UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR08008 INFORMATICS 2A: PROCESSING FORMAL AND NATURAL LANGUAGES

Saturday 13 th December 2014

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: D. K. Arvind
External Examiner: C. Johnson

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

PART A

1. This question concerns the language below.

$$L \;=\; \{xx \mid x \in \Sigma^*\}$$

In other words, L is the set of all strings produced by concatenating some string x with itself.

(a) In the case that $\Sigma = \{a\}$, show that L is regular. [*2 marks*]

(b) In the case that $\Sigma = \{a, b\}$, use the pumping lemma to prove that L is not regular. [*7 marks*]

(c) In the case that $\Sigma = \{a, b\}$, where does the language L reside in the Chomsky hierarchy? [*1 mark*]

2. Consider the following simplified lexical classes for integer and floating-point literals, specified using the `egrep` pattern language.

```
INT-LIT = [0-9]+
FLT-LIT = [0-9]*(\.)[0-9]+
```

In Haskell, the expression:

```
[7..10]
```

(which defines the integer list `[7,8,9,10]`) has the following lexing.

```
[     7      ..     10     ]
[  INT-LIT   OP   INT-LIT   ]
```

Here, the lexemes appear on the top row, with their lexical classes positioned beneath them. These lexical classes include singleton classes `[` and `]`, for opening and closing list brackets, and a lexical class `OP` of binary operators.

In answering the questions below, refer only to lexical classes mentioned above.

(a) What is the lexing of the Haskell expression below?

```
[7.10]
```

[*1 mark*]

Explain what happens at the following stages of the lexing process, when lexing the seven-character expression ‘ `[7..10]` ’ .

(b) After the second character (‘ `7` ’) has been read. [*3 marks*]

(c) After the third character (the first ' . ') has been read (for the first time). [*3 marks*]

(d) After the fourth character (the second ' . ') has been read (for the first time). [*3 marks*]

3. Use the *Viterbi algorithm* to compute the most probable POS tagging for the word sequence

fat orange ducks

using transition and emission probabilities as follows:

	to N	to V	to A
from start	0.5	0.2	0.3
from N	0.3	0.6	0.1
from V	0.8	0.2	0.2
from A	0.6	0.1	0.3

Transitions

	fat	orange	ducks
N	0.2	0.3	0.5
V	0.0	0.0	0.2
A	0.4	0.5	0.0

Emissions

Show your working and include backtrace pointers in your Viterbi matrix. [*10 marks*]

4. (a) The three main parsing algorithms considered in the course are the *LL(1)*, *CYK* and *Earley* algorithms. State which of these are top-down algorithms, and which are bottom-up. [*2 marks*]

(b) State one potential advantage of Earley parsing over CYK parsing in the context of natural language processing. [*1 mark*]

(c) Consider the context-free grammar

$$\begin{array}{rcl} \mathsf{S} & \rightarrow & \mathsf{NP\ VP} \\ \mathsf{NP} & \rightarrow & \mathsf{N} \mid \text{the } \mathsf{N} \\ \mathsf{VP} & \rightarrow & \mathsf{V} \mid \mathsf{V\ N} \end{array}$$

where S is the start symbol, and N, V are terminals referring respectively to the class of nouns (e.g. *things*) and that of verbs (e.g. *happen*).

Use the Earley algorithm to parse the sentence *things happen*. You should show the execution of the algorithm as a table with a row for each step. Each row should include the start and end position of the portion of input processed, and a letter to indicate whether the step is due to the predictor, scanner or completer. For example, the first row of your table should be

$$\mathsf{S} \rightarrow \bullet\ \mathsf{NP\ VP} \qquad [0,0] \qquad \mathrm{P}$$

[*7 marks*]

