CS481F01 Prelim 1

A. Demers

5 Oct

1. Which of the following sets is (are) regular? Justify your answers briefly.

(a) $\{ \ 0^{i^2} \mid i \geq 0 \ \}$
(b) $\{ \ 0^{i^2} \mid i \geq 0 \ \}^*$
(c) $\{ \ 0^i 1^j \mid i \equiv j \pmod{11} \ \}^*$
(d) $\{ \ w\$x \mid w, x \in \{0,1\}^* \ \wedge \ (\sharp 0(w) \ = \ \sharp 1(x)) \ \}$
(e) $\{ \ 0^i w 1^i \mid w \in \{0,1\}^* \ \wedge \ i \geq 0 \ \}$
(f) $\{ \ wx \mid w, x \in \{0,1\}^* \ \wedge \ (\sharp 0(w) \ = \ \sharp 1(x)) \ \}$
(g) the set of all syntactically correct Java programs
(h) the text of question 1 of this prelim
(i) $L_{A,S} \ = \ \{ \ x \mid (\exists y \in A) \ xy \in S \ \}$

For part (i), assume $A \subseteq \{0,1\}^*$ is an arbitrary regular set and $S \subseteq \{0,1\}^*$ is an arbitrary (not necessarily regular) set. If the given set is necessarily regular for all A and S, give a convincing argument that this is true. Otherwise, give a counterexample.

(answer a) Not regular. The set $\{ \ i^2 \mid i \geq 0 \ \}$ is not ultimately periodic.

(answer b) Regular. The set $A \ = \ \{ \ 0^{i^2} \mid i \geq 0 \ \}$ contains $0^0 = \epsilon$ and $0^1 = 0$; consequently $A^* \ = \ \{0\}^*$, which is regular.

(answer c) Regular. Rewrite as $\{ \ 0^i 1^j \mid (i \pmod 1)1) = (j \pmod 1)1) \ \}$, and observe that there are only 11 distinct values for $(i \pmod 1)1)$, and these can be remembered in the state of a FA.

(answer d) Not regular, proved in lecture by inverse homomorphism.

(answer e) Regular. The language includes $\{0^0 w 1^0 \mid w \in \Sigma^*\}$ which is all of Σ^*.

(answer f) Regular. This one is tricky. Claim any $z \in \Sigma^*$ can be written in this form, by strong induction on $|z|$. The basis is trivial. For the inductive step, there are two cases: $z = z'0$ or $z = z'1$. In the first case, use the i.h. to write $z' = wx$ where $\sharp 0(w) = \sharp 1(x)$, and observe that $z = w(x0)$ has the required property. In the second case, if z' consists entirely of $1's$ the result is immediate. So write $z' = y0z''$ where y consists entirely of $1's$. Use the i.h. to write $z'' = wx$ and observe that $z = (y0w)(x1)$ has the required property.

(answer g) Not regular. For example, use a homomorphism to map this to $\{0^i 1^i\}$.

(answer h) Regular. It may not seem so, but this question is finite.

(answer i) Not regular. Let $A = \{\epsilon\}$ and let S be any non-regular set.

2. In the introduction to this course we argued that we could always model function evaluation by language recognition, representing a function by the language of its argument-result pairs. Here we examine this claim more critically.

Let $\Sigma = \{0, 1\}$, and let f be a function from Σ^* to Σ^*.

A language $L \subseteq (\Sigma \cup \{\$\})^*$ is said to *represent f by pairs* if

$$L \quad = \quad \{\ x\$y \mid x, y \in \Sigma^* \ \wedge\ y = f(x)\ \}$$

A language $L \subseteq \Gamma^*$ is said to *represent f by homomorphisms* if there exist homomorphisms g and h from Γ^* to Σ^* such that

$$y = f(x) \qquad \text{iff} \qquad (\exists z \in L)((x = g(z)) \wedge (y = h(z)))$$

Now, let $P_p(f)$ be the proposition "there is some regular language A that represents f by pairs," and let $P_h(f)$ be the proposition "there is some regular language B that represents f by homomorphisms,'

(a) Does $P_p(f)$ imply $P_h(f)$?

