

# Lecture 1: Introduction

Prof. David L. Dill

Department of Computer Science

1

## Representing Sets (discussion)

Suppose you, as a programmer, need to represent a small, finite, set.

- What does “represent” mean?

Answer: You can answer questions about it, perform computations on it.

Simple common question (membership): Is  $x \in S$ ?

Other questions:

Is  $S = \emptyset$ ?

Is  $S \cap T = \emptyset$ ? Etc.

Ok, suppose you want to a *finite representation* of *infinite* sets. How do you do it?

3

## Outline

- What is this course about? What is it good for?
- Course administration
- Basic concepts: Strings, languages, and problems.

Reading: Chapter 1 of the textbook

2

## One view of formal language theory

Automata and complexity theory is concerned with properties of *formal languages*. (A “formal language” is just an impressive name for a set of strings.)

In formal language, automata, and complexity theory, a *language* is just a set of strings.

Like many mathematical definitions, this leaves behind most of what we think of as “languages,” but can be made precise. And it leads to very profound results.

Formal languages are only interesting when they are infinite.

What about sets of other types? E.g., Sets of numbers? Sets of computer programs? Representing numbers or computer programs as strings is simple.

4

## Self-reference

Formal language theory becomes almost surrealistic because of “self reference.”

Finite representation = a string, so we can talk about *languages of other (finitely representable) languages*.

E.g., we can talk about the language of all Turing machines that halt, and then wonder about the properties of *that* language.

This leads to special proof techniques, and can tie your neurons into knots.

5

## Questions from formal language theory

What (infinite) sets are representable?

What can a computer do with the representations, in theory?

What *cannot* be done with the representations, in theory?

What problems are easy, hard, or impossible to solve computationally?

7

## What is a representation?

Suppose you have representation that can be stored in a computer.

Can all sets be represented?

No: Compare the number of possible strings (which is countable) with the number of sets of strings (uncountable).

A particular set that cannot be represented is the set of all irrational numbers – there are “too many” irrational numbers.

This raises profound questions: Which sets can be represented on a computer and which can't?

6

## Another view of formal language theory

For practical purpose, a language is the same thing as a Boolean function. Such a function is also called a *property* or a *predicate*.

For example, the predicate  $\text{even}(x)$ , which returns “true” iff  $x$  is (string representation of) an even number, can be considered to represent the set of even numbers (think of it as an “implicit set lookup”).

So, if we can answer questions about languages, we are also answering questions about properties of objects.

Example: A program that can test whether a Java program halts on all inputs tests membership in the language of all Java programs that halt on all inputs.

“Some languages cannot be represented” = “Some predicates cannot be effectively computed”

“What sets can be represented” = “What can computers do?”

8

## Applications of Automata Theory

This is probably the most theoretical course in the undergraduate curriculum.

But a lot of this material has very important practical applications.

New applications are being discovered all the time – even for things that were originally invented to answer purely theoretical questions.

9

## Application: Computer languages

Basis for compilers and program analysis.

- Lexical analysis
- Parsing
- Program analysis

Many interesting problems in programming language implementations are hard or impossible to solve in general.

Examples:

- Equivalence of grammars.
- Almost any exact analysis.

11

## Value of the course

In 20 years, computers and programming will be vastly different. But this material will be very much the same – and will still be useful.

Provides insight into fundamental questions

- Defines the questions
- Answers some
- Many are open!
- Very close connection with logic, algorithms, linguistics, others.

Provides advance problem-solving tools.

- Springboard for more advanced courses
- Research
- Applications

Practice with mathematics and proofs.

General knowledge (e.g., management, technology and policy).

10

## Application: Formal Verification

Formal verification attempts to prove system designs (e.g. programs) correct, or to find bugs.

Methods are generally from logic and automata theory. Many of the constructions in this course are used in practical tools.

- Automata constructs (e.g., product construction)
- Reductions to SAT (an NP-completeness proof technique).
- “Bounded model checking” – the idea is from Cook’s theorem

It is also important to know a little about complexity theory, since many problems in this area are hard or impossible to solve, in general.

12

## Course administration

Textbook: Hopcroft, Motwani, and Ullman, Introduction to Automata Theory, Second Edition.

Grades: Homeworks (50%), Midterm (20%), Final (30%)

Homeworks are due at the beginning of class. Solutions will be available right after class, so lates will not be accepted.

Midterm is in class, Nov 4.

Final is Thursday, Dec 11, 12:15 PM - 3:15 PM.

**There will be no alternate dates/times for the midterm or final.**

Do not announce to me that you have made unrefundable plane reservations to leave before the final.

Web page: To appear shortly at [cs154.stanford.edu](http://cs154.stanford.edu)

13

## Strings

**Informally:** A *string* is a finite sequence of symbols from some alphabet.

**Examples:**

- $\epsilon$  – the empty string (the same for every alphabet). (Leaving a blank space for the empty string is confusing, so we use the Greek letter “epsilon”).  
 $\epsilon$  is not a symbol! It is the string with no symbols; the string of zero length.
- 000, 01101 are strings over the binary alphabet
- “Weinerdog” is a string over the Ascii character set, or the English alphabet.

**Recursive definition of strings over alphabet  $\Sigma$ .**

(This is a bit more formal than the book's treatment, but equivalent.)

**Base:**  $\epsilon$  is a string over  $\Sigma$

**Induction:** If  $x$  is a string over  $\Sigma$  and  $a$  is a symbol from  $\Sigma$ , then  $xa$  is a string over  $\Sigma$ .

(Think of  $xa$  as appending a symbol to an existing string.)

**Notation:** The set of all strings over an alphabet  $\Sigma$  is written  $\Sigma^*$ .

15

## Basic concept

**Def** An *alphabet* is a non-empty finite set. The members of the alphabet are called *symbols*.

**Examples:**

- Binary alphabet  $\{0, 1\}$
- Ascii character set – the first 128 numbers, many of which are printed as special characters. Also, any other finite character set.

The capital Greek sigma ( $\Sigma$ ) is often used to represent an alphabet.

14

## Length of a string

Many functions are defined recursively on the structure of strings, and many proofs are done by induction on strings.

**Informally:** The *length* of a string is the number of occurrences of symbols in the string (the number of different positions at which symbols occur).

The length of string  $x$  is written  $|x|$ .

16

## Concatenation of strings

**Informally:** The concatenation of strings  $x$  and  $y$  over alphabet  $\Sigma$  is the string formed by following  $x$  by  $y$ . It is written  $x \cdot y$ , or (more often)  $xy$ .

**Examples:**

- $abc \cdot def = abcdef$
- $\epsilon \cdot abc = abc$

## Languages

**Def** A *language* over  $\Sigma$  is a subset of  $\Sigma^*$ .

**Note:** Of course, this omits almost everything that intuitively think is important about a language, such as meaning. But this definition nevertheless leads to incredibly useful and important results.

**Examples:**

- $\emptyset$  (the empty language)
- $\{\epsilon\}$  (the language consisting of a single empty string).
- The set of all strings with the same number of  $a$ s as  $b$ s.
- The set of all prime numbers, written as binary strings.
- The set of all strings representing Java programs that compile without errors or warnings.
- The set of all first-order logic formulas.
- The set of all theorems of number theory, in an appropriate logical notation.
- The set of all input strings for which a given Boolean Java function returns “true.”