Decision Problems for Finite Automata

Now we wish to examine decision problems for the sets accepted by finite automata (or the regular sets). When we tried to decide things concerning the r.e. sets we were disappointed because everything nontrivial seemed to be unsolvable. Our hope in defining finite automata as a much weaker version of Turing machines was to gain solvability at the expense of computational power. Let us see if we have succeeded. Our first result indicates that we have.

Theorem 1. *Membership is solvable for the regular sets.*

Proof. This is very easy indeed. With a universal Turing machine we can simulate any finite automaton. To decide whether it accepts an input we need just watch it for a number of steps equal to the length of that input.

So far, so good. In order to decide things a bit more intricate than membership though, we need a very useful technical lemma which seems very strange indeed until we begin to use it. It is one of the most important results in finite automata theory and it provides us with a handle on the finiteness of finite automata. It tells us that only the information stored in the states of a finite automaton is available during computation. (This seems obvious, but the proof of the following lemma points this out in a very powerful manner.)

Lemma (Pumping). *Let $M = (S, I, \delta, s_0, F)$ be a finite automaton. Then for any string x accepted by M whose length is no less than the size of S, there are strings u, v, and w (over the alphabet I) such that:*

a) $x = uvw$,
b) v is not the empty string, and
c) for all $k \geq 0$, $uv^k w \in T(M)$.

Proof. Let $x = x_1 \ldots x_n$. Suppose that x is accepted by the finite automaton M and has length n where n is not smaller than the size of S, the state set of M. Then as M processes x, it goes through the sequence of states:

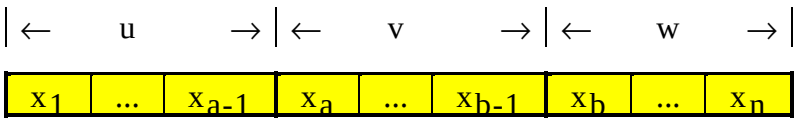
$$s_{j_1}, \ldots, s_{j_{n+1}}$$

where M is in state s_{j_i} as it reads x_i, and:

a) $s_{j_1} = s_0$,
b) for each $i \leq n$, $\delta(s_{j_i}, x_i) = s_{j_{i+1}}$, and
c) $s_{j_{n+1}} \in F$.

Since M has no more than n states (our initial assumption about the length of x not being less than the number of M's states), at least one of the states in the sequence must be repeated because there are n+1 states in the sequence. We shall assume that this repeat occurs at s_{j_a} and s_{j_b} where $a < b$.

Now let's consider the string $x_a \ldots x_{b-1}$, the portion of the input which is processed by M as it goes through the sequence of states $s_{j_a}, \ldots, s_{j_b}$. We shall say that v is this substring of x and note that since $a < b$, $v \neq \varepsilon$. Now we shall assign the remaining characters of x to u and w. The following picture illustrates this.



It is obvious that uvw = x. And, when M processes x:

a) $\delta^*(s_{j_1}, u) = \delta^*(s_0, u) = s_{j_a}$ [since $s_{j_1} = s_0$]
b) $\delta^*(s_{j_a}, v) = s_{j_b} = s_{j_a}$ [since $s_{j_b} = s_{j_a}$]
c) $\delta^*(s_0, uv) = s_{j_b} = s_{j_a}$ [same reason]
d) $\delta^*(s_{j_b}, w) = \delta^*(s_{j_a}, w) = s_{j_{n+1}} \in F$ [same again]

In other words, M enters and leaves the substring v in *the same state*. Now we shall examine exactly what this means. If we were to omit v and process uw, M would leave u in $s_{j_a} = s_{j_b}$ and finish w in $s_{j_{n+1}}$ just as before. Thus uw is in T(M). If we were to make M process uvvw then M would leave uv in $s_{j_b} = s_{j_a}$, leave uvv in s_{j_b} and finish w in the same state as before. Thus $uvvw \in T(M)$. In fact, no matter how many times we add another v between u and w, M always leaves and enters each v in s_{j_a} and therfore finishes the entire input in the same final state. Thus for any $k \geq 0$, $uv^k w \in T(M)$.

