

What is Computing?

Reading

- Malik Magdon-Ismael. Discrete Mathematics and Computing.
 - Chapter 23

Overview

- Decision problems
- Languages
 - Describing a language
- Complexity of a computing problem

What is a Computing Problem?

- There are many classes of computing problems
 - And many questions one can ask whose answers can be computed
- For example, *decide* YES or NO whether a given integer $n \in \mathbb{N}$ is prime
 - This is an example of a decision problem
- First, list the primes in increasing order (primes are countable)
$$primes = \{2,3,5,7,11,13,17,19,23, \dots\}$$
- Here's a quick decision algorithm:
 - Given $n \in \mathbb{N}$, walk through *primes*
 1. If you come to n output YES
 2. If you come to a number bigger than n , output NO
- Not the fastest approach to primality testing, but gets to the heart of computing
- To talk about what is computable, we need to come up with the LANGUAGES of computing!

Decision Problems

- Consider the set of all prime numbers in binary:

$$\mathcal{L}_{prime} = \{10, 11, 101, 111, 1011, 1101, 10001, 10011, 10111, 11101, \dots\}$$

- To answer a question like “Is 9 prime?”, we need to look up the binary representation of 9, 1001, and check if $1001 \in \mathcal{L}_{prime}$
- Consider a push-lamp. 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
 - Assuming lamp starts in OFF state, what number of pushes correspond to ON?
 - 1, 3, 5, 7, ..., i.e., all odd numbers
 - What binary strings correspond to all odd numbers?
 - All strings that end in 1 bit:
$$\mathcal{L}_{push} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, \dots\}$$
 - The light is on after 1010 pushes if and only if $1010 \in \mathcal{L}_{push}$



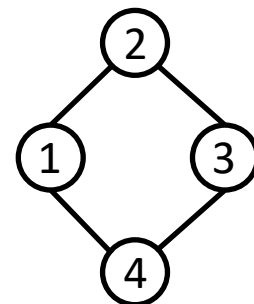
Decision Problems, cont'd

- Consider an electric door with an electric doormat
 - Door opens if you step on the doormat
 - If you step on the doormat, we encode the event as 1
 - If you step off the doormat, we encode the event as 0
 - E.g., 10110 means on, off, on, on, off → open
 - (Two people can step on it at the same time)
 - What are all strings that should lead to the door being open?
 - All strings that start with 1 and have more 1's than 0's
$$\mathcal{L}_{door} = \{1, 11, 101, 110, 111, 1011, 1101, 1110, 1111, \dots\}$$
 - Given input w , e.g., $w = 1011$, the door is open if and only if $w \in \mathcal{L}_{door}$
- **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?
 - Answer: yes
- Which problem is harder?
 - (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”
 - What information does a decision algorithm need?
 - Encode the vertices | edges | start/end nodes | path length



“ 1, 2, 3, 4 | (1, 2)(2, 3)(3, 4)(4, 1) | 1, 3 | 3 ”

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

```
0011000100101100001100100010110000110011001011000011010001111100001010000011000100101100001100100010100100101000001100100010110000110011  
0010100100101000001100110010110000110100001010010010100000110100001011000011000100101001011111000011000100101100001100110111110000110011
```

$\mathcal{L}_{path} = \{All\ strings\ of\ form\ "nodes|edges|endpoints\ of\ path|target\ distance"\ for\ which\ the\ distance\ between\ the\ endpoints\ in\ the\ graph\ is\ at\ most\ the\ target\ difference\}$

Is Optimization Really Harder than Decision?

- Can I use the decision problem to obtain an answer for the optimization problem?
- Suppose I ask the decision procedure the following questions
 - Is there a path in the graph between nodes x and y of length at most **1**?
 - Suppose I get back NO
 - Is there a path in the graph between nodes x and y of length at most **2**?
 - Suppose I get back NO
 - Is there a path in the graph between nodes x and y of length at most **3**?
 - Suppose I get back NO
 - Is there a path in the graph between nodes x and y of length at most **4**?
 - Suppose I get back YES
- Keep asking 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.**

Languages

- 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.

