Mehryar Mohri
Speech Recognition
Courant Institute of Mathematical Sciences
Homework assignment 1 – Solution
Due: October 22, 2008

A. Finite automata, regular expressions

[40 points]

1. [5 points] Give regular expressions describing the following languages:

   (a) The set of strings of $\{a, b\}^*$ starting with a and ending with b.
   $a\Sigma^* b$, where $\Sigma = \{a, b\}$

   (b) The set of strings of $\{a, b\}^*$ containing two consecutive a's.
   $\Sigma^* aa\Sigma^*$.

   (c) The set of strings of $\{a, b\}^*$ containing exactly two a's.
   $\Sigma'^* a\Sigma'^* a\Sigma'^*$, where $\Sigma' = \Sigma - \{a\} = \{b\}$.

2. [5 points] Construct the minimal deterministic automaton of $\Sigma^* abaab$ for $\Sigma = \{a, b\}$.

   You can construct this automaton by determinizing and minimizing a non-deterministic automaton representing $\Sigma^* abaab$ (Figure 1(a)), or use failure transitions, as discussed in class. Figure 1(b) shows the resulting automaton. The automaton has exactly $n_s = |abaab| + 1$ states and $n_s \times |\Sigma|$ transitions, since it is complete.

   Describe how you could use this automaton to search in a text for the occurrences of $abaab$.

   An occurrence of $abaab$ in text t corresponds to a prefix of t ending with $abaab$, that is an element of $\Sigma^* abaab$. The automaton constructed accepts exactly all of these prefixes, and thus locates the occurrences of $abaab$.

   Read the text with this automaton, i.e. take transitions according to the symbols found in text. By definition of the automaton, every time a final state is reached, you have just read $abaab$ and thus found an occurrence of that string.

   [15 points] Let $w \in \Sigma^*$ be a string over the finite alphabet Σ. Show that the minimal automaton accepting $\Sigma^* w$ has exactly $|w| + 1$ states. What is its number of transitions?

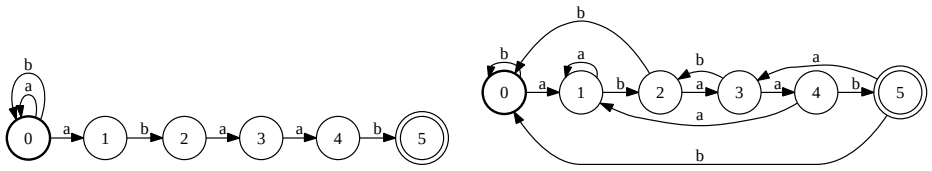

Figure 1: (a) Non-deterministic automaton representing $\Sigma^* abaab$. (b) Minimal deterministic automata representing the same regular expression.

Let p and q be the states reached by reading from the initial state of $M(w)$ two distinct prefixes of w. p and q cannot be equivalent since there are distinct suffixes of w that can be read from them to reach a final state. Thus, $p \neq q$ and $M(w)$ must have a distinct state associated to each prefix of w ($|w| + 1$ states).

The inspection of the automaton of Figure 1(b) helps determining the definition of the minimal deterministic automaton $M(w)$ accepting $\Sigma^* w$ for an arbitrary $w \in \Sigma^*$.

$M(w)$ has exactly $|w| + 1$ states, each corresponding to a distinct prefix of w read form the initial state. We can identify each state with the corresponding prefix ϵ or $w_1 \cdots w_k$, for $w = w_1 \cdots w_m$. The initial state is ϵ and the final state w. The destination state of the transition labeled with a leaving $w_1 \cdots w_k$ is the longest suffix of $w_1 \cdots w_k a$ that is a prefix of w. It is straightforward to verify that this automaton accepts exactly $\Sigma^* w$: the state reached by reading a string x is the longest suffix of x prefix of w.

3. [15 points] Give a deterministic automaton accepting the numbers x written in base 2 such that $x \equiv 3 \bmod 4$. How can you generalize the result (different base, different residue than 3, or different divisor than 4)? (Hint: assign one state to each possible rest of the division of x by 4).

   Let the alphabet Σ be defined by $\Sigma = \{0, \ldots, B-1\}$ where B is the base considered. To represent the numbers x in that base such that $x \equiv u \bmod v$, create a distinct state identified with each rest of the division, $Q = \{0, \ldots, v-1\}$. 0 is the initial state and $u \bmod v$ the final state. The destination state of the transition from $q \in Q$ labeled with a is $R(q)a \bmod q$, where $R(q)$ is the representation of q in base B. It is straightforward to verify that this automaton accepts exactly $\{x \colon x \equiv u \bmod v\}$. Figure 2 shows that automaton for the special case where $B = 2$, $u = 3$, and $v = 4$.

