UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR08008 INFORMATICS 2A: PROCESSING FORMAL AND NATURAL LANGUAGES

Saturday 10 th December 2016

09:30 to 11:30

INSTRUCTIONS TO CANDIDATES

1. **Answer all five questions in Part A, and two out of three questions in Part B. Each question in Part A is worth 10% of the total exam mark; each question in Part B is worth 25%.**
2. **Use a single script book for all questions.**
3. **Calculators may be used in this exam.**

Convener: I. Simpson
External Examiner: I. Gent

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

Part A

1. (a) Draw the state diagram for a simple NFA N over the alphabet $\{a, \ldots, z\}$ that accepts precisely the strings ending in *lalla*. The states should be labelled as $0, 1, 2, \ldots$ in the obvious way. You may use obvious abbreviations wherever a large number of very similar transitions appears. [*2 marks*]

   (b) Use the subset construction to convert N to an equivalent DFA M. You need only include the reachable states in your diagram. Label each state of M to show which *set* of states from N it corresponds to. Again, abbreviations are acceptable. [*7 marks*]

   (c) How may we use M to identify *all* occurrences of the substring *lalla* within a longer string over $\{a, \ldots, z\}$? [*1 mark*]

2. (a) State the Pumping Lemma for regular languages. [*3 marks*]

   (b) Consider the language $L = \{a^m b^n \mid m \neq n\}$ over the alphabet $\{a, b\}$. What difficulty do we encounter if we try to use the Pumping Lemma directly to show that L is not regular? It will suffice here to illustrate the failure of one plausible attempt at doing this, pinpointing where a problem arises. [*4 marks*]

   (c) Give an indirect proof that the above language L is not regular, by using standard closure properties of regular languages and appealing to a standard example of a non-regular language. [*3 marks*]

3. Consider the following emission and transition probabilities for parts of speech. (For simplicity, we use MOD, a generic part of speech standing in for adjectives and adverbs). Assume every sequence must be preceded by START and followed by STOP.

	All	were	the	UNK
DT	–	–	1.0	–
MOD	0.2	–	–	0.8
NN	–	–	–	0.5
VB	–	0.8	–	0.2

from ↓ to →	DT	MOD	NN	VB	STOP
START	0.5	0.3	0.2	–	–
DT	–	0.3	0.7	–	–
MOD	–	0.5	0.3	0.2	–
NN	–	–	–	0.5	0.5
VB	.03	0.3	0.3	–	0.1

The UNK token in the emission table should be substituted for any input word that does not otherwise appear in the table. Now suppose that we receive this line from Lewis Carroll's *Jabberwocky* as input.

All mimsy were the borogoves

Using the tables above, use the Viterbi algorithm to determine the most probable tag sequence an HMM tagger would assign to this sentence. Show your work. [*10 marks*]

4. Consider the following grammar.

   S → NP VP (1.0)
   VP → VB NP PP (0.4) | VB NP (0.6)
   PP → Prep NP (1.0)
   NP → NP PP (0.3) | NN (0.7)
   NN → scientists (0.3) | space (0.5) | whales (0.2)
   VB → count (1.0)
   Prep → from (1.0)

   Now suppose we have the following input sentence, a February 2014 headline from the BBC News:

   Scientists count whales from space

   Draw all possible parse trees for this sentence and compute their probabilities. Which parse would the parser choose? [*10 marks*]

5. A **polarity item** is a word or phrase that can only appear in either a positive or negative context. For example, *somewhat* can appear only in positive contexts, while *at all* can appear only in negative contexts:

   you like the film somewhat
   you do not like the film at all
   * you do not like the film somewhat
   * you like the film at all

   (Recall that an example sentence preceded by * is one that is considered ungrammatical). Now suppose that we have the following grammar in which nonterminals are capitalized, terminals are lowercase, and the start symbol is S:

   S → NP VP
   VP → VB ADVP | NEG VB ADVP
   NP → you | the film
   NEG → do not
   VB → like
   ADVP → somewhat | at all

   This grammar accepts both the grammatical and ungrammatical sentences above. You should redesign it so that it accepts the two grammatical sentences, and rejects the ungrammatical ones.

   (a) Design a new grammar without using agreement features. [*4 marks*]
   (b) Design a new grammar using agreement features. [*4 marks*]
   (c) Explain how the grammars with and without agreement features differ, and whether the difference is functionally important. (**Hint**: do the grammars generate the same string languages? What about their analyses?) [*2 marks*]

Part B

6. In typical Unix-style command shells, a Java program may be executed using the command `java` in conjunction with the name of the program to be run. The latter may be either the name of a class file, or (if preceded by `-jar`) the name of a Java archive file containing the program. In addition, the command may specify a number of run-time *options* (these appear before the name of the program), as well as a list of *arguments* to be passed to the program itself (these appear after the name of the program).

