11-711 Algorithms for NLP
Midterm Exam

October 18, 2012

• Before you go on, write your name at the space provided on the bottom of this page and every page of the exam.

• There are 10 pages in this exam (including this page). Verify that you have a complete copy.

• Write up your answers following each question in the exam. Adequate space has been provided.

• If you really feel that you need more space you may continue on the reverse side, but you must clearly mark the page so that we know your answer continues on the reverse side.

• The exam is open book and open notes, and is worth a total of 100 points.

• It should require 75 minutes to complete, though you will be given 85 minutes. Budget your time accordingly.

• Keep answers short and to-the-point. Concise, direct answers will receive more credit than longer, essay-like answers.

• If you find a question ambiguous, state your assumptions precisely, and proceed.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

Student Name: ________________________________
Problem 1 - Properties of Regular and CF Languages (30 points)

We define the following **language-insertion** operation on languages:

\[
\text{Ins}(L_1, L_2) = \{ uvw \mid uw \in L_2 \text{ and } v \in L_1 \}
\]

1. Let \( L_1 \) be a **regular** language and \( L_2 \) be a **context-free** language. Prove that \( \text{Ins}(L_1, L_2) \) is a **CFL** in the following way:

   Describe a general construction that given a DFSA \( A \) such that \( L(A) = L_1 \) and a PDA \( M \) such that \( L(M) = L_2 \), constructs a new PDA \( M' \) such that \( L(M') = \text{Ins}(L_1, L_2) \).

   Specify all transitions of the new PDA \( M' \). Briefly argue informally why your construction is correct, and state how you would formally prove its correctness. **You are not required to give a full formal proof!**

(20 points)

**SOLUTION:** \( L_1 \) is regular, thus there exists a DFSA \( A = (Q_1, \Sigma, \delta_1, q_{01}, F_1) \) s.t. \( L(A) = L_1 \). \( L_2 \) is CF, thus there exists a PDA \( M = (Q_2, \Sigma, \Gamma, \delta_2, q_{02}, z_{02}, F_2) \) s.t. \( L(M) = L_2 \).

We construct a new PDA \( M' \) such that \( L(M') = \text{Ins}(L_1, L_2) \). \( M' \) has states and transitions corresponding to two “copies” of \( M \) and one copy of \( A \). It also has a stack alphabet that includes \( \Gamma \cup Q_2 \). Computation starts with the first “copy” of \( M \). Epsilon transitions from any state of \( M \) allow the computation to transition to the “copy” of \( A \), while storing the state we were in at the top of the stack. The computation through the “copy” of \( A \) maintains the stack without modifications. From any state in \( F_1 \) we allow an epsilon transition to the second “copy” of \( M \) according to the state stored in the stack. Computation can then continue until the input is accepted. To prove correctness, we would need to show dual containment between \( L(M') \) and \( \text{Ins}(L_1, L_2) \).
2. Let $L_1$ be a **regular** language and $L_2$ be a **context-free** language. For each of the following statements, answer whether the statement is true or false, and give a brief explanation for your answer. If true, give a brief argument why it is true. If false, give a counter example.

(10 points)

(a) $L = \text{Ins}(\Sigma^*, L_1)$ is a CFL.

SOLUTION: True. A similar construction to the one in part ?? can be used to show that regular languages are closed under the Ins operator. Since both $\Sigma^*$ and $L_1$ are regular, $\text{Ins}(\Sigma^*, L_1)$ is a regular language. Since regular languages are a subset of CFLs, $\text{Ins}(\Sigma^*, L_1)$ is also a CFL.

(b) $L = \text{Ins}(L_1, L_2)$ is a regular language.

SOLUTION: False. Let $L_1 = \{\epsilon\}$ and $L_2 = \{0^n1^n \mid n \geq 0\}$. $\text{Ins}(L_1, L_2) = L_2$, which is a CFL but not a regular language.

(c) $L = \text{Ins}(\overline{L_1}, L_2)$ is a CFL.

SOLUTION: True. Since $L_1$ is regular and regular languages are closed under complementation, so is $\overline{L_1}$. In part-1 we showed that the Ins operator in this case generates a CFL.

