11-711 Algorithms for NLP
Final Exam

Fall 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 14 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 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 three hours to complete. 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.

Good Luck!

Student Name: _____

Student Name: _____

Question	Minutes	Points	Score
1	?	?	
2	?	?	
3	?	?	
4	?	?	
5	?	?	
Total	135	100	

Problem 1 - Formal Language Theory (35 minutes)

Let us consider the following restricted class of context-free grammars LG, where every rule in the grammar has one of the following two forms:

```
A --> a B
A --> a
```

where $a \in T$ (a is a terminal symbol), and $A, B \in V$ (A and B are non-terminals).

1. Prove that the class of LG grammars can only recognize **Regular** languages, by showing a general construction that given an LG grammar G, constructs an NDFSA A, such that $L(A) = L(G)$. Argue informally why the construction is correct, and **explicitly** state how you would formally prove that it is correct. You do NOT need to write the details of the formal proof.

   SOLUTION: Let $G = (T, V, P, S)$ be an LG. We construct an NDFSA $A = (Q, \Sigma, \delta, q_0, F)$ that has the following components:

   - $Q = \{q_0, q_f\} \cup \{q_A \mid A \in V\}$
   - $\Sigma = T$
   - $q_S \in \delta(q_0, \epsilon)$
   - $q_B \in \delta(q_A, a)$ iff $A \rightarrow aB \in P_G$
   - $q_f \in \delta(q_A, a)$ iff $A \rightarrow a \in P_G$

   Informally: the NDFSA simulates a leftmost derivation of the grammar G. Since any such derivation starts with S and at each step adds a terminal symbol and has a single non-termainal in the sentential form, we "remember" the non-terminal in the state of the NDFSA. The derivation ends with an application of a rule that derives a single terminal symbol. The corresponding move of the NDFSA transitions to the single accepting state q_f. In order to formally prove the correctness of the constriction, we would need to show language equivalence by dual containment: (1) that any word w generated by the grammar G must also be accepted by the NDFSA A (by induction on the length of the grammar derivation); and (2) that any word w accepted by the NDFSA must also be generated by the grammar G (by induction on the length of the NDFSA computation path).

Student Name: _____

2. Consider the following grammar G that describes simple arithmetic expressions over the numbers $\{0, 1\}$.

```
(1)  S --> + A S
(2)  S --> * A S
(3)  S --> 0
(4)  S --> 1
(5)  A --> 0
(6)  A --> 1
```

Note, for example, that the strings $+1 * 10$ and $*1 + 1 + 01$ are in $L(G)$, but $0 + 1 * 1$ is not.

(a) Convert the grammar G into an equivelent LG grammar G', such that $L(G') = L(G)$. List the rules of the resulting new grammar G'.

SOLUTION: The following grammar:

```
(1)  S --> + B
(2)  S --> * B
(3)  B --> 0 S
(4)  B --> 1 S
(5)  S --> 0
(6)  S --> 1
```

(b) Apply your NDFSA construction algorithm from question 1.1 to the grammar G', and list the transition function of the resulting NDFSA A.

SOLUTION: We construct $A = (\{q_0, q_f, q_S, q_B\}, \{0, 1\}, \delta, q_0, \{q_f\})$ and the following transition function:

- $q_S \in \delta(q_0, \epsilon)$
- $q_B \in \delta(q_S, +)$
- $q_B \in \delta(q_S, *)$
- $q_S \in \delta(q_B, 0)$
- $q_S \in \delta(q_B, 1)$
- $q_f \in \delta(q_S, 0)$
- $q_f \in \delta(q_S, 1)$

(c) Run the NDFSA A you constructed above on the input $w = +1 * 10$ and show the sequence of states that the machine goes through until the input is accepted.

SOLUTION: $q_0 \rightarrow^{\epsilon} q_S \rightarrow^{+} q_B \rightarrow^{1} q_S \rightarrow^{*} q_B \rightarrow^{1} q_S \rightarrow^{0} q_f$

3. Consider the class of languages that can be recognized by the restricted class of context-free grammars LG. For each of the following statements, answer whether the statement is **true** or **false**, and give a brief explanation for your answer.

   (a) If L is a language that can be generated by an LG grammar, then L is a context-free language.