If we go back and examine our proof of the pumping lemma, we find that we can prove something a little more powerful. In fact, something that will come in handy in the future. Something which will make our lives much more pleasing. Here it is.

> **Corollary.** *The substring v of x can be forced to reside in any portion of x which is at least as long as the number of states of the automaton.*
>
> **Proof.** Just note that in any substring of x which is no shorter than the number of states, we can find a repeated state while processing. This provides a v and the proof proceeds as before.

This technical lemma (referred to as the *pumping lemma* from now on) is one of the most useful results in theoretical work involving finite automata and the regular sets. It is the major tool used to

a) detect non-regular sets, and
b) prove decision problems solvable for finite automata.

The usefulness of the pumping lemma comes from the fact that it dramatically points out one of the major characteristics of finite automata, namely that they have only a finite amount of memory. Another way to state this is to say that if a finite automaton has n states, then it *can only remember n different things*! In fact, if $\delta^*(s_0, x) = \delta^*(s_0, y)$ then the machine with the transition function δ cannot tell the difference between x and y. They look the same to the machine since they induce the same last state in computations involving them. And, if a finite automaton accepts a very long string, then chances are that this string contained repetitive patterns.

Our first use of the pumping lemma will be to present a non-regular set. This is the favorite example for computer science theorists and illustrates the method almost always used to prove sets non-regular.

> **Theorem 2**. *The set of strings of the form $\{0^n1^n\}$ for any $n \geq 0$ is not a regular set.*
>
> **Proof.** Assume that the set of strings of the form 0^n1^n is a regular set and that the finite automaton $M = (S, I, \delta, s_0, F)$ accepts it. Thus every string of the form 0^k1^k for k larger than the size of the state set S will be accepted by M.
>
> If we take one of these strings for some $k > |S|$ then the pumping lemma assures us that there are strings u, v, and w such that:

a) $uvw = 0^k1^k$,
b) $v \neq \varepsilon$, and
c) for all $n \geq 0$, $uv^nw \in T(M)$.

Invoking the corollary to the pumping lemma assures us that the substring v can be in the midst of the 0's if we wish. Thus $v = 0^m$ for some (nonzero) $m \leq k$. This makes $uw = 0^{k-m}1^k$ and the pumping lemma states that $uw \in T(M)$. Since uw is not of the form 0^n1^n we have a contradiction. Thus our assumption that strings of the form 0^n1^n can be accepted by finite automata and be regular set is incorrect.

As we mentioned earlier, almost *all* of the proofs of non-regularity involve the same technique used in the proof of the last theorem. One merely needs to examine the position of v in a long string contained in the set. Then either remove it or repeat it several times. This will always produce an improper string!

Next, we shall use the deflation aspect of the pumping lemma in order to show that emptiness is solvable for regular sets.

Theorem 3. *If a finite automaton accepts any strings, it will accept one of length less than the size of its state set.*

Proof. Let M be a finite automaton which accepts the string x and that the length of x is no less than the size of M's state set. Assume further that M accepts no strings shorter than x. (This is the opposite of our theorem.)

Immediately the pumping lemma asserts that there are strings u, v, and w such that $uvw = x$, $v \neq \varepsilon$, and $uw \in T(M)$. Since $v \neq \varepsilon$, uw is shorter than $uvw = x$. Thus M accepts shorter strings than x and the theorem follows.

Here is a sequence of corollaries which follow from the last theorem. In each case the proof merely involves checking membership for all strings of length less than the size of the state set for some finite automaton or the machine which accepts the complement of the set it accepts. (Recall that the class of regular sets, namely those accepted by finite automata is closed under complement.)