(answer a) Yes. Use the inverse of the homomorphism

$$u(0) = u(a) = 0 \qquad u(1) = u(b) = 1 \qquad u(\$) = \$$$

then intersect with $L((0{+}1)^*\$(a{+}b)^*)$. The resulting language is clearly regular, and by using

$$\begin{array}{lll} g(0) = 0 & g(1) = 1 & g(a) = g(b) = g(\$) = \epsilon \\ h(a) = 0 & h(b) = 1 & h(0) = h(a) = h(\$) = \epsilon \end{array}$$

clearly represents f by homomorphism.

(b) Does $P_h(f)$ imply $P_p(f)$?

(answer b) No. The identity function $f(x) = x$ is a counterexample. Clearly Σ^* represents the identity function using

$$g(0) = h(0) = 0 \qquad\qquad g(1) = h(1) = 1$$

But the (only) language representing the identity function by pairs is $\{w\$w \mid w \in \Sigma^*\}$, which is not regular.

3. Consider the following languages of balanced parentheses:

$L_{()}$ is the set of strings of balanced parentheses nested arbitrarily deeply – for example,

$$() \qquad (()) \qquad ()() \qquad (()(())(()())) \qquad ((((((((())))))))) \ldots$$

are all strings in $L_{()}$.

$L^k_{()}$ is the set of strings of balanced parentheses nested no more than k deep. For example, the string $(()(()))$ is in mbox$L^3_{()}$ but not mbox$L^2_{()}$.

$L_{()[]}$ is the set of strings of balanced parentheses of two different types, () and []. We require different kinds of parentheses to be properly matched, so for example the string [(()())[[]]] is in $L_{()[]}$, but the string [(()()]) is not.

$L^{j,k}_{()[]}$ is the set of strings of balanced parentheses of two types, with the nesting of () limited to j, and the nesting of [] limited to k. The nesting depth is counted separately for the two kinds of parentheses, so for example the string [[(([]))]] is in $L^{2,3}_{()[]}$.

Believe it or not, this is mostly a Myhill-Nerode question.

(a) Describe the equivalence classes of the relations

$$\equiv_{L_{()}}, \qquad \equiv_{L_{()[]}}, \qquad \equiv_{L^k_{()}}, \qquad \equiv_{L^{j,k}_{()[]}}$$

Do this informally, but in enough detail to enable a reader to decide whether $[x]_\equiv \; = \; [y]_\equiv$ for arbitrary strings x and y.

(answer a) For any language L, the equivalence classes of $\equiv_L$ are sets of strings that behave equivalently under extension; i.e.,

$$x \equiv_L y \;\; \text{iff} \;\; (\forall z)(xz \in L \Leftrightarrow yz \in L)$$

For our parenthesis languages, a string xz is in L iff z "closes" all the open – that is, unmatched – (and [characters in x. So you can think of z as the string in $(`)' + `]')^*$ that matches all the unmatched left bracket symbols of x. Any y with the same sequence of unmatched bracket symbols as x will also match z, hence be equivalent to x. We may as well choose the shortest such y, which consists entirely of (and [characters, and use this as the canonical representative of the equivalence class. Specifically:

For $L_{()}$ the equivalence classes correspond to (arbitrary-length) strings in $\{(\}^*$. For $L^k_{()}$ the classes correspond to strings in $\{(\}^*$ of length at most k.

For $L_{()[]}$ the equivalence classes correspond to (arbitrary-length) strings in $\{(, [\}^*$. For $L^{j,k}_{()[]}$ the classes correspond to strings in $\{(, [\}^*$ containing at most j (characters and a most k [characters. For a given pair $\langle j, k\rangle$ there are many such strings, and the order of symbols is important. For example, "(([]))" is in $L_{()[]}$, but "([(]))" is not.

There is, in addition, a *single* equivalence class containing all strings that have errors. All such strings are equivalent, since there is no way to correct an error by extending the string.