   SOLUTION: True. Since an LG grammar is a restricted form of a CFG, any language generated by it is by definition a CFL.

   (b) If L is a language that can be generated by an LG grammar, then L must be a finite language.

   SOLUTION: False. The language $\{0,1\}^+$ is an infinite language that is generated by the LG grammar:

```
(1)  S --> 0 S
(2)  S --> 1 S
(3)  S --> 0
(4)  S --> 1
```

   (c) If L is a regular language, then there exists some LG grammar G such that $L = L(G)$.

   SOLUTION: False. The language $L = \{\epsilon\}$ cannot ne generated by an LG grammar.

   (d) There exists a context-free language that cannot be generated by an LG grammar.

   SOLUTION: True. The language $L = \{0^n 1^n | n \geq 0\}$ cannot be generated by an LG grammar, since any language generated by an LG grammar can be simulated by an NDFSA and must therefore be regular, and the above language L is not regular according to the pumping lemma.

   (e) The set of languages that can be recognized by an LG grammar is a proper subset of the set of context-free languages.

   SOLUTION: True. Since we have shown that any language generated by an LG grammar is a CFL, but not all CFLs can be generated by an LG grammar.

Student Name: _____

Earley Parsing
(45 minutes/25 points)

Several parsers in recent years have found it useful to use the following extended version of Context-Free Grammars, which allows specifying limited forms of regular expressions on the right-hand side (RHS) of context-free production rules. Formally, an extended CFG G is defined as $G = (T, V, P, S)$, where T, V and S are respectively a set of terminal symbols, a set of non-terminal symbols, and a starting non-terminal. This is the same as a standard CFG. Productions P however consist of a left-hand side (LHS) non-terminal $A \in V$, and a RHS, that is a finite string of zero or more elements $E_1 E_2 \cdots E_k$, where each E_i is one of the following:

1. a terminal symbol $a \in T$
2. a non-terminal $B \in V$
3. $(B|C)$, denoting a disjunction of B or C, where B and C are nonterminals from V
4. $[B]$, denoting an optional $B \in V$
5. B^*, denoting zero or more occurrences of $B \in V$

For example, the rule $A \rightarrow (B|C)\ D^*$ denotes that the RHS starts with either B or C, followed by zero or more Ds.

1. We would like to modify the Earley Parsing algorithm to handle the extended version of Context-Free Grammars, that allows specifying limited forms of regular expressions on the right-hand side (RHS) of context-free production rules. Grammar rules have the form that was specified in Problem 1. For simplicity, assume that the extended grammar formalism allows **only cases 1, 2 and 3: elements on the RHS of a rule are either a terminal b, a non-terminal B, or $(B|C)$, denoting a disjunction of the two non-terminals: B or C.**

   Modifying the algorithm to directly handle extended rules requires revising the operators. Modify the original operators of the Earley algorithm to correctly handle grammar rules of the extended-CFG formalism. For your convenience, the original algorithm for each operator is provided below. Write your modified algorithm in the space below that. Briefly explain why the modified algorithm is correct.
   (15 points)

Earley's Three Operators:

- **Predictor:** If state $[A \to X_1... \bullet C...X_m, j] \in S_i$ then for every rule of the form $C \to Y_1...Y_k$, add to S_i the state $[C \to \bullet Y_1...Y_k, i]$
  Modified Version:
  SOLUTION: The Predictor operator must be modified as follows:
  - If state $[A \to X_1... \bullet C...X_m, j] \in S_i$ then for every rule of the form $C \to Y_1...Y_k$, add to S_i the state $[C \to \bullet Y_1...Y_k, i]$
  - If state $[A \to X_1... \bullet [B|C]...X_m, j] \in S_i$ then add to S_i the two states states $[A \to X_1... \bullet B...X_m, j]$ and $[A \to X_1... \bullet C...X_m, j]$

- **Completer:** If state $[A \to X_1...X_m \bullet, j] \in S_i$ then for every state in S_j of form $[B \to X_1... \bullet A...X_k, l]$, add to S_i the state $[B \to X_1...A \bullet ...X_k, l]$
  Modified Version:
  SOLUTION: The Completer operator does NOT need to be modified.