   A simplified version of the syntax of such commands is given by the following LL(1) grammar. The start symbol is command, and the six terminals are

   `java` `-` `-jar` `:` `=` *str*

   For the purpose of this question, we shall treat `-jar` as a single lexical item, although in other contexts, `-` will be treated as a token by itself. The terminal *str* stands for a single lexical class of *strings* which we shall use for many different purposes: as file names, option names, option values and argument values. The productions of the grammar are as follows:

   $$\begin{aligned}
   \text{command} &\rightarrow \texttt{java}\ \text{opts}\ \text{file}\ \text{args} \\
   \text{opts} &\rightarrow \epsilon \mid \text{opt}\ \text{opts} \\
   \text{opt} &\rightarrow \texttt{-}\ \textit{str}\ \text{qualifier} \\
   \text{qualifier} &\rightarrow \epsilon \mid \texttt{:}\ \textit{str} \mid \texttt{=}\ \textit{str} \\
   \text{file} &\rightarrow \textit{str} \mid \texttt{-jar}\ \textit{str} \\
   \text{args} &\rightarrow \epsilon \mid \textit{str}\ \text{args}
   \end{aligned}$$

   (a) Write down the set E of *potentially empty* non-terminals for this grammar. Using this, calculate the *First* sets for all of the non-terminals in the grammar. [*4 marks*]

   (b) Calculate the *Follow* sets for all the non-terminals in the grammar. (This is more demanding than part (a) and hence carries more credit.) [*6 marks*]

   (c) Using these *First* and *Follow* sets, or else by simple inspection, construct the LL(1) parse table for the grammar. You may use either a double page or a single page in landscape orientation for this. [*9 marks*]

   (d) Suppose now that we attempt to parse the sequence of tokens

   `java` *str* `-jar` *str*

   via the standard LL(1) parsing algorithm, using your parse table from part (c).

QUESTION CONTINUES ON NEXT PAGE

Construct a table showing how the computation proceeds, displaying at each step the operation performed, the input remaining, and the state of the stack. Continue with the computation until *either* a successful parse is achieved, *or* some error is encountered. If the latter occurs, indicate precisely where and how the error is detected by the parser. [*6 marks*]

7. This question explores the notion of *interleaving* of regular languages, a concept not formally defined within the course.

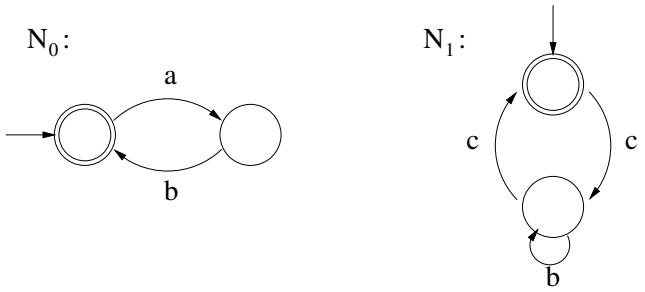
   By an interleaving of two strings s and t, we mean any string u whose symbol occurrences may each be tagged with either 0 or 1 in such a way that the symbols tagged with 0 (in order) spell out precisely the string s, and those tagged with 1 spell out t. For example, each of the following is a possible interleaving of the strings *moon* and *note*:

   moonnote *notemoon* *monotone* *nomtoeon*

   By the interleaving of two *languages* L_0 and L_1, we mean the language consisting of all possible interleavings of a string from L_0 and a string from L_1:

   $$L_0 \parallel L_1 \;=\; \{u \mid \exists s \in L_0,\, t \in L_1.\; u \text{ is an interleaving of } s \text{ and } t\,\}.$$

   (a) Consider the following NFAs N_0, N_1 over the alphabet $\{a, b, c\}$:

   Draw the state diagram for an NFA N such that $\mathcal{L}(N) = \mathcal{L}(N_0) \parallel \mathcal{L}(N_1)$, where $\parallel$ is the interleaving operation defined above. (Hint: Your NFA should have a state for every *pair* of states (q_0, q_1) where q_i is a state of N_i for $i = 0, 1$. However, it should clearly not be the same as the NFA corresponding to the *intersection* $\mathcal{L}(N_0) \cap \mathcal{L}(N_1)$.) [*5 marks*]