- Examples:

- A finite language

$$\{\varepsilon, 1, 10, 01\}$$

- All finite strings

$$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \dots\}$$

- All prime numbers

$$\mathcal{L}_{prime} = \{10, 11, 101, 111, 1011, 1101, 10001\}$$

- Push-lamp language

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

- Doormat language

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

Computing Problems Are Languages, cont'd

- More examples

- All unary strings (strings of all 1's)

$$\mathcal{L}_{unary} = \{\varepsilon, 1, 11, 111, 1111, \dots\} = \{1^{\bullet n} | n \geq 0\}$$

- All strings with repeated 01 substrings

$$\mathcal{L}_{(01)^n} = \{\varepsilon, 01, 0101, 010101, \dots\} = \{(01)^{\bullet n} | n \geq 0\}$$

- All strings where n 0's are followed by n 1s

$$\mathcal{L}_{0^n 1^n} = \{\varepsilon, 01, 0011, 000111, \dots\} = \{0^{\bullet n} 1^{\bullet n} | n \geq 0\}$$

- All palindromes

$$\mathcal{L}_{pal} = \{\varepsilon, 0, 1, 00, 11, 000, 010, 101, 111, \dots\}$$

Describing a Language: String Patterns and Variables

- Some languages are easier to describe than others

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

- In this case, we can use a variable to formally define \mathcal{L} :

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

– (informally, $\{(01)^{\bullet n} \mid n \geq 0\}$)

- Some cases are slightly harder (maybe use 2 variables):

$$\begin{aligned} \{u \bullet v \mid u \in \Sigma^* \text{ and } v = u^R\} &= \\ &= \{\varepsilon, 00, 11, 0000, 1111, \dots\} \end{aligned}$$

– What is this set?

– All even palindromes

- **Exercise.** Define

$$\mathcal{L}_{add} = \{0100, 011000, 001000, 00110000, 00010000, 0001100000, 01110000, 0011100000, 000111000000, \dots\}$$

– Answer: $\{0^{\bullet n} \bullet 1^{\bullet m} \bullet 0^{\bullet n+m}\}$

Regular Expressions

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

```
ls FOCS*
```

- Does anyone know what that command does?
 - lists everything in the folder that starts with FOCS
 - the * is a “wild-card”, means “everything”

The Regular Expression:

$$\{1, 11\} \cdot \overline{\{0, 01\}^*} \cdot (\{00\} \cup \{1\}^*)$$

- Regular expression basic building blocks are finite languages:

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

- Combine these using
 - union, intersection, complement
 - So far so good
 - concatenation (\cdot), Kleene-star ($*$)
 - Um, OK?

- **Concatenation of languages.**

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

- Example:

$$\{0, 01\} \cdot \{0, 11\} = \{00, 011, 010, 0111\}$$

- What about $\{0, 11\} \cdot \{0, 01\}$?

$$\{0, 11\} \cdot \{0, 01\} = \{00, 001, 110, 1101\}$$

- Concatenation is not commutative! ($\mathcal{L}_1 \cdot \mathcal{L}_2 \neq \mathcal{L}_2 \cdot \mathcal{L}_1$)

- Self-concatenation:

$$\{0, 01\} \cdot \{0, 01\} = \{0, 01\}^{\bullet 2} = \{00, 001, 010, 0101\}$$

The Regular Expression:

$$\{1, 11\} \cdot \overline{\{0, 01\}^*} \cdot (\{00\} \cup \{1\}^*)$$

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

$$\begin{aligned} \{0,01\}^* &= \{\varepsilon, 0,01,00,001,010,0101,000, \dots\} \\ &= \bigcup_{n=0}^{\infty} \{0,01\}^{\bullet n} \end{aligned}$$

- Similarly,

$$\begin{aligned} \{1\}^* &= \{\varepsilon, 1,11,111,1111,11111, \dots\} \\ &= \bigcup_{n=0}^{\infty} \{1\}^{\bullet n} \end{aligned}$$

- To generate 1110111 (from the regular expression in the title):

$$\begin{aligned} 11 &\in \{1,11\} \\ 10 &\in \overline{\{0,01\}^*} \\ 111 &\in \{00\} \cup \{1\}^* \end{aligned}$$

- Hence,

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

Questions Regarding Regular Expressions

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

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

- The approach we've seen so far requires a lot of search
- Can we write a regular expression for all palindromes (strings which equal their reversal)?
 - “Reverse” is not an operator in regular expressions

Recursively Defined Languages: Palindromes

- Let's define the palindromes language:

$\varepsilon, 0, 1 \in \mathcal{L}_{\text{palindrome}}$ [basis]

$w \in \mathcal{L}_{\text{palindrome}} \rightarrow 0 \bullet w \bullet 0 \in \mathcal{L}_{\text{palindrome}}$ [constructor rules]

