

Lecture 1: Overview & Fundamental Concepts

What is a Problem?

Classification (recognition) problem: Determining for a class of inputs whether a given input belongs to a certain class;

e.g., recognizing strings containing a specified substring;
recognizing well-formed arithmetic expressions;
recognizing prime numbers;
recognizing satisfiable boolean formulas; etc.

Mapping problem: Mapping each of a class of inputs to an output related to the input in some specified way;

e.g., Making up/down/wait decisions for 1 or more elevators,
as a function of buttons pushed on various floors;
mapping character strings to phonemes for speech generation;
finding a solution to a boolean formula; etc.

Specifying a Problem Declaratively (Descriptively)

We often use **ordinary language plus mathematical notation** to specify a problem declaratively;

e.g., “binary strings containing 0011”;
 $\{x0011y \mid x, y \text{ are binary strings}\};$
 $\{\langle G, k \rangle \mid k \text{ is the size of the largest clique of } G,$
where G is an undirected graph $\};$ etc.

We may also use various **declarative formalisms** for problem description, such as regular expressions, formal grammars, or the λ -calculus;

e.g., $(0 \cup 1)^*0011(0 \cup 1)^*;$
 $S \rightarrow 0011 \mid BS \mid SB, B \rightarrow 0 \mid 1;$
 $\lambda xy(x + y);$ etc.

What Is Computation?

Computation is a systematic way of obtaining an answer to a problem.

The systematic nature of computation allows the use of **computing devices** for actual computation.

The abstraction of devices of the “same kind” is a **model**

- “Is this model ‘different’ from that model?”
- “What problems can be solved under this model?”

Problem Classes

From a declarative perspective:

Class of Problems = Descriptive Formalism + Interpretation of the Formalism; (or, an informal description of the class)

From a computational perspective:

Class of Problems = Computation model + Concept of “Solving a Problem”

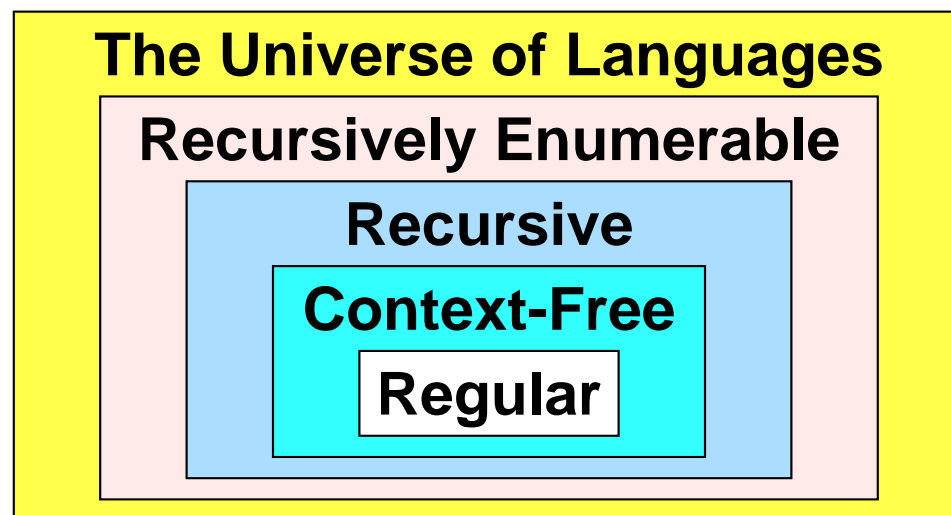
- “Are this model (or descriptive formalism) and that model (or descriptive formalism) equivalent?” → “Is class A equal to class B?”
- “Can any problem be solved under this model (or described in this formalism)?” → “Exactly what is class A?”

Computation Resources

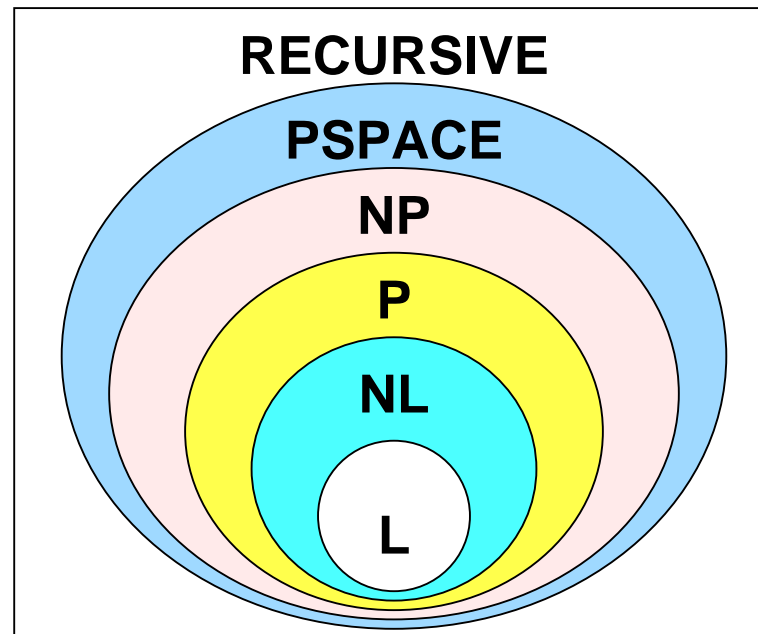
Distinction between resource-bounded computation and resource-unbounded computation.