Corollary (Emptiness Problem)**.** *Whether or not a finite automaton accepts anything is solvable.*

Corollary (Emptiness of Complement)**.** *Whether or not a finite automaton rejects anything is solvable.*

Corollary. *Whether or not a finite automaton accepts everything is solvable.*

Another cautionary note about these decision problems is in order. It is quite refreshing that many things are solvable for the regular sets and it is wonderful that several solvable decision problems came immediately from one lemma and a theorem. But, it is very *expensive* to try and solve these problems. If we need to look at all of the input strings of length less than the size of the state set (let's say that it is of size n) then we are looking at almost 2^n strings! Imagine how long this takes when n is equal to 100 or so!

Flushed with success we shall attempt (and succeed) at another decision problem which is unsolvable for the class of recursively enumerable sets.

Theorem 4. *Whether a regular set is finite is solvable.*

Proof Sketch. We know that if a finite automaton accepts any strings at all then some will be of length less than the size of the state set. (This does not help directly, but it gives a hint as to what we need for this theorem.) The deletion aspect [uw Y T(M)] of the pumping lemma was used to prove this. Let's use the inflation aspect [uv^nw Y T(M)] of the lemma to look for an infinite set.

For starters, if we were to find a string accepted by a finite automaton M which was longer than or equal to the size of it's state set, we could use the aforementioned inflation aspect of the pumping lemma to show that machine M must accept an infinite number of strings. This means that:

a finite automaton accepts only strings of length less than the size of its set of states, if and only if it accepts a finite set.

Thus, to solve the finiteness problem for M = (S, I, δ, s_0, F), we need to determine whether or not:

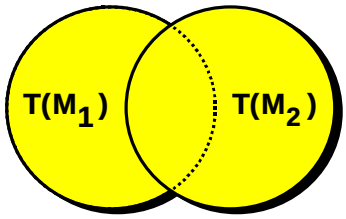
$$T(M) - \{\text{strings of length} < |S|\} = \varnothing.$$

A question now arises as to how many inputs we must examine in order to tell if M will accept an input longer than or equal to the size of its state set. The answer is that we only need to consider input strings up to twice the size of the state set. (The proof of this is left as an exercise, but it is much the same as the proof of the emptiness problem.) This ends our proof sketch.

The next decision problem is included because it demonstrates another technique; using old problems to solve a new one. We see this in the areas of unsolvability and complexity when we use reducibility to show problems unsolvable or intractable. Here we show that the problem of set equivalence is reducible to the emptiness problem and thus solvable.

Theorem 5. *Whether two regular sets are identical is solvable.*

Proof. Let's take two finite automata (M_1 and M_2) and examine a picture of the sets they accept.



If the intersection is the same as both sets then indeed they are identical. Or, on the other hand, if the areas outside the intersection are empty then both sets are identical. Let's examine these outside areas.

$T(M_1) \cap \overline{T(M_2)}$ $\qquad$ $T(M_2) \cap \overline{T(M_1)}$

The picture on the left represents the set accepted by M_1 and rejected by M_2 while that on the right is the set which M_2 accepts and M_1 rejects.

If these two areas (or sets) are empty then the sets accepted by M_1 and M_2 are exactly the same. This means that the equivalence problem for $T(M_1)$ and $T(M_2)$ is exactly the same as the emptiness problem for:

$$[\, T(M_1) \cap \overline{T(M_2)} \,] \cup [\, T(M_2) \cap \overline{T(M_1)} \,]$$

So, if we can solve the emptiness problem for the above set, then we can solve the equivalence problem for $T(M_1)$ and $T(M_2)$. Since the regular sets are closed under union, complement, and intersection; the above set is a regular set. And, we know that emptiness is solvable for the class of regular sets.

Here is yet another cautionary note. The last proof was quite slick and elegant, but one should not do the construction in an attempt to prove that two finite

automata accept the same set. We know that the complexity of any algorithm which attempts to do this is very large since it would take a while to do emptiness for a machine formed from the unions and intersections of four different machines.

We shall close this section on a cheerful note with an intuitive, imprecise statement about finite automata and the class of regular sets.

> **Folk Theorem.** *Just about everything concerning finite automata or the regular sets is solvable!*