5. Consider the following context-free grammar, which generates English sentences with *reflexive pronouns* as objects:

$$
\begin{array}{rcl}
\mathsf{S} & \rightarrow & \mathsf{NP\ VP} \\
\mathsf{NP} & \rightarrow & \text{Anna} \mid \text{Bill} \\
\mathsf{NP} & \rightarrow & \mathsf{Det\ N} \\
\mathsf{VP} & \rightarrow & \mathsf{V\ Refl} \\
\mathsf{Det} & \rightarrow & \text{every} \mid \text{some} \\
\mathsf{N} & \rightarrow & \text{girl} \mid \text{boy} \mid \text{robot} \\
\mathsf{V} & \rightarrow & \text{hides} \mid \text{washes} \\
\mathsf{Refl} & \rightarrow & \text{herself} \mid \text{himself} \mid \text{itself}
\end{array}
$$

(a) In English, reflexive pronouns are required to agree with their subjects in gender/animacy. As it stands, the above grammar generates sentences that do not respect this rule, such as 'Every boy washes itself'. Construct a *parameterized* version of the above context-free grammar that enforces this rule with the help of a suitable *attribute*. You should not parameterize any more of the non-terminals than is necessary. [*6 marks*]

(b) Returning to the original version of the grammar, we now consider a compositional semantics for the language, in which phrases of category V, VP and NP are interpreted by terms of type $<e,<e,t>>$, $<e,t>$, and $<<e,t>,t>$ respectively. The following are two of the clauses from the definition of such a semantics:

$$
\begin{array}{rcll}
\mathsf{S} & \rightarrow & \mathsf{NP\ VP} & \{\mathsf{NP.Sem(VP.Sem)}\} \\
\mathsf{NP} & \rightarrow & \text{Bob} & \{\lambda P.P(Bob)\} \\
\mathsf{V} & \rightarrow & \text{washes} & \{\lambda y.\lambda x.Washes(x,y)\}
\end{array}
$$

Write down a semantic attachment for the production $\mathsf{VP} \rightarrow \mathsf{V\ Refl}$ that fits with these clauses and captures the usual meaning of this construction. Also write down the logical formula (in β-normal form) that you would expect to obtain as the interpretation of the sentence

some robot washes itself

(note that you have not been given all the clauses that would be needed to derive this formally). [*4 marks*]

PART B

6. The pushdown automata (PDAs) presented in lectures used *empty stack* as the acceptance condition. In this question we consider a version of PDAs that instead accept if an *accepting control state* is reached at the end of the input. There is no requirement for the stack to empty.

   Such a variant PDA, for a given alphabet Σ, is specified by the usual data required for a PDA (a finite set Q of control states, a start control state $s \in Q$, a finite stack alphabet Γ, a start stack symbol $\bot \in \Gamma$, and finite transition relation Δ) together with one extra component:

   - A finite set of accepting control states $F \subseteq Q$.

   As usual, we write transitions in Δ in the form

   $$q \xrightarrow{c,\, v \,:\, \alpha} q'$$

   meaning that when the automaton is in control state $q \in Q$ and $v \in \Gamma$ is popped from the top of the stack, the input symbol or empty string $c \in \Sigma \cup \{\epsilon\}$ can be read to reach control state $q' \in Q$ with $\alpha \in \Gamma^*$ pushed back onto the stack.

   A run of such a variant PDA on a string $x \in \Sigma^*$ is *accepting* if the PDA ends up in some control sate $q \in F$, having read the entire input string x.

   Henceforth in this question, 'PDA' will always mean PDA with accepting states. Nondeterministic PDAs of this kind recognise exactly the context-free languages.

