \\ \text{palindromes} \end{array}$$

Pop Quiz. Formally define $\mathcal{L}_{\text{add}} = \{0100, 011000, 001000, 00110000, 00010000, 0001100000, 01110000, 0011100000, 000111000000, \dots\}$

For more complicated patterns, we use regular expressions, e.g. the Unix/Linux command

ls FOCS* (Lists everything that starts with FOCS (* is the “wild-card”).)

The Regular Expression: $\{1, 11\} \bullet \overline{\{0, 01\}^*} \bullet (\{00\} \cup \{1\}^*)$

Basic building blocks are finite languages:

- $\{1, 11\}$
- $\{0, 01\}$
- $\{00\}$
- $\{1\}$

Combine these using

- union, intersection, complement
 - (Familiar.)
- concatenation \bullet , Kleene-star $*$
 - (What?!?)

Concatenation of languages.

$$\mathcal{L}_1 \bullet \mathcal{L}_2 \bullet \mathcal{L}_3 = \{w_1 \bullet w_2 \bullet w_3 \mid w_1 \in \mathcal{L}_1, w_2 \in \mathcal{L}_2, w_3 \in \mathcal{L}_3\}.$$

$$\begin{aligned} \{0, 01\} \bullet \{0, 11\} &= \{00, 011, 010, 0111\} \\ \{0, 11\} \bullet \{0, 01\} &= \{00, 001, 110, 1101\} && \mathcal{L}_1 \bullet \mathcal{L}_2 \neq \mathcal{L}_2 \bullet \mathcal{L}_1 \\ \{0, 01\} \bullet \{0, 01\} &= \{0, 01\}^{\bullet 2} = \{00, 001, 010, 0101\} && \text{(self-concatenation)} \end{aligned}$$

Pop Quiz. What is $\{0, 01\} \bullet \{1, 10\}$? What is $\{0, 01\}^{\bullet 3}$? What is $\{0, 01\}^{\bullet 0}$?

Kleene star: All possible concatenations of a finite number of strings from a language.

$$\begin{aligned} \{0, 01\}^* &= \{\epsilon, 0, 01, 00, 001, 010, 0101, 000, 0010, \dots\} = \bigcup_{n=0}^{\infty} \{0, 01\}^{\bullet n}; \\ \{1\}^* &= \{\epsilon, 1, 11, 111, 1111, 11111, \dots\} = \bigcup_{n=0}^{\infty} \{1\}^{\bullet n}. \end{aligned}$$

Pop Quiz. Which of the strings $\{101110, 00111, 00100, 01100\}$ can you generate using $\{0, 01\}^* \bullet \{1, 10\}^*$?

The Regular Expression: $\{1, 11\} \bullet \overline{\{0, 01\}^*} \bullet (\{00\} \cup \{1\}^*)$

$$\{0, 01\}^* = \{\varepsilon, 0, 01, 00, 001, 010, 0101, 000, 0010, \dots\}$$

$$\{1\}^* = \{\varepsilon, 1, 11, 111, 1111, 11111, \dots\}$$

To generate 1110111:

$$11 \in \{1, 11\}$$

$$10 \in \overline{\{0, 01\}^*}$$

$$111 \in \{00\} \cup \{1\}^*$$

Hence $1110111 \in \{1, 11\} \bullet \overline{\{0, 01\}^*} \bullet (\{00\} \cup \{1\}^*)$

Pop Quiz Is there another way to generate 1110111?

Pop Quiz Yes or no: $11110010 \in \{1, 11\} \bullet \overline{\{0, 01\}^*} \bullet (\{00\} \cup \{1\}^*)$?

Challenges Involving Regular Expressions

- 1 Is there a simple procedure to test if a given string satisfies a regular expression?

$$11110010 \in \{1, 11\} \bullet \overline{\{0, 01\}^*} \bullet (\{00\} \cup \{1\}^*) \quad ???$$

- 2 Regular expression for all palindromes (strings which equal their reversal)?

Recursively Defined Languages: Palindromes

① $\varepsilon, 0, 1 \in \mathcal{L}_{\text{palindrome}}$.