- **Scanner:** If state $[A \to X_1... \bullet a...X_m, j] \in S_i$ and the next input word is $x_{i+1} = a$, then add to S_{i+1} the state $[A \to X_1...a \bullet ...X_m, j]$
  Modified Version:
  SOLUTION: The Scanner operator does NOT need to be modified.

2. Does the revised Earley algorithm still have the property that at most **one** operator can apply to any item? Briefly explain your answer, or demonstrate with an example. **(5 points)**

3. Do the modifications you proposed have any consequences on the $O(n^3)$ worst-case time complexity of the algorithm? Briefly explain why, or why not. **(5 points)**

1 Dependency Parsing with Labels (?? minutes/30 points)

In class, you learned how the problem of finding the maximum-scoring dependency parse for a sentence can be accomplished using dynamic programming (if only projective trees are considered) or the Chu-Liu-Edmonds (CLE) algorithm (if nonprojective trees are also considered). The algorithms we discussed assume that the score of a parse tree is defined by the sum of the scores of directed attachments in the tree. Let the words in the input sentence be denoted $w_1 w_2 \dots w_n$, and let the score of the attachment of word w_j as a child of word w_i be $s_{j,i}$. We let $w_0 = \$$ denote the root symbol, so that $s_{j,0}$ is the score of attaching word w_j to the root. Let $\boldsymbol{s}$ denote the entire set of attachment scores. If, in a given tree, we let π_j denote the index of w_j's parent, then the score of the tree is:

$$\sum_{j=1}^{n} s_{j,\pi_j} \tag{1}$$

In class, we only discussed in detail the case of *unlabeled* dependency parsing, though dependencies are usually described linguistically as having *types* or labels. Of course, the set of dependency labels used by a given parser are a design decision. One kind of label captures syntactic relations like the one held between a verb and its subject or a preposition and its complement. Another kind of label can be used to encode phrase-structure syntax (i.e., what is modeled by context-free grammars).

1. Describe how to transform any phrase-structure tree into a labeled dependency tree *without* losing any information. Assume that the phrase-structure tree already has head-marking (i.e., every nonterminal node is annotated with the index of its head child). Do not explain how to recover the parent for each word; we assume you already know how to do this. It may help you to draw an example to work out the problem, but your solution should not make use of an example; it should explain how to construct the labels for the dependency tree, given any head-marked phrase-structure tree. SOLUTION: For the attachment of child w_i to parent w_j, let the label be the sequence of nonterminals for which w_i is the head word. For word i, we denote this sequence $\boldsymbol{N}_i = \langle N_1^i, N_2^i, \dots, N_{\ell_i}^i \rangle$, with N_1^i being the preterminal for w_i and proceeding "up" the tree. This should get partial credit, but there is one more thing that needs to be specified; we need to give the index of the nonterminal in $\boldsymbol{N}^j$ to which $N_{\ell_i}^i$ should attach. Call this number I^i. The label will be $L_i = \langle \boldsymbol{N}^i, I^i \rangle$.
(5 points)

2. Describe how to recover the phrase-structure tree from a labeled dependency tree constructed according to your answer to part 1. In your solution, let π_i denote the index of the parent of w_i, ϕ_i denote the number of children of w_i, $\boldsymbol{\xi}_i = \langle \xi_{i,1}, \dots, \xi_{i,\phi_i} \rangle$ denote the sequence of w_i's children, and L_i be the label of the attachment of w_i as a child of w_{π_i}. You may not need to use all of these symbols. SOLUTION: Given the sequence label and parent nonterminal attachment index for each word, the phrase structure tree can be reconstructed by traversing the dependency tree top-down. When

you encounter a word w_i, construct the tree fragment of nonterminals $\boldsymbol{N}^i$ with w_i as head, and attach $N^i_{\ell_i}$ to $N^{\pi_i}_{I^i}$.
(5 points)

3. One solution to labeled dependency parsing is to solve it in two stages. First, find the best unlabeled dependency parse using CLE or dynamic programming with the scores $\boldsymbol{s}$. Then use a separate model designed to label the edges of that tree. This is called a "pipeline." You will define an algorithm that finds the best labeling of a tree. Letting L_i again denote a label for the attachment of w_i to its parent w_{π_i}, let the score of labeling of a (fixed) tree be defined by:

$$\boldsymbol{\sigma}(L_0, L_1, \ldots, L_n) = \sum_{i=1}^{n} \sigma(i, L_i, L_{\pi_i}) \tag{2}$$

That is, we score each word's attachment label, taking into consideration the label that its parent's attachment receives. (For completeness, let the label of the root's "attachment" be $L_0 = \emptyset$, a special null symbol.)