   (a) Consider a PDA with two control states $Q = \{p1, p2\}$, start state $p1$, input alphabet $\Sigma = \{a, b\}$, stack alphabet $\Gamma = \{\bot, a\}$, start stack symbol $\bot$, a single accepting control state $F = \{p2\}$, and transition relation:

   $$p1 \xrightarrow{a,\, \bot \,:\, a} p1 \qquad p1 \xrightarrow{a,\, a \,:\, a\,a} p1$$
   $$p1 \xrightarrow{\epsilon,\, a \,:\, a} p2 \qquad p2 \xrightarrow{b,\, a \,:\, \epsilon} p2$$

   Describe in detail an execution of this PDA that accepts the string

   $$a\,a\,a\,b$$

   [6 marks]

   (b) Give a concise mathematical description of the language recognised by the PDA of part (a). *[2 marks]*

   Let $M_1 = (Q_1, s_1, \Gamma, \bot, F_1, \Delta_1)$ be a PDA over Σ as defined above. Let $M_2 = (Q_2, s_2, F_2, \Delta_2)$ be a nondeterministic finite automaton (NFA) over the same alphabet Σ (for simplicity we assume that M_2 has a single start state s_2). We construct a new PDA $M = (Q, s, \Gamma, \bot, F, \Delta)$ as follows.

QUESTION CONTINUES ON NEXT PAGE

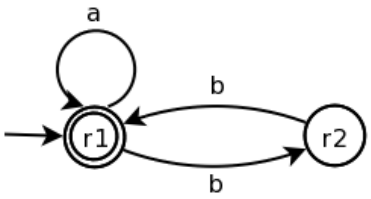
$$Q \;=\; Q_1 \times Q_2 \qquad s \;=\; (s_1, s_2) \qquad F \;=\; F_1 \times F_2 \;\;,$$

and the set Δ of transitions contains:

- $(q_1, q_2) \xrightarrow{c,v \,:\, \alpha} (q_1', q_2')$ obtained by pairing transitions $q_1 \xrightarrow{c,v \,:\, \alpha} q_1'$ in Δ_1 with transitions $q_2 \xrightarrow{c} q_2'$ in Δ_2 that use the same character c,
- and also $(q_1, q_2) \xrightarrow{\epsilon,v \,:\, \alpha} (q_1', q_2)$, for every transition $q_1 \xrightarrow{\epsilon,v \,:\, \alpha} q_1'$ in Δ_1.

We call the PDA M constructed from M_1 and M_2 in this way the *product PDA*.

(c) Let M_1 be the PDA from part (a), and let M_2 be the NFA below.



Write out the start state s, set of accepting states F, and transition relation Δ of the product PDA constructed from M_1 and M_2. [*8 marks*]

(d) Give a consise mathematical description of the language recognised by the product PDA obtained in part (c). [*2 marks*]

(e) Outline an argument for why the statement below is true.

> *If L_1 is a context-free language and L_2 is a regular language then the intersection $L_1 \cap L_2$ is also a context-free language.*

[*4 marks*]

(f) Using the pair of languages below as a counterexample, show that context-free languages are not closed under intersection.

$$L_1 \;=\; \{a^m b^n c^n \mid m, n \geq 0\} \qquad L_2 \;=\; \{a^m b^m c^n \mid m, n \geq 0\}$$

[*3 marks*]

7. The following is a grammar for a very simple class of logical expressions, containing operations written in textual form. The terminals are

$$\mathsf{and} \quad \mathsf{not} \quad (\quad) \quad \mathsf{var}$$

where var represents a lexical class of boolean variables (e.g. $p, q, r, \ldots$). The grammar implements bracketed expressions, where and is a binary infix operation, and not is a unary prefix operation, which binds more tightly than and.

$$\begin{aligned} \mathsf{Exp} &\rightarrow \mathsf{Exp1\ Ops} \\ \mathsf{Ops} &\rightarrow \epsilon \mid \mathsf{and\ Exp1\ Ops} \\ \mathsf{Exp1} &\rightarrow \mathsf{var} \mid \mathsf{(Exp)} \mid \mathsf{not\ Exp1} \end{aligned}$$

(a) This grammar is LL(1). Write out its parse table. (You do not need to explain how you obtain the parse table. In particular, you are not required to say what the *first* and *follow* sets of the nonterminals are.) [*6 marks*]

(b) Describe the step-by-step execution of the LL(1) predictive parsing algorithm, in parsing the expression below using your parse table, where p, q are boolean variables.