[basis]

② $w \in \mathcal{L}_{\text{palindrome}} \rightarrow 0 \bullet w \bullet 0 \in \mathcal{L}_{\text{palindrome}},$
 $1 \bullet w \bullet 1 \in \mathcal{L}_{\text{palindrome}}.$

[constructor rules]

③ Nothing else is in $\mathcal{L}_{\text{palindrome}}$.

[minimality]

Pop Quiz. Similar looking languages: $\{0^n 1^k \mid n, k \geq 0\}$ and $\{0^n 1^n \mid n \geq 0\}$

Give recursive definitions of these languages.

Give regular expressions for these languages.

These computing problems look similar.

They are **VERY** different. Which do you think is more “complex”?

How to define complexity of a computing problem?

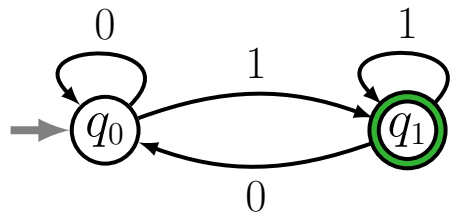
Complexity of a Computing Problem

$$\mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1011, \dots\} \quad (\text{strings ending in } 1)$$

difficult problem \leftrightarrow “complex” **YES**-set \leftrightarrow hard to test membership in **YES**-set

How do we test membership? That brings us to *Models Of Computing*.

