COMS 381, Summer 2005
Supplementary Handout 2
Decision Problems for Regular Languages
Tuesday, June 7th

1 Introduction

When formal languages (regular languages and other kinds) are used in practical applications or in more sophisticated theoretical proofs, we often find that certain basic *decision problems* arise. For instance, we may wish to know if two languages L_1 and L_2 are the same, or if a language is infinite. We will see decision problems for all the classes of languages that we cover in 381; this handout is devoted to decision problems for regular languages only.

1.1 Formal definition of a decision problem

Formally (see Lecture 2 in the textbook), decision problems are mathematical questions of the form:

Given sets A and B with $B \subseteq A$ and an item $x \in A$, is x in B?

Intuitively, the set A represents all possible inputs to the decision problem, and B represents the subset of possible inputs having some particular property. For instance, A could be the set of all graphs, and B the set of all connected graphs. The decision problem above would then be the question: Given any graph x, is x connected?

1.2 Difficulty of solving decision problems for formal languages

Decision problems can be very easy to solve quickly, very computationally demanding, or unsolvable. We will see unsolvable decision problems soon for context-free languages; for now, we focus on problems which can be solved.

2 Decision problems

2.1 Given a regular language L, is $L = \emptyset$?

Note that this is asking about a property of an entire language, not just a string. Thus in the formalism above our set A is the set of all regular languages, and our set B contains the empty regular languages (i.e. those which contain no strings).

Our approach will depend on how L is specified.

- If L is given to us as a DFA or an NFA N, the emptiness question for L is equivalent to the question: does N have any final states that are accessible from the start state? If yes, $L \neq \emptyset$, and if not, $L = \emptyset$. This suggests a simple algorithm: we simply use depth-first search (DFS) on the underlying transition diagram of N. We answer 'yes' if our search never reaches an accepting state, and 'no' if it does reach an accepting state.

- If L is given to us as a regular expression, we can again convert the regular expression to a DFA and continue as in the previous case.

2.2 Given a regular language L, is $L = \Sigma^*$?

This problem is easy to solve using our solution to the previous one: Given L, we can

a) Obtain a DFA accepting L

b) Convert this to a DFA accepting $\sim L$ as explained in class

c) Apply the algorithm from Decision Problem 2 to check if $\sim L = \emptyset$.

2.3 Given a pair of regular languages L_1 and L_2, is $L_1 = L_2$?

Note that this is a decision problem that can arise in practice. Every time you check if two automata or two regular expressions are equivalent, you are solving this problem.

Note that $L_1 = L_2$ iff the symmetric difference of L_1 and L_2 is empty (that is, there is no string belonging to one but not both of the languages). Formally, the symmetric difference (call it SD) of L_1 and L_2 is expressed as:

$$SD = (L_1 \cap \overline{L_2}) \cup (L_2 \cap \overline{L_1})$$

Since this expression uses only union, complement and intersection operations on L_1 and L_2, we can construct a DFA accepting SD from DFAs from L_1 and L_2. Then, we can use our algorithm for Decision Problem 2 to test for emptiness of SD.

2.4 Given a regular language L, is L infinite?

This is somewhat less intuitive than the previous problems. However, we can solve this problem using the pumping lemma. If L is regular, it has a number k as specified in the pumping property in Chapter 11 - k is sometimes called the *pumping length*. Note that we can obtain k effectively from L - express L as a DFA and let k be the number of states of this DFA. We make the following claim:

Claim 1 A regular language L is infinite iff L contains at least one string r such that $|r| \geq k$.

Proof ($\Rightarrow$) Suppose L is infinite. Then it contains strings of arbitrarily large length, so the result follows immediately.

($\Leftarrow$) Suppose L contains at least one string r such that $|r| \geq k$. By the pumping lemma, we can write r as xyz, with $x = \varepsilon, y = r$ and $z = \varepsilon$. Then, we can write y (which is equal to r) as uvw, with $|v| > 0$. By the pumping lemma, for all $i \geq 0$ $uv^iw \in L$. Thus we have a set of infinitely many strings all of which belong to L; consequently L is infinite as claimed. $\square$

Clearly, if we can give an algorithm to check whether L contains at least one string of length $\geq k$, we will be done. However, if we simply start checking ever longer strings until we find one that belongs to L, our algorithm may never terminate. We need some upper bound on the length of strings that need to be checked.

Claim 2 A regular language L contains at least one string r such that $|r| \geq k$ iff L contains at least one string r such that $2k \geq |r| \geq k$.

Proof ($\Leftarrow$) Nothing to prove.

($\Rightarrow$) Suppose for a contradiction L contains strings longer than k, but no strings r with $2k \geq |r| \geq k$. Now, consider any string $s \in L$, $|s| > 2k$. Use the pumping lemma as above, with $x = \varepsilon$, y the inital prefix of length k of s, and z the rest of s (thus $|z| > k$). Now write y as uvw in the usual way, and consider the string $xuwz$. We know $xuwz = uwz \in L$ by the pumping lemma. But we also know that $|uwz| \geq k$, since $|z| > k$. If we also have $|uwz| \leq 2k$, we have a contradiction.

Otherwise, repeat the process on the new string uwz. Again, we can use the pumping lemma to produce a shorter string that is also in L. Iterate this process as many times as necessary; since the strings we produce get shorter every time, we will end up by obtaining a string s which must belong to L and have length between k and $2k$. Thus, we are guaranteed to arrive at a contradiction. $\square$

The complete algorithm for checking whether a language is infinite is therefore:

a) Obtain the pumping length k (e.g. by constructing a DFA)

b) Check if any string r having length between k and $2k$ is in L. If yes, answer that L is infinite, if no, answer that it is finite.