$\rightarrow 1 \bullet w \bullet 1 \in \mathcal{L}_{\text{palindrome}}$

Nothing else is in $\mathcal{L}_{\text{palindrome}}$ [minimality]

Recursively Defined Languages, cont'd

- Here are two very similar looking languages:

$$\{0^n 1^k \mid n, k \geq 0\}$$

$$\{0^n 1^n \mid n \geq 0\}$$

- Are they the same?
- Second one only has strings with the same number of 0s and 1s
- These computing problems look similar.
- They are **VERY** different. Which do you think is more “complex”?
 - Turns out the second one is much harder. Why?
 - We need to remember the number of 0s in order to check if the number of 1s is the same
- How to define complexity of a computing problem?

Complexity of a Computing Problem

- Remember the lamp-push language:

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

- (strings ending in 1)
- We define a difficult set intuitively as a set where:
 - We have a “complex” YES-set
 - It is computationally “hard” to test membership in YES-set
- How do we test membership? That brings us to *Models Of Computing*.
- Suppose we are given a string 1101
 - Is it in the push language?
 - Yes
 - Turns out we can come up with a very simple device to test membership

Complexity of a Computing Problem, cont'd

- Remember the lamp-push language:

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

- (strings ending in 1)
- Suppose we are given a string 1101
- Consider the following machine with two states



- Depending on the machine's state, there are four possible rules:

Complexity of a Computing Problem, cont'd

- Remember the lamp-push language:

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

- (strings ending in 1)
- Suppose we are given a string 1101
- Consider the following machine with two states



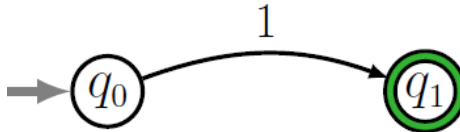
- Depending on the machine's state, there are four possible rules:
 1. In state q_0 , when you process a 0, transition to state q_0

Complexity of a Computing Problem, cont'd

- Remember the lamp-push language:

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

- (strings ending in 1)
- Suppose we are given a string 1101
- Consider the following machine with two states



- Depending on the machine's state, there are four possible rules:
 - In state q_0 , when you process a 0, transition to state q_0
 - In state q_0 , when you process a 1, transition to state q_1

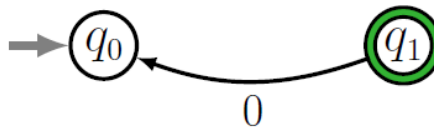
Complexity of a Computing Problem, cont'd

- Remember the lamp-push language:

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

– (strings ending in 1)

- Suppose we are given a string 1101
- Consider the following machine with two states



- Depending on the machine's state, there are four possible rules:
 - In state q_0 , when you process a 0, transition to state q_0
 - In state q_0 , when you process a 1, transition to state q_1
 - In state q_1 , when you process a 0, transition to state q_0

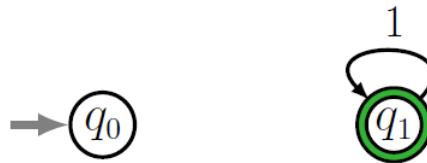
Complexity of a Computing Problem, cont'd

- Remember the lamp-push language:

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

– (strings ending in 1)

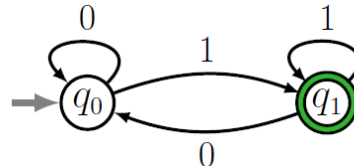
- Suppose we are given a string 1101
- Consider the following machine with two states



- Depending on the machine's state, there are four possible rules:

- In state q_0 , when you process a 0, transition to state q_0
- In state q_0 , when you process a 1, transition to state q_1
- In state q_1 , when you process a 0, transition to state q_0
- In state q_1 , when you process a 1, transition to state q_1

- Full machine is then

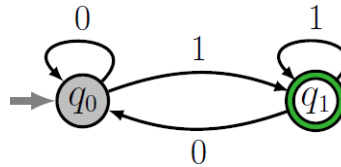


A Simple Computing Machine (DFA)

- DFA stands for deterministic finite automaton
- Let's see how the computing machine works
- We process the string in order
 - After each symbol we perform a transition in the DFA
 - We start from the DFA initial state
 - If we end in an accepting state, the string is accepted, i.e., it is in the language

