Lecture 1: Introduction

Prof. David L. Dill

Department of Computer Science

Outline

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

Representing Sets (discussion)

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

- What does “represent” mean? (what questions/operations?)
- What representations would be appropriate?

Ok, suppose you want to represent *infinite* sets. How do you do it?

- What does “represent” mean? (what questions/operations?)
- What representations would be appropriate?

One view of formal language theory

Automata and complexity theory is concerned with properties of *formal languages*.

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.)

Basically, any object or value that is of interest to computer science can be represented as a string. So a set of anything can be considered a language.

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?

What is a representation?

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

Can all sets be represented?

No: Compare the number of possible strings (which is countable) with the number of sets of string (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?

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.

Another view of formal language theory

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

For example, the predicate $\mathbf{even}(x)$, which returns "true" iff x is 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?"

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).

Application: Computer languages

Basis for tools and programming techniques.

- 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.

Application: Formal Verifi cation

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.

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 (Σ) is often used to represent an alphabet.

Strings

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

Examples:

- ϵ – 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").

  ϵ **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 Σ.

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

Base: ϵ is a string over Σ

Induction: If x is a string over Σ and a is a symbol from Σ, then xa is a string over Σ.

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

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 $|x|$.

Recursive definition of length

Base: $|\epsilon| = 0$.

Induction: $|xa| = |x| + 1$.

Concatenation of strings

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

Examples:

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

Recursive definition of concatenation:

The definition is recursive on the structure of the second string:

Base: $x\epsilon = x$ if x is a string over Σ.

Induction: $x(ya) = (xy)a$ if x and y are strings over Σ and $a \in \Sigma$.

Note: The parentheses are not symbols, they are for grouping, so $x(ya)$ is x concatenated with ya.

Languages

Def A *language* over Σ is a subset of Σ^*.

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 as as bs.
- 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."

Proof Guidelines

1. State what is being proved precisely and clearly.
2. Start proof with an explanation of the strategy (e.g. “induction on y”)
3. Provide guideposts (e.g., **Base**, **Induction**)
4. Highlight the interesting key parts of the proof (where did you have to be clever?)
5. Make it easy for the graders to see these things.

Proof Expectations

We don’t want to lose sight of the forest because of the trees.

Here are the “forest-level” points with proofs.

- What is the proof strategy?
  - Induction on strings. **What are the base and induction steps?**
  - Induction on expressions. **What are the base and induction steps?**
  - Diagonalization
  - Reduction from another problem. **Which direction is the reduction?**
- What are the key insights in the proof?
- Often this is a construction (often something that can be implemented as a computer program)
  - Translation between regular expressions, various finite automata.
  - Translation from one problem to another.

Make sure your document explain these things clearly in your proofs.

If we can see QUICKLY that you did the right kind of proof and got the major points right, you’ll get nearly full credit.