(d) $L = \text{Ins}(L_1, L_1)$ is a regular language.

SOLUTION: True. Regular languages are closed under the Ins operator and $L_1$ is regular.

(e) $L = \text{Ins}(L_2, L_2)$ is a CFL.

SOLUTION: True. A similar construction to part-1 can be used to show that CF languages are closed under the Ins operator. Since $L_2$ is CF, so is $\text{Ins}(L_2, L_2)$. 

3
Problem 2 - CFGs and FSAs
(15 points)

Since context-free languages are a superset of regular languages, we know that it must be possible to construct a CFG to generate any regular language. Working the opposite direction, given a CFG that generates a regular language, it should be possible to construct an FSA that accepts the same language. In NLP tasks, adding constraints to a normally context-free problem to allow the use of finite state machinery can be incredibly useful.¹

Consider the following grammar $G$ for verb phrases that include verbs (v), adverbs (rb), adjectives (jj), prepositions (in), and nouns (n). This grammar generates sequences of parts of speech (not words). The language generated by this grammar is in fact a regular language.

```
Grammar G:

S  ->  VP

VP  -->  rb  VP
VP  -->  v  NP
VP  -->  v  PP
VP  -->  v

NP  -->  jj  NP
NP  -->  n

PP  -->  in  NP
```

1. Draw out a deterministic finite state machine that accepts $L(G)$. Make sure to clearly define your start and final states. You may use short hand notation for arcs, meaning that in any given state, if there exists no arc for a vocabulary item, there is an implied transition to a sink state and the input is rejected. Do not use any epsilon transitions.

(8 points)

¹The machine translation system called HiFST is a good example of this.
2. Write a regular expression that defines exactly the language $L(G)$. Briefly describe why the language defined by your regular expression is exactly the same as that generated by the grammar $G$.

(7 points)

**SOLUTION:**

$rb^* v + rb^* v jj^* n + rb^* v in jj^* n$. All VPs can start with any finite number of $rb$ (rule-2). The rest of the string must be generated by either rule-3, rule-4 or rule-5, resulting in three conjuncts. rule-5 generates a single $v$. rule-3 generates a $v$ followed by a single NP, which can consist of a finite number of $jj$ followed by a single $n$. rule-4 generates a PP, which consists of an $in$ followed by a single NP (as above).
Problem 3 - Weighted Finite-State Machines
(55 points)

1. Let $s = \langle s_1 s_2 \ldots s_n \rangle$ denote a string. $s_i^j$ denotes the sequence $\langle s_i s_{i+1} \ldots s_j \rangle$; interpret it as $\epsilon$ if $j < i$.

We define the following function on (string, integer) pairs:

$$tr_1(\langle s_1 s_2 \ldots s_n \rangle, k) = \langle s_1^{k-2} s_k s_{k-1} s_k^n \rangle$$  \hspace{1cm} (1)

for $k \in \{2, \ldots, n\}$. In English: this function takes the $(k-1)$th and $k$th symbols in $s$ and swaps them. The $tr_1$ function can be used to implement any reordering transformation, if we allow it to be applied multiple times. For example,

$$tr_1(carnegie, 2) = acrnegie$$
$$tr_1(acrnegie, 8) = acrnegei$$
$$tr_1(acrnegei, 7) = acrneegi$$
$$tr_1(acrneegi, 5) = acrenegi$$
$$tr_1(acrenegi, 8) =acreneig$$
$$tr_1(acreneig, 6) = acreenig$$
$$tr_1(acreenig, 7) = acreeing$$
$$tr_1(tr_1(tr_1(tr_1(carnegie, 2), 8), 7), 5), 8), 6), 7) = acreeing$$

Suppose you are given a string $s \in \Sigma^n$. We will define an algorithm that uses a weighted finite-state automaton to implement the following function:

$$f_s(t) = \begin{cases} 
1 & \text{if } s = t \\
\text{Null} & \text{otherwise} \\
k & \text{if } k \text{ is the smallest integer such that } tr_1(s, k) = t 
\end{cases}$$  \hspace{1cm} (2)

Note: It is possible that $t = tr_1(s, k)$ for more than one value of $k$. The algorithm should always return the smallest value of $k$, that is, the first position $k$ in the string $s$ such that transposing $s_{k-1}$ with $s_k$ yields $t$.