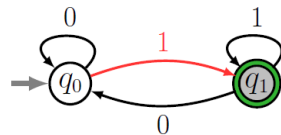
A Simple Computing Machine, cont'd

- Start the machine in the initial state

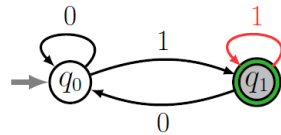


- Then process the string in order

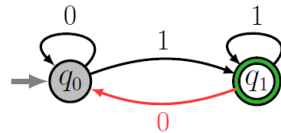
1101



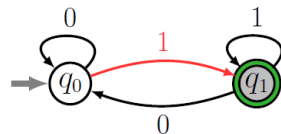
1101



1101

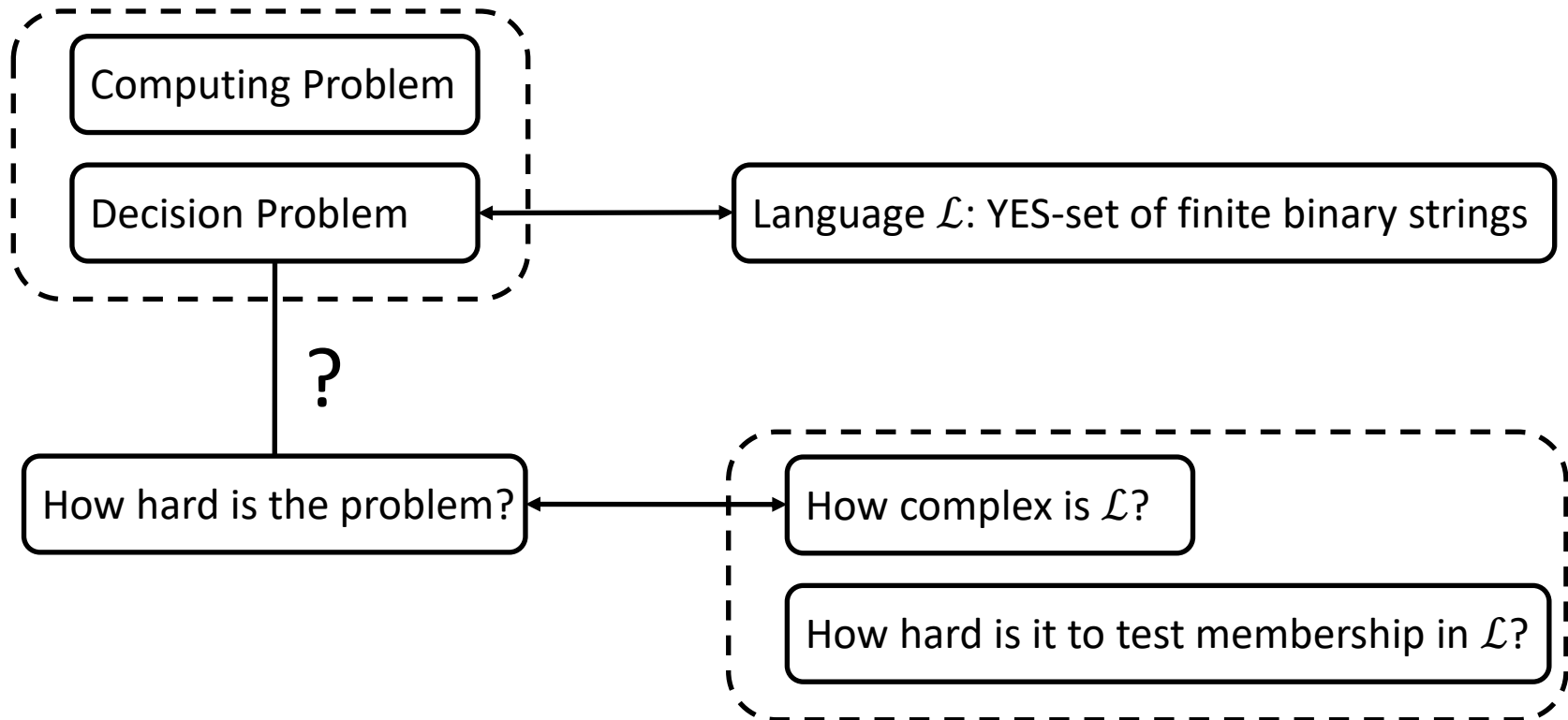


1101



- Remember the lamp-push language:
 $\mathcal{L}_{push} = \{1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, \dots\}$
- Strings in \mathcal{L}_{push} end in “accepting” state q_1
- Strings not in \mathcal{L}_{push} do not

Computing Problems and Their Difficulty



Computing Problems and Their Difficulty, cont'd

- 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.