UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFORMATICS 2A: PROCESSING FORMAL AND NATURAL LANGUAGES

Friday 22nd August 2014

14:30 to 16:30

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: J. Bradfield
External Examiner: C. Johnson

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

PART A
ANSWER ALL QUESTIONS IN PART A.

1. (a) Name the four language classes in the Chomsky hierarchy in order of increasing expressive power. That is, start with the smallest language class and end with the largest language class. [*4 marks*]

   (b) Give three *different* definitions of the smallest language class in the Chomsky hierarchy. [*3 marks*]

   (c) Describe, in a few sentences, the high-level structure of an argument that shows that the three definitions in your answer to part (b) are all *equivalent*; that is, they all define the same class of languages. (The level of detail required here is that you should clearly identify any constructions involved, but you need not describe in detail how the constructions are defined.) [*3 marks*]

2. The code fragment:

```
if X<=A2z Then 123ELSE else0
```

   has the following intended lexing.

```
if   X   <=  A2z  Then  123  ELSE  else0
IF  VAR  OP  VAR  THEN  NUM  ELSE   VAR
```

   The bottom row displays the lexical classes, which, for this example, classify: individual keywords (`IF, THEN, ELSE`), variables (`VAR`), infix operators (`OP`), and numeric literals (`NUM`).

   (a) Use the `egrep` pattern language to give a reasonable definition for each of the six lexical classes above. [*8 marks*]

   (b) Assign priorities to the six lexical classes so that a longest-match lexer would produce the intended lexing behaviour. [*2 marks*]

3. (a) Suppose we are given an arbitrary context-free grammar (CFG) with set of terminals Σ, set of nonterminals N, set of productions P, and start symbol S. List the properties that a tree must possess in order to be a *parse tree* for the grammar. [*5 marks*]

   (b) What does it mean for a CFG to be *structurally ambiguous?* [*1 mark*]

   (c) Give an example of a structurally ambiguous CFG, and justify your claim that it is structurally ambiguous. [*4 marks*]

4. (a) In a *bigram tagger*, we are given, for each word w, a frequency table for the possible parts of speech of w, given the part of speech of the preceding word (if any). In the following examples, rows correspond to potential tags for w itself, and columns to the tag of the preceding word (or start of sentence marker). Note that 'her' may be either an *object pronoun* (OP) or a *possessive pronoun* (PP).

let	N	V	start
N	2	1	5
V	4	2	11

her	N	V	start
OP	2	23	1
PP	6	17	14

duck	N	V	OP	PP	start
N	5	7	3	4	2
V	8	4	6	0	1

fly	N	V	start
N	5	3	1
V	10	4	3

Use these tables to tag the phrase:

let her duck fly

Give the reason for each tag you assign. [*5 marks*]

(b) Explain briefly how the idea of a bigram tagger can be generalized to that of an *n-gram tagger*. Mention one potential advantage of a 4-gram tagger over a bigram tagger, and one problem that such a tagger would typically face. [*3 marks*]

(c) What is the main advantage of the *Viterbi algorithm* over a bigram tagger? Briefly indicate how and why this algorithm might plausibly assign a different tagging to the above phrase. [*2 marks*]

5. The following is a probabilistic context-free grammar with start symbol S:

S → NP V NP (1.0)
NP → N (0.5) | A N (0.3) | NP N (0.2)
V → forecasts (0.2) | calm (0.2) | describe (0.2)
| warns (0.2) | predicts (0.2)
N → forecasts (0.4) | seas (0.4) | office (0.2)
A → calm (0.4) | stormy (0.5) | meteorological (0.1)

(a) Draw all possible parse trees for the sentence

meteorological office forecasts calm seas

and calculate their probabilities, showing how your answers are derived. [*5 marks*]

(b) Now suppose we are given the following sentence with a missing word:

meteorological office forecasts — seas

We are also told that the missing word is one of the following:

calm stormy describe

According to the above grammar, which is the *most probable* of these choices? In other words, which choice yields the word sequence that is most likely to be generated by the above rules? Justify your answer. [*5 marks*]

PART B
ANSWER TWO QUESTIONS FROM PART B.