We give the algorithm for calculating $f_s$, leaving undefined the WFSA $F$. Given $t$:

(a) Find the score, denoted $y$, of the best (highest-scoring) path through $F$ recognizing the string $t$.
(b) If a path exists, return $\frac{1}{y}$.
(c) Otherwise, return NULL.
Define $\mathcal{F}$.

(20 points)

**SOLUTION:** There will be $3n - 1$ states and $3n - 3$ transitions. Start by defining $n + 1$ states: $q_0, q_1, \ldots, q_n$. $q_0$ is the start state, with weight 1. Next, add $n - 1$ states: $q_1', \ldots, q_{n-1}'$. Next, add $n - 1$ states: $q_2'', \ldots, q_n''$. Let the two final states be $q_n$ and $q_n''$ with weight 1. For each $i \in \{1, \ldots, n - 1\}$, add:

- a transition from $q_{i-1}$ to $q_i$ emitting $s_i$ with weight 1;
- a transition from $q_{i-1}$ to $q_i'$ emitting $s_{i+1}$ with weight $\frac{1}{i}$; and
- a transition from $q_i'$ to $q_{i+1}''$ emitting $s_i$ with weight 1.

For each $i \in \{3, \ldots, n\}$, add a transition from $q_{i-1}''$ to $q_i''$ emitting $s_i$ with weight 1.

---

2 You might be tempted to provide your answer by drawing a WFSA. However, we have not given you $s$; so you must instead define *how to construct* the WFSA, given an arbitrary $s$, in terms of states $Q$, initial state weights $\pi : Q \to \mathbb{R}_{\geq 0}$, transition weights $\delta : Q \times \Sigma \times Q \to \mathbb{R}_{\geq 0}$, and stopping weights $\xi : Q \to \mathbb{R}_{\geq 0}$. To make it easier for you, we will assume that, for all states $q \in Q$, $\pi(q) = 0$ unless otherwise stated, $\xi(q) = 0$ unless otherwise stated, and for all transitions $\langle q, s, q' \rangle \in Q \times \Sigma \times Q$, $\delta(q, s, q') = 0$ unless otherwise stated.
2. Describe a condition on \( s \) that guarantees \( F \) will be deterministic.

(5 points)

**Solution:** \( F \) will be deterministic if there are no two symbols \( s_i \) and \( s_{i+1} \) that are identical.

3. A different way to think about \( \text{tr}_1 \) is as a relation that holds between two strings. Concretely, we say that the relation holds between \( s \in \Sigma^n \) and \( t \in \Sigma^n \) if there exists some \( k \in \{2, \ldots, n\} \) such that

\[
\text{tr}_1(s, k) = t
\]

(3)

(Note that this is a symmetric relation; we can switch \( s \) and \( t \) in the above formula with no effect on the definition.)

Define\(^3\) a **finite-state transducer** \( R \) that implements the relation.

(10 points)

**Solution:** Define \( R \) as follows.

- \( q_0 \) is a start state. It has a self-transition that accepts \( \sigma : \sigma \), for every \( \sigma \in \Sigma \).
- \( q_1 \) is a final state. It has a self-transition that accepts \( \sigma : \sigma \), for every \( \sigma \in \Sigma \).
- For every \( \sigma \in \Sigma \) and every \( \sigma' \in \Sigma \), define a state \( q_{\sigma, \sigma'} \). Add two transitions:
  - \( q_0 \rightarrow q_{\sigma, \sigma'} \), accepting \( \sigma' : \sigma \).
  - \( q_{\sigma, \sigma'} \rightarrow q_1 \), accepting \( \sigma : \sigma' \).  