\models 1 1 0 1

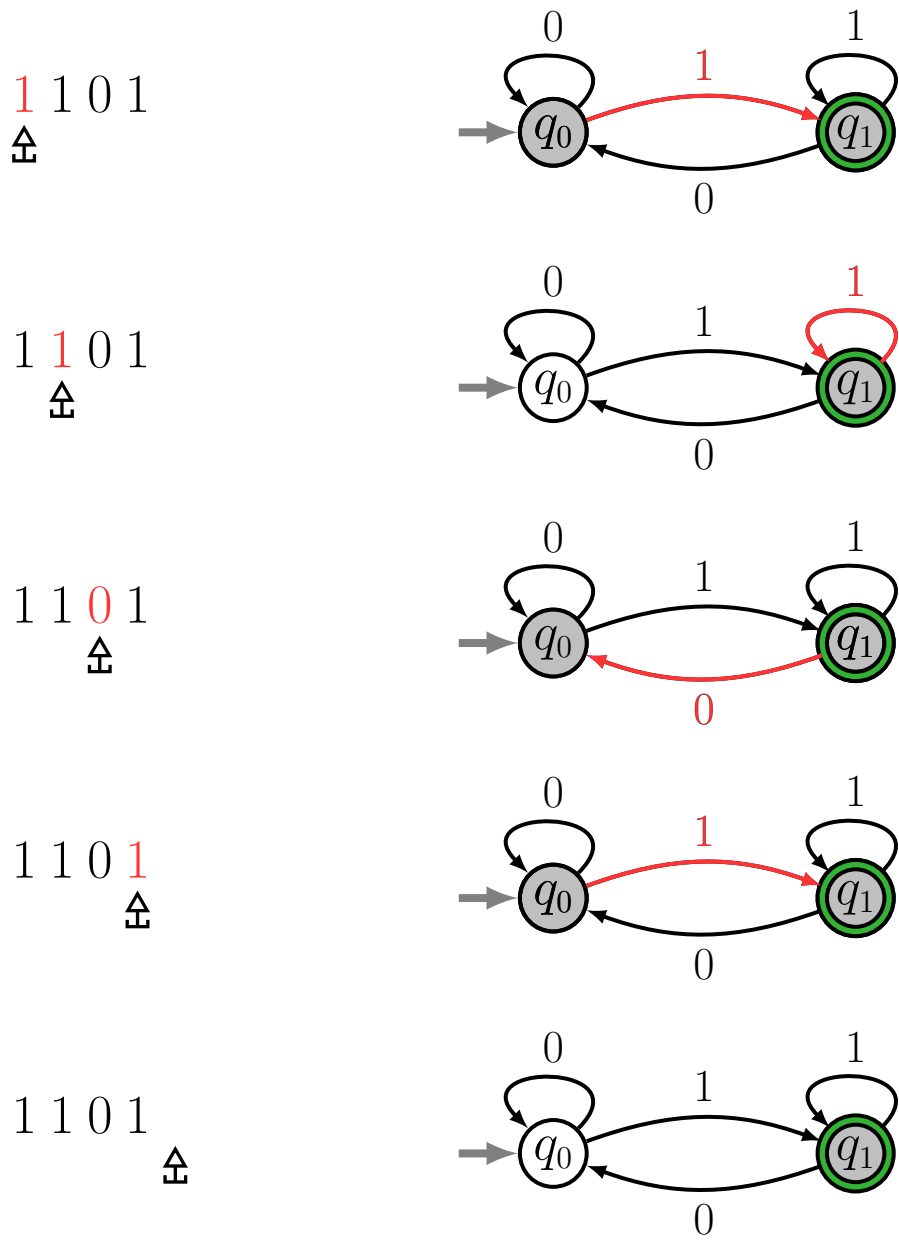


Visual encoding of four (machine-level) instructions:

- 1: In state q_0 , when you process a 0, transition to state q_0 .
- 2: In state q_0 , when you process a 1, transition to state q_1 .
- 3: In state q_1 , when you process a 0, transition to state q_0 .
- 4: In state q_1 , when you process a 1, transition to state q_1 .

“Easy” to implement as a mechanical device.

A Simple Computing Machine (DFA)

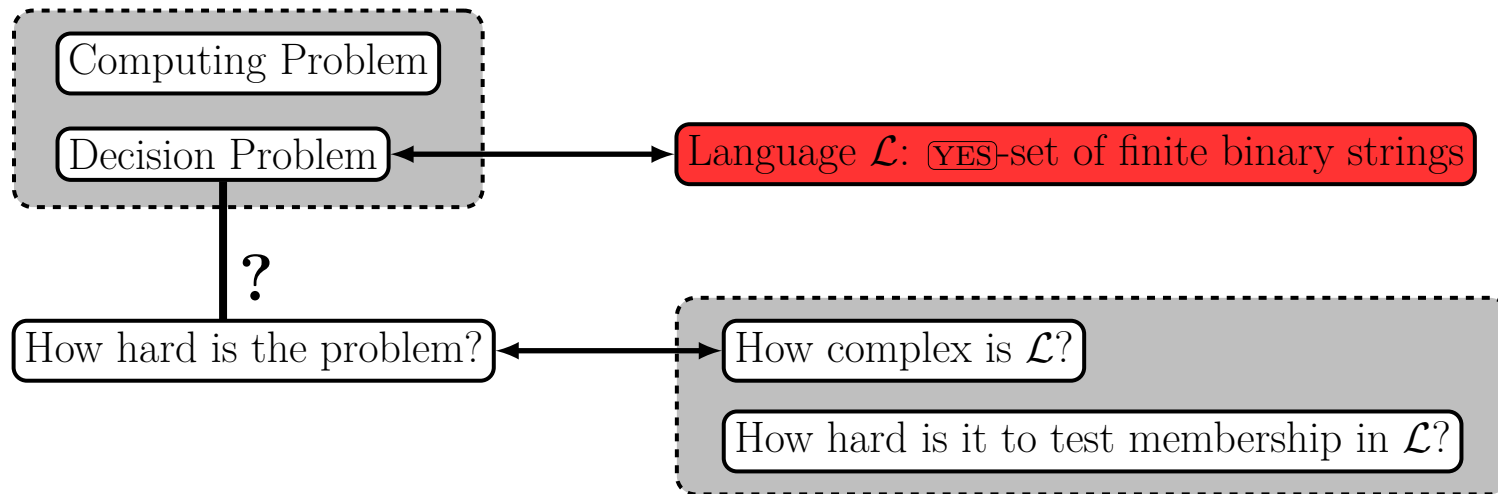


(current state in gray)

$$\mathcal{L}_{\text{push}} = \{1, 01, 11, 001, 011, 101, 111, 0001, \dots\}$$

Strings in $\mathcal{L}_{\text{push}}$ end in the “accepting” state q_1 .
 Strings not in $\mathcal{L}_{\text{push}}$ do not.

Computing Problems and Their Difficulty



A problem can be harder in two ways.

- 1 The problem needs more resources. For example, the problem can be solved with a similar machine to ours, except with more states.
- 2 The problem needs a different *kind* of computing machine, with superior capabilities.

The first type of “harder” is the focus of a follow-on algorithms course.

We focus on what *can and can't be solved* on a particular kind of machine.