6. A company that specialises in drilling deep under the earth's surface uses a drill that has two configurations: *Down*, in which the the drill bit can be turned a small angle clockwise to drill down; and *Up*, in which the drill bit can be turned the same angle anticlockwise, to raise the bit. Instructions are given to the drill using words over the alphabet $\Sigma = \{d, u, c\}$. Here: d instructs the drill to turn clockwise and hence move the bit down; u instructs the drill to turn anticlockwise and hence move the bit up; and c instructs the drill to change configuration (either from *Down* to *Up*, or from *Up* to *Down*). In a legal instruction sequence, d instructions are only permitted when the drill is in the *Down* state, and u instructions are only permitted when the drill is in the *Up* state. The drill starts off in the *Down* state.

   The language of valid instruction sequences can be defined mathematically as

$$L = \{x \in \Sigma^* \mid \text{there are an even number of } c \text{ symbols to the left of every } d, \text{ and an odd number of } c \text{ symbols to the left of every } u\}$$

   where "to the left of" means occurring anywhere between the start of the word and the symbol in question.

   (a) Draw a deterministic finite automaton (DFA) that recognises the above language L. [*4 marks*]

   (b) Using your DFA, write out a system of simultaneous equations describing the language L, and solve these equations using Arden's Rule to produce a regular expression for L. [*10 marks*]

   (c) A *test drilling* consists of two phases: first the bit is drilled down to a certain depth; then it is raised back to its starting position. The language of instruction sequences for test drillings can be defined mathematically by:

$$\{d^n\, c\, u^n \mid n \geq 1\}$$

   Say whether the language of test drillings is regular. If it is, justify this by giving an NFA or regular expression for the language. If it is not regular, prove this using the Pumping Lemma. [*7 marks*]

   (d) Give a context-free grammar for the language of test drillings. [*3 marks*]

   (e) After a test, the drill bit may again be lowered to the level reached by the test drilling so that drilling can continue. The language of control sequences for such extended tests is described mathematically by:

$$L = \{d^n\, c\, u^n\, c\, d^n \mid n \geq 1\}$$

   State the level at which this language resides in the Chomsky hierarchy. (You do not need to justify your answer.) [*1 mark*]

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

$$\texttt{add} \quad \texttt{take-away} \quad (\quad) \quad \mathsf{int}$$

where int represents a lexical class of numeric literals (e.g. 0, 23).

$$\begin{aligned} \mathsf{Exp} &\rightarrow \mathsf{Exp1}\ \mathsf{Ops} \\ \mathsf{Ops} &\rightarrow \epsilon \mid \texttt{add}\ \mathsf{Exp1}\ \mathsf{Ops} \mid \texttt{take-away}\ \mathsf{Exp1}\ \mathsf{Ops} \\ \mathsf{Exp1} &\rightarrow \mathsf{int} \mid (\mathsf{Exp}) \end{aligned}$$

(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.

$$2\ \texttt{add}\ 1$$

[*6 marks*]

We next consider how the approach to semantics, used for natural languages, can also be applied in the context of formal languages, such as the one above. To this end, we equip the grammar with semantic clauses, whose effect is to compute a number as the meaning of an expression.

