Homework 4
Due Oct 29, 2007

Prof. Amit Chakrabarti
Computer Science Department
Dartmouth College

Please think carefully about how you are going to organise your answers *before* you begin writing. Make sure your answers are complete, clean, concise and rigorous.

1. Design pushdown automata (PDAs) for each of the following languages:

   1.1. $\{x \in \{0,1\}^* : x \text{ contains more } 0\text{'s than } 1\text{'s}\}$. [10 points]

   1.2. $\{x \in \{0,1\}^* : |x| \text{ is odd and the middle symbol of } x \text{ is a } 0\}$. [10 points]

   1.3. $\{x \in \{0,1,2\}^* : N_0(x) = N_1(x) \text{ or } N_1(x) = N_2(x)\}$. Here $N_i(x)$ denotes the number of occurrences of the character i in the string x. [10 points]

2. This set of problems develops some theory around regular languages that we have not discussed in class. Here is your chance to build your theoretical skills by doing some basic independent work. For these problems you will need to be familiar with the notions of *equivalence relations*, discussed in Section 0.2 of your textbook, which was required reading prior to Lecture #3.

   2.1. For a language A over alphabet Σ, define the relation "$\equiv_A$" on strings in Σ^* as follows: "$x \equiv_A y$" means

   $$\forall\, w \in \Sigma^* \left(xw \in A \iff yw \in A\right).$$

   In other words, $x \equiv_A y$ means that any string w which extends x to a string in A also extends y to a string in A, and vice-versa. Note that x and y are not necessarily strings from A; they are strings from Σ^*.

   For example, consider the language $B = (01)^*$ over the alphabet $\{0,1\}$. We have $010 \not\equiv_B 011$ because taking $w = 1$, we get $010w = 0101 \in B$ whereas $011w = 0111 \notin B$. Similarly, $01 \not\equiv_B 101$ (take $w = \varepsilon$). However, $010 \equiv_B 0101010$ (convince yourself of this).

   Prove that "$\equiv_A$" is an equivalence relation. [5 points]

   2.2. The relation "$\equiv_A$" is called the *left equivalence relation* of the language A. An equivalence relation on a set partitions the set into disjoint subsets called equivalence classes in the following way: two elements belong to the same class iff they are related by the equivalence relation. Thus, "$\equiv_A$", which is a relation on Σ^*, partitions Σ^* into equivalence classes: these are called the *left equivalence classes* of the language A.

   For example, consider the language $C = \{x \in \{0,1\}^* : |x| \text{ is even}\}$ over the alphabet $\{0,1\}$. Convince yourself that any two even-length strings are related by "$\equiv_C$" as are any two odd-length strings. Also, no odd-length string is related by "$\equiv_C$" to an even-length string. Since every string in $\{0,1\}^*$ is either odd-length or even-length, we see that C has exactly two left equivalence classes: (1) odd-length strings, i.e., $\{0,1\}^* - C$, and (2) even-length strings, i.e., C.

   Similarly, convince yourself that the left equivalence classes of $B = (01)^*$ over the alphabet $\{0,1\}$ are: (1) B, (2) $(01)^*0$, and (3) $\{0,1\}^* - (B \cup (01)^*0)$.

   Describe the left equivalence classes of the language $L_1 = \{a, aa, aaa, b, ba, baa\}$ over the alphabet $\{a,b\}$. You will find that there are 5 equivalence classes and that exactly one of the 5 is infinite. [5 points]

   2.3. Describe the left equivalence classes of the language $L_2 = a^*b^*c^*$ over the alphabet $\{a,b,c\}$. [5 points]

2.4. Describe the left equivalence classes of the language $L_3 = (ab \cup ba)^*$ over the alphabet $\{a, b\}$.
[10 points]

2.5. Describe the left equivalence classes of the language $L_4 = \{0^n 1^n : n \geq 0\}$ over the alphabet $\{0, 1\}$.
[10 points]

2.6. For a language A over alphabet Σ and a string $x \in \Sigma^*$, let $[x]_A$ denote the left equivalence class (of A) to which x belongs. For instance, if the sets $\{01, 101, 1110\}$ and $\{\varepsilon, 0101, 10001, 11\}$ are two of the left equivalence classes of some hypothetical language B, then

- $[01]_B$ denotes the set $\{01, 101, 1110\}$,
- $[1110]_B$ also denotes this same set,
- $[\varepsilon]_B$ and $[11]_B$ both denote the set $\{\varepsilon, 0101, 10001, 11\}$,
- and so on.

Thus, as you can see, there is rarely one unique way to write a left equivalence class as $[x]_A$: there are usually many choices for x. These choices are called *representatives* of the equivalence class.

Prove that for any $x \in \Sigma^*$ and $a \in \Sigma$, the class $[xa]_A$ is completely determined by the class $[x]_A$ and the alphabet symbol a; i.e., prove that the particular x we pick as a representative of $[x]_A$ is immaterial.
[5 points]

2.7. Suppose a language A over alphabet Σ has finitely many left equivalence classes: $[x_1]_A$, $[x_2]_A, \ldots, [x_n]_A$, for some $n \geq 1$. Prove that A is regular! [10 points]

2.8. Wasn't that cool?

Now, let A be a regular language over the alphabet Σ. Prove that A has only finitely many distinct left equivalence classes. The function $\hat{\delta}$ we defined in class (and in the Oct 12 lecture notes) might be useful in writing a formal rigorous proof. [10 points]

2.9. Using one or more of the results you proved above, give an alternate proof that the language $\{0^n 1^n : n \geq 0\}$ is not regular. [5 points]

2.10. Using left equivalence classes, give an alternate proof that $\{x \in \{0, 1\}^* : x \text{ is a palindrome}\}$ is not regular.
[5 points]

Challenge Problems

CP4: Intuitively, the smaller the number of states of a DFA, the more efficient it is. A DFA M is said to be *minimal* if there does not exist a DFA M' with fewer states than M such that $\mathcal{L}(M') = \mathcal{L}(M)$. You have probably wondered how to ascertain that a DFA you have designed is minimal.

Prove that, if L is a regular language, the number of distinct left equivalence classes of L equals the number of states of a minimal DFA that recognizes L. Hence, describe an algorithm to "minimize" a given DFA, i.e., given a description of a DFA as input, the algorithm should output the description of a minimal DFA that is equivalent to the one given as input.

Your algorithm doesn't have to be very efficient. But, for extra extra credit, and to show off your CS 25 tricks, describe an *efficient* algorithm, i.e., one that runs *fast*.

Homework 4
Due Oct 29, 2007

Prof. Amit Chakrabarti
Computer Science Department
Dartmouth College

Hint for Problem 2.7

This hint is a bit of a spoiler, so I've put it here, and upside down, so that some of you can experience the joy of coming up with the idea by yourself. Create a DFA for A whose states *are* the left equivalence classes of A. Be careful when describing the transition function! Remember, an equivalence class can have several (maybe infinitely many) different representatives.