Provide a bottom-up dynamic programming algorithm for finding the best labeling of the tree assuming $\boldsymbol{\sigma}$ is given. Let $\mathrm{t}(L, i)$ be the score of the best possible labeling for the subtree rooted at w_i assuming that w_i's attachment to its parent is labeled L. Below are the base case and the definition of "goal" (the score of the best complete labeling for the tree). You need to state the recurrence.

$$\begin{aligned} \mathrm{t}(L, i) &= 0 \text{ whenever } w_i \text{ is a leaf} \\ \text{goal} &= \max_L \sum_{i:\pi_i=0} \mathrm{t}(L, i) + \sigma(i, L, \emptyset) \end{aligned}$$

(9 points)

SOLUTION:

$$\mathrm{t}(L, i) = \sum_{j:\pi_i=i} \max_{L'} \mathrm{t}(L', j) + \sigma(j, L', L)$$

The algorithm works bottom up, scoring partial labelings of each node for the best labelings of its children's subtrees. Like Viterbi, a backtrace is required. Runtime $O(L^d)$ if there are L labels and d is the maximum number of children of any node in the tree. Something better is possible if we binarize.
Noah's notes after grading in 2011. Note that s was a local score for just the label by itself, now removed; s' is what is now σ. I didn't take off for failing to mention the backtrace. I took off one point (each) for missing the s term, missing the s' term, extra max operators, using a product instead of a sum, unnecessary separate treatment of parents-of-leaves from other words; two points for not scoping the max broadly enough or unbound variables; three points for each missing scope; four points for missing max altogether or missing sum altogether; nine points for missing the recurrence.

There's an alternative form where you do a topological sort so that you always process children before parents, essentially giving a binarization. Most people who tried to do something like that got it mostly wrong.

4. True or false: if given a weighted context-free grammar $\mathcal{G}$, one could define a set of scores $\boldsymbol{\sigma}$ so that the score of any labeled dependency tree under $\boldsymbol{\sigma}$ (useing equation 2) equals the score (under $\mathcal{G}$) of the phrase-structure tree it encodes. SOLUTION: true
(1 point)

5. We next consider a single-pass algorithm that finds a labeled tree all at once. Here is a sketch of how the algorithm works:

   (a) Construct a directed graph. (This is the part you will describe in more detail below.)

   (b) Run the CLE algorithm find the best scoring directed spanning tree of the graph.

   (c) Apply zero or more operations to the tree given by CLE. (This is the part you will describe in part 6.)

   Assume you are given the scores of all possible *labeled* attachments, denoted $s_{j,i,L}$ (for attaching child w_j to parent w_i with label L), and an implementation of the CLE algorithm. Note that there may be multiple labels possible for any given attachment! Describe how to construct the graph in step 5a.

   SOLUTION: Every word is a vertex. Every candidate attachment to a parent with a label is a separate directed edge to that vertex. Compare all edges from the same parent to the same child; keep only the maximum scoring one. Or, if CLE is allowed to take multiple edges between the same pair of vertices, simply pass the whole graph to CLE.
   Noah's notes after grading in 2011. (This was worth 10 points that year.) To get full credit, you have to mention or discuss the issue of multiple possible labels for every possible parent-child pair. I took off 9 for missing this, and also for answers involving running CLE *first* (with what scores?) and then picking labels. Some people talked about enumerating all the labeled graphs, which if essentially correct got half credit. I took off 7 points when the vertices were constructed to include the label (this is incorrect, because the best tree will now be over vertices which are labels plus words, not words, so it's not a dependency tree).
   (3 points)

6. Describe how the output of CLE can be transformed in step 5c above to produce the desired output. SOLUTION: Nothing needs to be done; CLE will give a labeled tree.
(3 points)