How does bounding resources affect the power of computation?

Class Overview: Resource-Unbounded



Class Overview: Resource-Bounded



Alphabet, Strings, Languages, etc.

See page 13 of the textbook

Alphabet, Strings, Languages, etc.

- An **alphabet** is any finite set, whose members are called **symbols**.
- A **string (or word) over an alphabet** is a sequence of symbols from the alphabet written one after another.
- The **length** of a word w , denoted by $|w|$, is the number of symbols in it.
- The **empty string**, denoted by ϵ , is the string with no symbols in it.

Alphabet, Strings, Languages, etc. (cont'd)

- A string z is a **substring** of w if z appears consecutively within w .
- The **concatenation** of strings x and y is the string constructed by appending y after x .
- A **language** is a collection of strings.
- The **complement** of a language is the collection of all non-members.
- A **class** is a collection of languages.

Alphabet, Strings, Languages, etc. (cont'd)

Example Let $\Sigma = \{0, 1\}$ be an **alphabet**. The **symbols** of Σ are 0 and 1.

Let $z = 001111010$. Then z is a **string** over Σ . 1111 is a **substring** of z while 11111 is not. The **concatenation** of $a = 000$ and $b = 111$ is 000111.

The set of all strings over Σ with the same number of 0s and 1s is a **language**. The collection of all languages over Σ is a **class**.

Boolean Logic

A **Boolean variable** takes on one of 0 (FALSE) and 1 (TRUE). The **negation** of x , denoted by \bar{x} or $\neg x$, is $1 - x$.

We will be using six **binary Boolean operators**:

| (x, y) | $(0, 0)$ | $(0, 1)$ | $(1, 0)$ | $(1, 1)$ |
|-------------------|----------|----------|----------|----------|
| \wedge | 0 | 0 | 0 | 1 |
| \vee | 0 | 1 | 1 | 1 |
| \rightarrow | 1 | 1 | 0 | 1 |
| \leftarrow | 1 | 0 | 1 | 1 |
| \leftrightarrow | 1 | 0 | 0 | 1 |
| \oplus | 0 | 1 | 1 | 0 |

Boolean Logic (cont'd)

A **predicate** is a **function** whose **range** is TRUE, FALSE. A **relation** is a predicate whose number of arguments is a fixed constant.

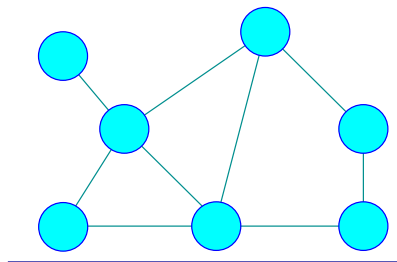
Properties of binary relation R over domain D :

- **reflexive** for all $x \in D$, xRx
- **symmetric** for all $x, y \in D$, $xRy \leftrightarrow yRx$
- **transitive** for all $x, y, z \in D$, $xRy \wedge yRz \rightarrow xRz$

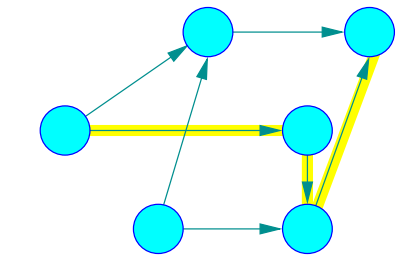
An **equivalence relation** is a binary relation that is reflexive, symmetric, and transitive

Graphs

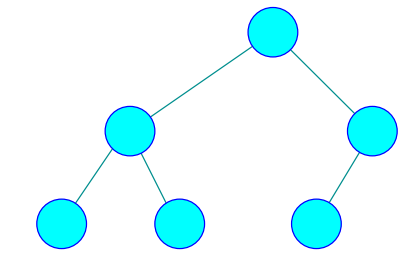
A **graph** consists of **nodes (vertices)** and **edges**. A **path** is a sequence of edges (or a sequence of nodes) that connects two nodes. A **tree** is a **connected, undirected graph** without **simple cycles**.



an undirected graph



a directed graph



a tree