ECS 120 Problem Set 1

Hardy Jones
999397426
Professor Rogaway
Spring 2014

Some problems were discussed with Brandon Maynard.

Problem 1 We can gain some insight by looking at the first few n-digit palindromes.

n	elements	$\|n\|$
1	1,2,3,4,5,6,7,8,9	9
2	11,22,33,44,55,66,77,88,99	9
3	101,202,303,404,505,606,707,808,909 111,212,313,414,515,616,717,818,919 ⋮ 191,292,393,494,595,696,797,898,999	90
4	1001,2002,3003,4004,5005,6006,7007,8008,9009 1111,2112,3113,4114,5115,6116,7117,8118,9119 ⋮ 1991,2992,3993,4994,5995,6996,7997,8998,9999	90

If we continue in this way we see that the number of palindromes increases by 10 for every 2 digits added.

We can see a recurrence relation here:

$$D_1 = 9$$
$$D_2 = 9$$
$$D_n = D_{n-2} \cdot 10, \text{ for n} > 2$$

From this we can manipulate it to a closed form:

$$D_n = 9 \cdot 10^{\lfloor \frac{n-1}{2} \rfloor}$$

We can prove this by induction.

Proof. Base Cases:

$n = 1$

$$D_1 = 9 \cdot 10^{\lfloor \frac{1-1}{2} \rfloor} = 9 \cdot 10^0 = 9$$

$n = 2$

$$D_2 = 9 \cdot 10^{\lfloor \frac{2-1}{2} \rfloor} = 9 \cdot 10^0 = 9$$

Inductive Hypothesis:

$$D_n = 9 \cdot 10^{\lfloor \frac{n-1}{2} \rfloor}$$

Inductive Case:

$$\begin{aligned} D_{n+1} &= D_{n-1} \cdot 10 \\ &= 9 \cdot 10^{\lfloor \frac{(n-1)-1}{2} \rfloor} \cdot 10 \\ &= 9 \cdot 10^{\lfloor \frac{n-2}{2} \rfloor} \cdot 10 \\ &= 9 \cdot 10^{\lfloor \frac{n}{2} - \frac{2}{2} \rfloor} \cdot 10 \\ &= 9 \cdot 10^{\lfloor \frac{n}{2} - 1 \rfloor} \cdot 10 \\ &= 9 \cdot 10^{\lfloor \frac{n}{2} - 1 + 1 \rfloor} \\ &= 9 \cdot 10^{\lfloor \frac{n}{2} \rfloor} \end{aligned}$$

Thus, we have proved that our formula is correct.

□

Now, we can calculate $D_{20} = 9 \cdot 10^{\lfloor \frac{20-1}{2} \rfloor} = 9 \cdot 10^9 = 9000000000$

Problem 2 Let's begin by enumerating some of the first few strings in lexographic order.

w_1	ε
w_2	0
w_3	1
w_4	00
w_5	01
w_6	10
w_7	11
w_8	000

Interestingly, this looks like the binary representation of each n in w_n without the leading digit. In the case of $n = 1$, removing the leading (only) digit leaves the empty string.

So we have a mapping: $n \in \mathbb{N} \to w_n$.

This means that $w_{1234567}$ can be easily computed. We just find the binary representation for 1234567_{10} and remove the first digit.

$$1234567_{10} = 100101101011010000111_2$$

Removing the first digit, we end up with: 00101101011010000111

Problem 3 (a) We can take some intuition from linear algebra for this. If we view each of the strings as a vector, concatenation as vector addition, and the number of characters in a string as the magnitude of the vector, then we can see that there are only

two linearly independent vectors in L. For simplicity, we take a and b to be these two independent vectors.

The question then becomes, how many different ways can we arrange these two vectors so that the resulting vector has magnitude 10?

Let's enumerate the first few lengths.

length	elements
0	ε
1	a, b
2	aa, ab, ba, bb
3	$aaa, aab, aba, abb, baa, bab, bba, bbb$

So it looks like the number of elements is 2^l, where l is the length of the string.

This means that there are $2^{10} = 1024$ strings of length 10 from this language.