$$\mathsf{not}\ p\ \mathsf{and}\ q$$

[*7 marks*]

Polish notation is an alternative notation for expressions, which has the advantage that every expression can be unambiguously specified without brackets. In Polish notation, operations precede their arguments. So, for example, the bracketed expression not (p and q) and r is expressed in Polish notation by:

$$\mathsf{and\ not\ and}\ p\ q\ r$$

The next part of the question considers how the approach to semantics, used for natural languages, can be applied to the grammar above to translate logical expressions into Polish notation. To this end, we equip the grammar with semantic clauses, whose effect is to compute an expression in Polish notation as the meaning of an Exp expression.

$$\begin{array}{rll} \mathsf{Exp} \rightarrow & \mathsf{Exp1\ Ops} & \{\ \mathsf{Ops.Sem\,(Exp1.Sem)}\ \} \\ \mathsf{Ops} \rightarrow & \epsilon & \{\ \lambda x.\, x\ \} \\ \mathsf{Ops} \rightarrow & \mathsf{and\ Exp1\ Ops} & \{\ \lambda x.\, \mathsf{Ops.Sem\,(and}\ x\ \mathsf{Exp1.Sem)}\ \} \\ \mathsf{Exp1} \rightarrow & p & \{\ p\ \} \\ \mathsf{Exp1} \rightarrow & \mathsf{(Exp)} & \{\mathsf{Exp.Sem}\} \\ \mathsf{Exp1} \rightarrow & \mathsf{not\ Exp1} & \{\mathsf{not\ \ Exp1.Sem}\} \end{array}$$

Note that the semantics of a boolean variable p is just the variable p itself.

QUESTION CONTINUES ON NEXT PAGE

(c) Draw the syntax tree for the expression

$$\mathsf{not}\ p\ \mathsf{and}\ q$$

leaving plenty of room for annotations. Starting at the bottom of the tree, annotate each node with the raw lambda-expression assigned to it by the semantics defined in the table above. [*6 marks*]

(d) Show the sequence of β-reductions by which the lambda-expression associated with the root of this tree reduces to a normal form. [*3 marks*]

(e) Write out an LL(1) grammar that generates the language of logical expressions in Polish notation that can result from the translation process above. [*3 marks*]

8. Consider the following corpus of five parsed sentences, in which the symbols

S NP VP PP Det Nom V Prep

are used as non-terminals, and English words appear as terminals.

- (S (NP I) (VP (V saw) (NP (Det a) (Nom (Nom dog)
  (PP (Prep on) (NP (Det the) (Nom beach)))))))
- (S (NP (Det the) (Nom dog)) (VP (V saw) (NP me)))
- (S (NP I) (VP (V saw) (NP (Det a) (Nom (Nom stick)
  (PP (Prep in) (NP (Det the) (Nom sand)))))))
- (S (NP I) (VP (VP (V threw) (NP (Det the) (Nom stick)))
  (PP (Prep towards) (NP (Det the) (Nom sea)))))
- (S (NP (Det the) (Nom dog)) (VP (V caught) (NP (Det the) (Nom stick))))

(a) Extract the set of production rules that are used to generate the above sentences from the start symbol S, leaving space for adding probabilities. Then use the above corpus to assign a probability to each rule. [*10 marks*]

(b) Explain what it means for a context-free grammar to be in *Chomsky Normal Form* (CNF). State whether or not your grammar from part (a) is in CNF. [*2 marks*]

(c) Using the rule probabilities from part (a), compute the probability of the second sentence in the corpus ('the dog saw me'). [*3 marks*]

(d) Again using the probabilities from part (a), one may now apply the *probabilistic CYK* algorithm to find the most probable parse tree for the sentence

I caught the dog in the sea

Construct the CYK parse chart that results from doing this, including pointers where necessary to show the origin of each non-terminal in the chart. You need not actually compute the probabilities for the entries, but in any situation where a choice between two analyses arises, you should provide a justification for which is the most probable. [*10 marks*]