\(^3\)The guidance provided in footnote ?? still applies here. You cannot draw \( R \) because you do not know \( \Sigma \). Instead, explain how to construct it.
4. Consider again the function \( f_s \) in equation ?? (page ??). Give an algorithm that uses \( \mathcal{R} \) to calculate \( f_s(t) \) for any \( s \) and \( t \) of any length. As before, refer to footnote ?? for constructing WFSAs. To get full credit on this question, explain:

(a) How to construct a (W)FSA \( S \) representing \( s \).
(5 points)

(b) How to construct a (W)FSA \( T \) representing \( t \).
(5 points)

(c) What operations you must apply involving \( \mathcal{R} \), \( S \), and \( T \) to calculate \( f_s(t) \).
(10 points)

SOLUTION: The question was somewhat unfair; there are actually three parts required. In addition to \( \mathcal{R} \) and FSAs representing the two strings, a scoring machine, which we’ll call \( W \) is required, and we also need a way to handle the identity mapping. The final result will look like this:

\[
\frac{1}{\text{bestpathscore}(S \circ (\mathcal{R} \cup I) \circ W \circ T)}
\]

If no best path exists, return \textsc{null}. Here are the components:

(a) \( S \): There are \( n + 1 \) states: \( q_0, q_1, \ldots, q_n \). \( q_0 \) is the start state and \( q_n \) is the final state, both having weight 1. Transitions only exist between \( q_{i-1} \) and \( q_i \). For \( q_{i-1} \) to \( q_i \), there is a transition emitting symbol \( s_i \) with weight 1.

(b) \( T \): Just like \( S \), but for the string \( t \).
(c) $I$: a single state $q$, which is both the start state and the final state, both with weight one. For every symbol $\sigma \in \Sigma$, there is a self-transition on $q$ that reads and emits $\sigma$ with weight 1.

(d) $W$: This is the transducer that appropriately assigns weights. Let $N$ be some number $N \geq n$. There are $2N$ states: $q_0, q_1, \ldots, q_N$ and $q'_1, q'_2, \ldots, q'_{N-1}$. $q_0$ is the start state with weight 1 and every $q_i$ is a final state with weight 1. For each $i \in \{1, \ldots, N\}$, and for each $\sigma \in \Sigma$, there is a transition from $q_{i-1}$ to $q_i$ reading and emitting symbol $\sigma$ with weight 1. For each $i \in \{2, \ldots, N\}$, and for each $\sigma, \sigma' \in \Sigma$ such that $\sigma \neq \sigma'$:

- There is a transition from $q_{i-2}$ to $q'_{i-1}$ reading $\sigma'$ and emitting $\sigma$ with weight $1/i$.
- There is a transition from $q'_{i-1}$ to $q_i$ reading $\sigma'$ and emitting $\sigma$ with weight 1.

This machine basically finds bigram-mismatch pairs between the input and output strings and multiplies together the inverses of their positions. In case of multiple analyses, it will prefer to assume no swap has occurred (for repeated characters) or choose the earlier swap. This overgenerates, allowing multiple swaps, but it assigns scores properly to the cases we need it to handle.
5. Bonus: For this question, ignore the weights on \( R \) and think of it as an FST. Consider \( R \circ R \) (the composition of \( R \) with itself). Describe in English the relation that it implements.

\begin{align*}
\text{(up to 1 point)}
\text{SOLUTION: } \text{“} \text{tr}_2 \text{”}: \text{ holds for two strings that are exactly two local transposition edits from each other.}
\end{align*}

6. Bonus: For this question, ignore the weights on \( R \) and think of it as an FST. Let \( R^k \) denote \( k \) self-compositions of \( R \) (e.g., \( R^2 = R \circ R \), \( R^3 = R \circ R \circ R \), etc.) Consider \( \bigcup_{k=1}^{\infty} R^k = R \cup R^2 \cup R^3 \cup \cdots \). Describe in English the relation that it implements.

\begin{align*}
\text{(up to 1 point)}
\text{SOLUTION: This relation holds for strings that are permutations of each other.}
\end{align*}

7. Bonus: For this question, ignore the weights on \( R \) and think of it as an FST. Can \( \bigcup_{k=1}^{\infty} R^k \) be encoded as an FST?

\begin{align*}
\text{(up to 1 point)}
\text{SOLUTION: FSTs are not closed under infinite union; no.}
\end{align*}