$$\begin{aligned} \mathsf{Exp} &\rightarrow \mathsf{Exp1}\ \mathsf{Ops} && \{\ \mathsf{Ops.Sem}\,(\mathsf{Exp1.Sem})\ \} \\ \mathsf{Ops} &\rightarrow \epsilon && \{\ \lambda x.\, x\ \} \\ \mathsf{Ops} &\rightarrow \texttt{add}\ \mathsf{Exp1}\ \mathsf{Ops} && \{\ \lambda x.\, \mathsf{Ops.Sem}\,(x + \mathsf{Exp1.Sem})\ \} \\ \mathsf{Ops} &\rightarrow \texttt{take-away}\ \mathsf{Exp1}\ \mathsf{Ops} && \{\ \lambda x.\, \mathsf{Ops.Sem}\,(x - \mathsf{Exp1.Sem})\ \} \\ \mathsf{Exp1} &\rightarrow n && \{\ n\ \} \\ \mathsf{Exp1} &\rightarrow (\mathsf{Exp}) && \{\ \mathsf{Exp.Sem}\} \end{aligned}$$

Note that the semantics of a numeric literal n is just the number n itself.

(c) Draw the syntax tree for the expression

$$3\ \texttt{take-away}\ 2\ \texttt{add}\ 1$$

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. [*4 marks*]

(e) The semantics defined above implements the standard convention by which the operations of addition and subtraction *associate to the left*. For example, the expression 3 `take-away` 2 `add` 1 is interpreted as $(3-2)+1$. Martians, however, are rumoured to use the opposite convention, by which these operations *associate to the right*. Thus a Martian would read 3 `take-away` 2 `add` 1 as $3-(2+1)$. How can the grammar's semantic clauses be modified so that the resulting semantics interprets the operations as right associative? [*3 marks*]

8. Consider the following context-free grammar (with start symbol S) for a class of English sentences involving *tag questions*:

S	→	NP VP , TagQ
NP	→	The N
VP	→	Aux NegOpt1 V
TagQ	→	Aux NegOpt2 Pron ?
NegOpt1	→	ϵ | -n't
NegOpt2	→	ϵ | -n't
N	→	eagle | eagles | rocket | rockets
Aux	→	has | have | had | does | do | did
V	→	land | landed | fly | flown
Pron	→	it | they

For example, this grammar generates the sentence:

The eagle has landed, hasn't it?

Note that we understand -n't to be a suffix that attaches itself to the preceding word. The reason for including two different NegOpt non-terminals will appear below.

However, the grammar as it stands also generates many sentences that are ungrammatical in English. For example:

The eagles has landed, hasn't it?
The eagle has landed, hasn't they?
The eagle has landed, doesn't it?
The eagle hasn't landed, hasn't it?

(a) Construct a *parameterized* version of the above grammar which generates only grammatical sentences. Your parameterized grammar should make use of the following attributes and associated variables:

Attribute	Variable	Values
Number	n	sing, plur
Choice of auxiliary	a	have, had, do, did
Polarity	x	pos, neg

You need not write out all of the rules for expanding Aux and V, but should include a representative sample.

To start you off, the first rule of your grammar should be:

S → NP[n] VP[n,a,x] , TagQ[n,a,x]

Notice that you may give different parameterized rules for NegOpt1 and NegOpt2. [*11 marks*]

(b) Using your parameterized grammar, construct a CYK-style parse table for the sentence:

The rocket didn't fly, did it?

(You should do this for your grammar as it stands—do *not* convert it to Chomsky Normal Form.) Each cell in the table may contain one or more parameterized non-terminals, e.g. NP[sing]. Note that the same non-terminal may sometimes occur twice in the same cell but with different parameters.

You should include all possible entries in your table, whether or not they contribute to an overall parse of the sentence. You should not attempt to add entries for *empty* constituents, but should of course include entries for any larger constituents that involve them. Finally, you need not include pointers or other information to show how phrases are broken into their immediate constituents. [*11 marks*]

(c) Suppose now that we wish to implement an *auto-complete* facility for the above class of sentences: the user types the portion of a sentence up to the comma (e.g. "The eagle has landed,") and the system responds by supplying the appropriate tag question (e.g. "hasn't it?"). Briefly outline how such a system might be implemented with the help of the parameterized grammar constructed in part (a) above. [*3 marks*]