7. Describe the advantages and disadvantages of the two methods you defined above (pipeline and single-pass). SOLUTION: The pipeline allows modeling interactions among labels; the single-pass algorithm allows labels and attachments to interact. The single pass algorithm is probably faster (need to check asymptotics). There may

be more reasonable answers.
(4 points)

2 CFG with Unification Constraints (15 minutes/12 points) A-Side

Consider the following context-free grammar that recognizes simple sentences such as "*the students teach the students*":

Grammar:

```
S --> NP VP
NP --> DET N
NP --> N
VP --> V
VP --> V NP
```

Lexicon:

```
DET --> the
N --> student
N --> students
V --> look
V --> looks
V --> teach
V --> teaches
```

1. While this grammar parses all grammatical sentences for the given lexicon, it also parses many ungrammatical sentences. For example, sentences with subject/verb disagreement, such as "*the student teach*", are parsed. In addition to sentences with subject/verb disagreement, identify two types of ungrammatical sentences that can be parsed with the given grammar and provide an example for each.
   (4 points)

   SOLUTION: The following types of ungrammatical sentences parse:

   - Sentences with incorrect definiteness/determiner usage,
     ex: "*student teaches*"
   - Sentences with incorrect verb transitiveness,
     ex: "*students look the students*"

2. We would like to augment the basic grammar with unification constraints such that all grammatical sentences unify, producing valid parses, while ungrammatical sentences fail unification. In a paragraph, describe how the grammar could be augmented with unification constraints to address subject/verb disagreement and the additional types of ungrammatical sentences you identified above. Clearly indicate what features should be added to the lexical entries and what constraints should be added to the grammar rules.
(4 points)

SOLUTION: In the lexicon, nouns can be augmented with number (num = sg/pl), and verbs can be augmented with usage (use = sg/pl) and potential transitiveness (trans = +/-). Rules building a NP from a single N should disallow singular nouns (N.num = pl). Rules combining a V and NP should require V to be transitive (V.trans = +). Rules combining a NP and VP should require NP.N's number to agree with VP.V's usage.

3. We would like to demonstrate that adding simple, non-recursive feature-value unification constraints does not change the class of languages accepted by CFGs. For a given CFG $G = \langle N, \Sigma, R, S \rangle$ with unification constraints that use only simple atomic values (such as <`N agr> = *3pl`), describe $G' = \langle N', \Sigma', R', S' \rangle$, a CFG without constraints that recognizes the same language as G. Answer in 3–4 sentences.
(4 points)

SOLUTION: $\Sigma' = \Sigma$. Nonterminals in N' take the form $\langle n, f \rangle$, where $n \in N$ and f is a static representation of a feature structure. Rules in R' enumerate all valid combinations of feature structures for each nonterminal, for each way to fill the LHS and RHS of rules in R. Finally, rules $S' \rightarrow S^+$ exist where S^+ is a rule in R' with RHS of S with any feature structure.

3 Semantic Role Labeling (10 minutes/9 points)

Thematic Roles	
AGENT	***The waiter*** spilled the soup.
EXPERIENCER	***John*** has a headache.
FORCE	***The wind*** blows leaves into the yard.
THEME	He broke ***the ice***.
RESULT	The government has built ***a new railroad system***.
INSTRUMENT	He opened the door ***with a key***.
BENEFICIARY	He made the hotel reservation ***for his boss***.
SOURCE	She flew in ***from Boston***.
GOAL	She drove ***to Portland***.

Assign the various verb arguments in the following examples to their appropriate thematic roles, using the set of roles shown above. You should underline the words involved in each role and write the name of the role below the words. We have marked the verbs with SMALL CAPS. Arguments in roles not included above should not be marked (for example, TIME).

1. The hurricane STRUCK New York on Tuesday.

   (3 points)

2. Subways and tunnels all over town WERE FILLED with millions of gallons of seawater.

   (3 points)

3. Many residents ARE RETURNING to their homes anyways.
   (3 points)

SOLUTION: Given the thematic roles shown, one reasonable assignment of thematic roles is:

1. [FORCE The hurricane] struck [THEME New York] on Tuesday.
2. [THEME Subways and tunnels all over town] were filled [RESULT with millions of gallons of seawater].
3. [AGENT Many residents] are returning [GOAL to their homes] anyways.