University of Nevada, Las Vegas Computer Science 456/656

Sets

We write $x \in X$ to mean that x is a member of a set X.

A set is defined by what its members are. Items are either members of X or not. There is no such thing as being a member twice.

Finite sets can be written using braces. For example, $\{x, y\}$ is the set whose members are x and y.

We write $X \subseteq Y$ to mean that X is a subset of Y, that is, every member of X is a member of Y.

We write $\emptyset$ to denote the empty set, the set which has no members.

If X and Y are sets, $X \cup Y$, sometimes written $X + Y$, is the union of X and Y, the set of all items which are members of either X or Y.

If X and Y are sets, $X \cap Y$ is the intersection of X and Y, the set of all items which are members of both X and Y.

Languages

An *alphabet* is a finite set. The members of an alphabet are called *symbols*. One very important alphabet is the binary alphabet $\{0, 1\}$.

A *string* is a finite sequence of symbols. The *empty string* is the string with no symbols, usually indicated as either ϵ or λ. If all the symbols of a string w are members of an alphabet Σ, we say that w is a string *over* Σ. A *binary string* is a string over the binary alphabet. $\lambda, 0, 1, 00, 01, \ldots, 110100, \ldots$ are binary strings.

A *language* L over an alphabet Σ is a set of strings over Σ. A language over the binary alphabet is called a binary language.

We write Σ^* to mean the set of all strings over an alphabet Σ. Note that Σ^* is a language over Σ. L is a language over Σ if and only if $L \subseteq \Sigma^*$.

Problems

A 0/1 problem is a problem such that the answer is always either true or false. For example, *primality* is the problem of whether a given numeral represents a prime. We distinguish between a problem and an *instance* of the problem. For example, "549755813887" is an instance of the primality problem.

Every language L gives rise to a 0/1 problem, its *membership* problem, which is whether a given string is a member of L. Conversely, every 0/1 problem can be expressed as the membership problem of some language. For example, primality is the membership problem of P, the language consisting of all binary numerals representing prime numbers.

As another example, consider the 0/1 problem of whether a graph is connected, which we'll call the "graph connectivity problem." An instance of that problem is a graph G. How do we change this problem into a language? We first need to encode members of the class of graphs as strings. A graph G is defined to be an ordered pair (V, E), where V is a set, the set of vertices, and E is the set of edges. Each member of E is a set $\{u, v\}$ where u and v are members of V. We then select an encoding. We encode $G = (E, V)$ as first an integer n, the size of V, followed by a list of pairs of numbers in the range $1 \ldots n$, one for each member of E. We'll use base 10 numerals to encode integers, since you are familiar with them. $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (,)\}$. Then an encoding of a graph G, using the encoding scheme described above, is a string over Σ, which we call $[G]$, be the encoding of G. L_{graph} be the language of all encodings of graphs.

As an example, a clique of size 3 could be encoded as the string "3,(1,2)(1,3)(2,3)." We now define $L_{\text{CONNECTED}}$ to be the subset of L_{GRAPH} consisting of the encodings of connected graphs. The *graph connectivity* problem is the membership problem of $L_{\text{CONNECTED}}$. For example, "3,(1,2)(1,3)(2,3)"$\in L_{\text{CONNECTED}}$ because it is the encoding of a connected graph.

"4,(1,2)(3,4)"$\notin L_{\text{CONNECTED}}$ because it is the encoding of a disconnected graph.

")6,,5)"$\notin L_{\text{CONNECTED}}$ because it is not the encoding of any graph, hence not of any connected graph.

An algorithm which solves the connected graph problem has two parts: Given a string $w \in \Sigma^*$, the "easy" part is to determines whether $w \in L_{\text{GRAPH}}$. If the answer is "no," the algorithm is done; otherwise it must do the "hard" part: determine whether the encoded graph is connected.

To simplify our presentations, we usually ignore the easy part, assuming that the input string encodes an instance of the problem.

Complexity of a Language

The *computational complexity* of a language is defined to be the computational complexity of its membership problem.

For example, we say that a language L over an alphabet Σ is *quadratic time* if there is an algorithm which takes as input any string $w \in \Sigma^*$ and determines, within $O(n^2)$ steps, whether $w \in L$, where $n = |w|$. We define $\mathcal{P}_{\text{TIME}}$, which we usually abbreviate as just $\mathcal{P}$, to be the class of languages which are $O(n^k)$ time for some constant k. We call those *polynomial time* languages, or, equivalently, problems.

Classes of Languages

We will consider many classes of formal languages during the course. Classes can be nested or overlap. For example, every regular language is also a decidable language, but not vice-versa.

The smallest class we consider in this course is the class of *regular* languages. A simple definition is that a language L is regular if and only if there is a deterministic finite-state machine M such that, given any string, w, M can decide whether $w \in L$ in finite time.