The non-error equivalence classes for $L_{()}$ can also be though of as natural numbers. The equivalence class of a string corresponds to the number of unclosed parentheses it contains. For example,

$$(((\qquad (()(()(() \qquad ()()()(()((\ldots$$

are in equivalence class 3.

(b) Construct a *minimum state* DFA recognizing $L^4_{()}$. A state diagram is sufficient. Include *all* the states.

(answer b) The states are the equivalence classes of $\equiv_{L^4_{()}}$, that is,

$$\{\ q_\epsilon, q_(, q_{((}, q_{(((}, q_{((((}, q_{\mathbf{err}}\ \}$$

The transition function is

$$\begin{array}{lll} \delta(q_{(^i}, `(') &=& q_{(^{i+1}} \qquad 0 \le i < 4 \\ \delta(q_{(^i}, `)') &=& q_{(^{i-1}} \qquad 0 < i \le 4 \\ \delta(q, a) &=& q_{\mathbf{err}} \qquad \text{otherwise} \end{array}$$

The only final state is q_ϵ. It should be clear that this is the minimal machine, constructed using the Myhill-Nerode relation from part (a).

(c) Construct a minimum state DFA recognizing $L^{2,1}_{()[]}$. Give a state diagram, and describe the machine's operation well enough for us to understand it.

(answer c) We proceed in a similar fashion. Now the states of our machine are

$$\{\ q_{\mathbf{err}}\ \} \cup \{\ q_w \mid w \in \{`(', `['\}^* \ \wedge\ \sharp_((w) \le 2 \ \wedge\ \sharp_[(w) \le 1\ \}$$

The transition function is

$$\begin{array}{lll} \delta(q_w, `(') &=& q_{w(} \qquad \sharp_((w) < 2 \\ \delta(q_{w(}, `)') &=& q_w \qquad \sharp_((w) < 2 \\ \delta(q_w, `[') &=& q_{w[} \qquad \sharp_[(w) < 1 \\ \delta(q_{w[}, `]') &=& q_w \qquad \sharp_[(w) < 1 \\ \delta(q, a) &=& q_{\mathbf{err}} \qquad \text{otherwise} \end{array}$$

The only final state is q_ϵ. Again, this is just the minimal machine, constructed using the Myhill-Nerode relation from part (a).

(d) How many states are there in a minimum state DFA recognizing $L^{m,1}_{()[]}$, expressed as a function of m? Explain your answer. If you want to show off, give a formula for the number of states in the minimum state DFA recognizing $L^{m,n}_{()[]}$ as a function of m and n.

(answer d) Since at most one [is allowed, we can express the answer by brute force: for each possible number i of (characters, there are $i+2$ possibilities: $i+1$ possible positions of a [character, or no [character at all. This yields

$$N \quad = \quad \sum_{i=0}^{m}(i+2) \quad = \quad 2m+2+\sum_{i=0}^{m} i \quad = \quad 2m+2+\frac{m(m+1)}{2}$$

Note this is quadratic in m.

To show off, observe that the number of ways to construct a string of i (characters and j [characters is just the number of ways to choose j positions (the square brackets) out of $i+j$ positions (all the characters). This is just the binomial coefficient

$$\binom{i+j}{j}$$

leading to the expression

$$N \quad = \quad \sum_{i=0}^{m}\sum_{j=0}^{n} \binom{i+j}{j}$$

which you may feel free to simplify.

(e) For any m and n, show (inductively) how to construct a regular expression $R^{m,n}$ that generates $L_{()[]}^{m,n}$.

answer e This is similar to the technique we used in class extended to handle two kinds of parentheses. We'll define $R^{m,n}$ inductively by

$$\begin{array}{rcl} R^{0,0} & = & \epsilon \\ R^{i+1,j} & = & R^{i,j}\left\{(R^{i,j})R^{i,j}\right\}^{*} \\ R^{i,j+1} & = & R^{i,j}\left\{[R^{i,j}]R^{i,j}\right\}^{*} \end{array}$$

Since there is no requirement that m and n be equal, we need to induct separately on i and j.