(b) We can start by enumerating some lengths

length	elements
0	ε
1	a
2	aa, bb
3	aaa, abb, bba
4	$aaaa, aabb, abba, bbaa, bbbb$
5	$aaaaa, aaabb, aabba, abbaa, abbbb, bbaaa, bbabb, bbbba$

Here it looks like the number of elements is $fib(l)$, where l is the length of the string and $fib(l) = fib(l-1) + fib(l-2)$.

This means there are $fib(10) = 89$ strings of length 10 from this language.

Problem 4 We can use the division formula here: $s = qN + r$,

where $s \in \mathcal{S}, N = 314159265359, q, r \in \mathbb{Z}, q, r \geq 0$.

Since $\mathcal{S}$ is infinite, we have more elements in $\mathcal{S}$ than there are congruence classes modulo N. So, by the pigeon hole principle, at least two elements in $\mathcal{S}$ must have the same remainder when divided by N.

Using the division formula:

$$s_1 = q_1 N + r \text{ and } s_2 = q_2 N + r$$

If we manipulate and substitute for r:

$$\begin{aligned} s_1 &= q_1 N + (s_2 - q_2 N) \\ s_1 - s_2 &= q_1 N - q_2 N \\ &= (q_1 - q_2) N \end{aligned}$$

Thus, since $q_1 - q_2$ is an integer, $s_1 - s_2$ is a multiple of N.

More succinctly, the difference of two elements of $\mathcal{S}$ is a multiple of N.

Problem 5 (a) Given such an oracle, the first step is to check if the polynomial has an integer root. If it doesn't have an integer root, we're done, so just report *No Root*. In the other case, assuming our oracle is truthful, we need to find an integer root for the given polynomial.

So we have to start enumerating all possible choices for the variables of the polynomial. Since we want this algorithm to be sufficiently general, we'll have to find how many variables are in the polynomial first. Once we know this we can start enumerating the possibilities.

One way to handle this is to take a cue from linear algebra and view the problem as operating on polynomial space. Then, for however many distinct variables we have in our polynomial, we only need to find all possible enumerations of vectors in that space. This can be simplified by starting at the zero vector and working outwards in all dimensions of the vector space, using the standard basis.

For example, with a single variable polynomial, we have one variable to check, so we can enumerate the options: $\{0, 1, -1, 2, -2, 3, -3, \dots\}$. With a 2-variable polynomial we have two variables to check, so we can enumerate the options: $\{(0,0), (0,1), (1,0), (0,-1), (-1,0), (1,1), \dots\}$. This way we are guaranteed to try every possible choice of integers for the polynomial.

Since our oracle has told us that there is some integer solution, and we trust our oracle, we are guaranteed to at least halt at some point.

(b) We only know that $s(n)$ halts and is computable because our oracle has told us so. However, the oracle has not told us how long it will take to halt. This oracle has not solved the halting problem in general, merely for our specific case. Since we still have the halting problem around, we cannot make any assumptions about a function that takes more algorithmic steps than our oracle-based function. Thus we do not know that a function $S(n) \geq s(n), \forall n$ will ever halt. So any function S cannot be computed in general, and thus no algorithm can exist for S.

Problem 6 (a) True. $\varnothing^* = \{\varepsilon\}$ is a set containing the empty string.

(b) False. Languages are sets containing strings, ε is not a set.

(c) False. $\varnothing^*$ is finite, and given a universe $U = a$, $\overline{\varnothing^*}$ is finite as well.

(d) True. Given a universe $U = \{1^i | i \in \mathbb{N}\}$, $L = \{1^e | e \in \mathbb{N}, e \text{ is even}\}$ is infinite. Its complement $\overline{L} = \{1^o | o \in \mathbb{N}, o \text{ is odd}\}$ is also infinite.

(e) False. There are some reals without a finite representation. These numbers cannot be strings, since strings must be finite. Thus, no languages can be made of the reals.

(f) True. $\varnothing$ is a subset of all languages.

(g) False. $\varnothing^* = \{\varepsilon\}$ is finite.

(h) False. Assuming some infinite language L, $L \circ \varnothing = \varnothing$, which is finite.

(i) False. This is a rewording for the Kleene closure, so it cannot be a proper subset of itself.