B. Finite-State transducers, weighted automata, rational power series

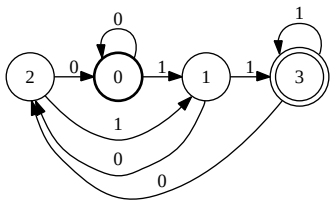

Figure 2: Finite automaton accepting numbers x written in base 2 such that $x \equiv 3 \bmod 4$.

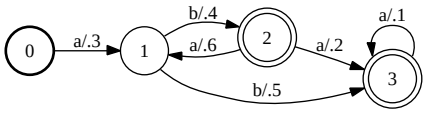

Figure 3: Weighted automaton over $(\mathbb{R}, +, \cdot, 0, 1)$.

[20 points]

1. [5 points] Give a *weighted regular expression* (rational power series) representing the weighted automaton of Figure 3 .

   $.12ab(.24ab)^*(\epsilon + .2a(.1a)^*) + .15a(.24ba)^*b(.1a)^*$

2. [15 points] Give a sequential transducer giving the result of the division of x by y (y is given, x varies). What is the number of states and transitions of that transducer. Could they be reduced? (Hint: see exercise A.3.).

   As in exercise A.3, associate a state to each rest of the division. The sequential transducer can be constructed in a straightforward way in general as in the special case of x div 3 in base 2 shown in Figure 4.

C. Algorithms, software library

The questions in this section should be answered by using the command-line utilities of the FSM or OpenFst library, except from the use of some simple scripts. The answers should be justified.

[40 points]

1. Download the CMU pronunciation dictionary from

   `http://www.cs.nyu.edu/~mohri/asr08/cmu_shuffled_dict.txt`

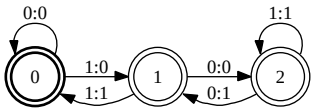

Figure 4: Sequential transducer in base 2 outputting x div 3 for input x.

This contains all the lines of the original CMU dictionary but the entries have been sorted in random order.

(a) [15 points] Create a pronunciation lexicon L_1 for the first half of the dictionary. Some words admit multiple pronunciations, include all pronunciations. Use the closure of the pronunciation-to-word transducer to define L_1. What is the number of states and transitions of L_1?

This can be done by writing a script taking each line and returning the textual representation of the corresponding transducer, then compiling these transducers and taking their union, or by writing a script that directly returns the textual representation of the union transducer. Let l_1 denote that transducer, then L_1 can be obtained by taking the closure $L_1 = l_1^*$.

(b) [20 points] L_1 can be used to produce pronunciations for the second half of the dictionary, possibly multiple ones. What percentage of the pronunciations of the second half of the dictionary are correctly predicted using L_1? To answer this question, use as much as possible automata and transducer operations and library utilities.

One way to proceed is to construct the composed machine $M = L_1 \circ l_2^{-1}$ mapping pronunciations to pronunciations. It suffices then to keep those paths of M mapping a string to itself. The input and output labels might be shifted with respect to each other due to the presence of ϵ-transitions. To synchronize them, you can apply a synchronization algorithm `fsmsynchronize`, then remove transitions with different input and output labels (this can be done by encoding the transducer using `fsmencode` and intersecting it with a universal acceptor), and then count the number of paths of the resulting transducer by using a shortest-distance algorithm via `fsmpush` in the $(+, \times)$ semiring.

There are other direct and algorithmically more efficient ways of keeping only the identity part of the transducer M. But, the solution just described is entirely based on the utilities of the FSM or Openfst li-

braries.

(c) Answer the same questions as before by using a second half of the dictionary and a transducer L_2.

(d) Based on the results you obtained, which transducer, L_1 or L_2, is more accurate?

L_2 turns out to be more accurate.

(e) [5 points] Use the most accurate transducer to find all possible word parsings of 'T UW M EH N IY P ER S AH N Z'. Give the result as an automaton and as a list of strings in order of fewest to greatest number of words per string.

It suffices to compose a simple automaton accepting this phonemic sequence with L_2 and project on the output to obtain the solution automaton. Using `fsmbestpath -u -n N` or `farprintstrings -u -n N` with a large enough parameter `N` returns the full list of candidate strings.