   (b) Now given two *arbitrary* NFAs $N_0 = (Q_0, \Delta_0, S_0, F_0)$, $N_1 = (Q_1, \Delta_1, S_1, F_1)$ over the same alphabet Σ, give a general mathematical definition of an NFA $N = (Q, \Delta, S, F)$ such that $\mathcal{L}(N) = \mathcal{L}(N_1) \parallel \mathcal{L}(N_2)$. Your definition should specify the set of states Q, the transition relation $\Delta \subseteq Q \times \Sigma \times Q$, the set of start states $S \subseteq Q$, and the set of accepting states $F \subseteq Q$. [*5 marks*]

   (c) Given an alphabet Σ, let $L(\Sigma)$ denote the set of strings s such that every symbol $a \in \Sigma$ occurs exactly once within s. Show how $L(\Sigma)$ may be obtained from a family of extremely simple languages via interleaving. Hence show how one may construct an NFA N_Σ for $L(\Sigma)$. How many states does this NFA have? [*5 marks*]

   (d) State what it means for an NFA to be *minimal* (noting that the usual definition for DFAs makes sense also for NFAs). Is your NFA N_Σ from part (c) minimal? Briefly justify your answer. [*5 marks*]

QUESTION CONTINUES ON NEXT PAGE

(e) Given a string s, let M_s denote the obvious minimal DFA that accepts the string s and nothing else. By applying the construction from part (b) to two copies of M_s, we obtain a machine M_s^2 that accepts the language L_s^2 consisting of all interleavings of s with itself. Show that for any $k \geq 0$, there is a string s such that M_s^2 has more than k times the number of states in the minimal DFA for L_s^2. [*5 marks*]

8. In English, the *respectively* construction relates two ordered sets of n words each, such that the ith word of the first set is related to the ith word of the second, for all i from 1 to n. For example, suppose we see the sentence:

   a wall, table, and floor are yellow, green, and blue, respectively,

   This means that a wall is yellow, a table is green, and a floor is blue.

   Before we delve into the construction, let's first remind ourselves how semantics are assigned to easier sentences. Suppose that we have the following highly simplified grammar:

   $$\begin{aligned} \text{S} &\rightarrow \text{a NP is JJ} \\ \text{NN} &\rightarrow \text{wall} \mid \text{table} \mid \text{floor} \\ \text{JJ} &\rightarrow \text{blue} \mid \text{green} \mid \text{yellow} \end{aligned}$$

   (a) The above grammar generates simple sentences without the *respectively* construction, such as *a table is green*. We would like this sentence to have the semantics $\exists x.\textsc{Table}(x) \wedge \textsc{Green}(x)$. Add lambda terms to the grammar so that it computes the correct semantics for each sentence that it generates. (Note: for this exercise, it is only important that your lambda terms compute to the correct semantics. It doesn't matter whether they are the same as those a semanticist would write.) [*5 marks*]

   (b) Now we will develop a simple version of the *respectively* construction. Modify the grammar (including the lambda terms) so that it will accept sentences of the form *a NN and NN are JJ and JJ , respectively* and assign the correct semantics to them. For example, if your parser encounters the sentence *a table and wall are blue and yellow , respectively*, it should assign the correct semantics: $\exists x.\exists y.\textsc{Table}(x) \wedge \textsc{Blue}(x) \wedge \textsc{Wall}(y) \wedge \textsc{Yellow}(y)$. [*5 marks*]

   (c) Let's move to a more general form of the *respectively* construction. For this part, modify your syntactic grammar (don't worry about semantics yet) so that it will accept sentences of the form *a NN , ... , NN and NN are JJ , ... , JJ and JJ , respectively*, for equal numbers of NN and JJ categories. If it's helpful, you can assume that the comma token (,) has category CC (coordinating conjunction). It is ok if your grammar accepts other sentences, for instance replacing comma (,) with *and*. [*4 marks*]

   (d) As we've seen in lecture, all regular languages are expressible as context-free grammars, and indeed our original grammar and the modified grammar grammar of (8b) produce regular languages. What about the langage of (8c)? Is it regular? It suffices to give an intuitive justification; a formal proof is not needed. [*3 marks*]

   (e) Add lambda terms to your grammar, or if you are unable to, explain informally what problem arises when you try to do so. [*5 marks*]

   (f) Does the *respectively* construction remind you of any linguistic phenomena we discussed in class? How is it similar? How is it different? [*3 marks*]