COMS W3261
Computer Science Theory
Chapter 1 Notes

Alexander Roth

2014-09-06

Definitions

Grammars A useful model when designing software that processes data with a recursive structure.

Regular Expressions Denote the structure of data, especially text strings.

Regular Expression Special Characters

`[A-Z]` Represents a range of characters from capital 'A' to capital 'Z'.

`[ ]` Represents a blank character alone.

`*` Represents "any number of" the preceding expression.

`( )` Used to group components of the expression.

Decidability What can a computer do at all? Problems that a computer can solve are called "decidable".

Intractability What can a computer do efficiently? The problems that can be solved by a computer using no more time than some slowly growing function of the size of the input are called "tractable."

Introduction to Formal Proofs

Reduction to Definitions

- If you are not sure how to start a proof, convert all terms in the hypothesis to their definitions.

1. A set S is *finite* if there exists an integer n such that S has exactly n elements. We write $\|S\| = n$, where $\|S\|$ is used to denote the number of elements in a set S.

2. If the set S is not finite, we say S is *infinite*. Intuitively, an infinite set is a set that contains more than any integer number of elements.

3. If S and T are both subsets of some set U, then T is the *complement* of S (with respect to U) if $S \cup T = U$ and $S \cap T = \emptyset$. That is each element of U is in exactly one of S and T; put another way, T consists of exactly those elements of U that are not in S.

Proof by Contradiction Assume that the conclusion is false. Then use that assumption, together with parts of the hypothesis, to prove the opposite of one of the given statements of the hypothesis. Shown that it is impossible for all parts of the hypothesis to be true and for the conclusion to be false at the same time, there is only one possibility. The conclusion must be true whenever the hypothesis is true. Thus, the theorem is true.

Ways of Saying "If-Then"

1. H implies C.
2. H only if C.
3. C if H.
4. Whenever H holds, C follows.

Examples

1. $x \geq 4$ implies $2^x \geq x^2$.
2. $x \geq 4$ only if $2^x \geq x^2$.
3. $2^x \geq x^2$ if $x \geq 4$.
4. Whenever $x \geq 4$, $2^x \geq x^2$ follows.

The operator $\rightarrow$ can also take the place of "if-then".

If-And-Only-If Statements

Forms of the statement "A if and only if B" include: "A iff B", "A is equivalent to B" or "A exactly when B". This statement is actually two if-then statements. We prove this by doing:

1. The *if part*: "if B then A," and
2. The *only-if part*: "if A then B," which is often stated in the equivalent form "A only if B."

The operators $\leftrightarrow$ and $\equiv$ are used to denote "if-and-only-if" statements.

How Formal Do Proofs Have to Be?

There are certain things that are required in proofs, and omitting them surely makes the proof inadequate. For example, any deductive proof that uses statements which are not justified by given or previous statements, cannot be adequate.

Additional Forms of Proofs

Proving Equivalences About Sets

If E and F are two expressions representing sets, the statement $E = F$ means that the two sets represented are the same.

The Contrapositive

The *contrapositive* of the statement "if H then C" is "if not C then not H". A statement its contrapositive are either both true or both false, so we can prove either to prove the other.

The Converse

The *converse* of an if-then statement is the "other direction"; that is, the converse of "if H then C" is "if C then H." Unlike the contrapositive, which is logically equivalent to the original, the converse is *not* equivalent to the original statement.

Counterexamples

Statements that have no parameters, or that apply to only a finite number of values of its parameter(s) are called *observations*. It is often easier to prove that a statement is not a theorem than to prove it *is* a theorem.

Inductive Proofs

In automata theory, inductive proofs are used with recursively defined concepts such as trees and expressions of various sorts (e.g. regular expressions).

Inductions on Integers

- *The Induction Principle*: If we prove $S(i)$ and we prove that for all $n \geq i$, $S(i)$ implies $S(n+1)$, then we may conclude $S(n)$ for all $n \geq i$.

Structual Inductions

When we have a recursive definitions, we can prove theorems about it using the following proof form, which is called *structural induction*. Let $S(X)$ be a statement about the structures X that are defined by some particular recursive definition.

1. As a basis, prove $S(X)$ for the basis structure(s) X.
2. For the inductive step, take a structure X that the recursive definition says is formed from $Y_1, Y_2, \ldots, Y_k$. Assume that the statements $S(Y_1), S(Y_2), \ldots, S(Y_k)$ hold, and use these to prove $S(X)$.

Our conclusions is that $S(X)$ is true for all X.

Mutual Inductions

- Each of the statements must be proved separately in the basis and in the inductive step.
- If the statements are "if-and-only-if," then both directions of each must be proved, both in the basis and in the induction.

1 The Central Concepts of Automata Theory

Alphabet

An *alphabet* is a finite, nonempty set of symbols. Conventionally, we use Σ for an alphabet.

Strings

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

The Empty String

The *empty string* is the string with zero occurrences of symbols. Denoted by ϵ.

Length of a String

It is often useful to classify strings by their *length*, that is the number of positions for symbols in the string. The standard notation for the length of a string w is $|w|$.

Powers of an Alphabet

If Σ is an alphabet, we can express the set of all strings of a certain length from that alphabet by using an exponential notation. We define Σ^k to be the set of strings of length k, each of whose symbols is in Σ.

- The set of all strings over an alphabet Σ is conventionally denoted Σ^*.
- The set of nonempty strings from alphabet Σ is denoted Σ^+
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$.
- $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$.

Concatentation of Strings

Let x and y be strings. Then xy denotes the *concatenation* of x and y, that is, the string formed by making a copy of x and following it by a copy of y. ϵ is the *identity for concatenation*, since when concatenated with any string it yields the other string as a result.

Languages

A set of strings all of which are chosen from some Σ^*, where Σ is a particular alphabet, is called a *language*. If Σ is an alphabet, and $L \subseteq \Sigma^*$, then L is a *language over* Σ. The only important constraint on what can be a language is that all alphabets are finite.

Problems

A *problem* is the question of deciding whether a given string is a member of some particular language. If Σ is an alphabet, and L is a language over Σ then the problem L is:

- Given a string w in Σ^*, decide whether or not w is in L.

Summary

Finite Automata Finite automata involve states and transitions among states in response to inputs.

Regular Expressions These are structural notation for describing the same patterns that can be represented by finite automata.

Context-Free Grammars These are an important notation for describing the structure of programming languages and related sets of strings.

Turing Machines These are automata that model the power of real computers. They allow us to study decidability, the question of what can or cannot be done by a computer. They also let us distinguish tractable problems.

Deductive Proofs This basic method of proof proceeds by listing statements that are either given to be true, or that follow logically from some of the previous statements.

Languages and Problems A language is a (possibly infinite) set of strings, all of which choose their symbols from one alphabet. When the strings of a language are to be interpreted in some way, the question of whether a string is in the language is sometimes called a problem.