Module Title: Informatics 2A
Exam Diet: Aug 2016–17
Brief notes on answers:

1. (a) The equations are

$$\begin{aligned} X_0 &= bX_0 + aX_1 \\ X_1 &= bX_0 + aX_2 \\ X_2 &= bX_0 + aX_2 + \epsilon \end{aligned}$$

[Roughly 1 mark per correct equation.]

(b) Many ways of solving the equations are acceptable. For example, substituting for X_1 in the first equation gives

$$X_0 = bX_0 + a(bX_0 + aX_2) = (b + ab)X_0 + aaX_2$$

So by Arden's rule:

$$X_0 = (b + ab) * aaX_2$$

Substituting this into the third equation:

$$X_2 = b(b + ab) * aaX_2 + aX_2 + \epsilon = (b(b + ab) * aa + a)X_2 + \epsilon$$

By Arden's rule again:

$$X_2 = (b(b + ab) * aa + a)^*$$

so the language accepted by the DFA is

$$X_0 = (b + ab) * aa(b(b + ab) * aa + a)^*$$

[4 marks for clear working, 2 marks for reaching a correct solution, 1 mark for flagging uses of Arden's rule. Note that the language is question consists of all strings ending in aa — this will help in identifying correct solutions.]

2. (a) The accepting computation is

Transition	Input remaining	Stack
	$abba$	$*$
$a, * : a*\ (p \to p)$	bba	$a*$
$b, a : ba\ (p \to p)$	ba	$ba*$
$\epsilon, b : b\ (p \to q)$	ba	$ba*$
$b, b : \epsilon\ (q \to q)$	a	$a*$
$a, a : \epsilon\ (q \to q)$	ϵ	$*$
$\epsilon, * : \epsilon\ (q \to q)$	ϵ	Stack empties

[6 marks — roughly 1 mark per correct transition, with some flexibility.]

(b) The language consists of even-length palindromes over $\{a, b\}$. A context-free grammar for this would be $S \to \epsilon \mid aSa \mid bSb$. This is not LL(1) [nor is any CFG for this language.]

[1 mark for description, 2 marks for the grammar, 1 mark for 'not LL(1)'.]

3. (a) Bookwork. A grammar is in CNF if the RHS of each production consists of either just a single terminal or just two non-terminals [2 marks]. We need to convert a grammar to CNF in order to use the CKY parsing algorithm in its standard form [1 mark].

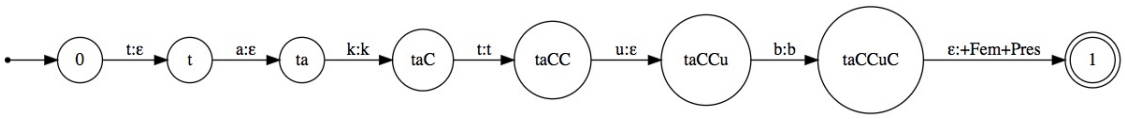
   (b) Several solutions are possible depending on the exact order of applying the steps. A typical one is:

$$\begin{aligned} S &\rightarrow XY \mid AZ \mid DX \\ X &\rightarrow AZ \\ Y &\rightarrow d \mid DX \\ Z &\rightarrow BC \\ A &\rightarrow a \\ B &\rightarrow b \\ C &\rightarrow c \\ D &\rightarrow d \end{aligned}$$

   [3 marks for evidence of understanding, 6 marks for a fully correct solution. The example is chosen to be slightly tricky.]

   (c) In this case, the language of G is the language of G' plus ϵ [1 mark].

4. (a) The idea is to produce an transducer that (1) emits elements of the trilateral root when encountered, (2) remembers the pattern as it is encountered, substituting C for each emitted character, and (3) outputs the set of features corresponding to that pattern when it determines that it has encountered exactly that pattern.



   (b) It is easy to generalize to a fixed set of patterns that work with arbitrary roots. Since the transducer above essentially memorizes its pattern, we can extend it to memorize set of patterns as a prefix tree. It should emit the input character when it is in a C part of a pattern, and otherwise behaves as before.

   (c) To work for a fixed set of roots, the state of the transducer must also remember the consonants of the root as they are encountered. Hence the state set of the transduceer is a cross-product of the set of pattern prefixes and root prefixes, and it can only enter a final state if it has seen a valid pattern and a valid root. (Essentially, it must memorise all possible forms).

5. (a) $\forall x.\textsc{Dog}(x) \Rightarrow \textsc{Barks}(x)$.

   (b) $\exists x.\textsc{Dog}(x) \wedge \forall y.\textsc{Cat}(y) \Rightarrow \neg\textsc{Likes}(x, y)$

   (c) $\exists x.\exists y.\textsc{Dog}(x) \wedge \textsc{Cat}(y) \wedge \neg\textsc{Likes}(x, y)$

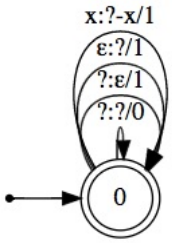
   (d) $\exists x.\exists y.\textsc{Dog}(x) \wedge \textsc{Cat}(y) \wedge \textsc{Chases}(x, y)$

   (e) Two possible solutions:

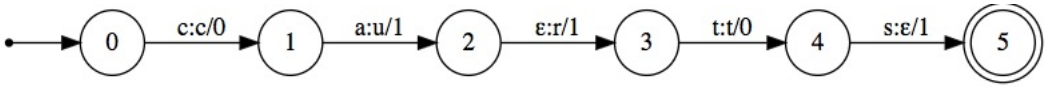
   (i). $\exists y.\forall x.\textsc{Dog}(x) \wedge \textsc{Cat}(y) \wedge \textsc{Chases}(x, y) \Rightarrow \textsc{Catches}(x, y)$.

   (ii). $\forall x.\forall y.\textsc{Dog}(x) \wedge \textsc{Cat}(y) \wedge \textsc{Chases}(x, y) \Rightarrow \textsc{Catches}(x, y)$

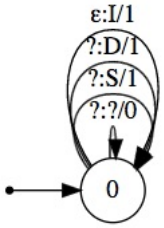
6. (a) The edit transducer looks something like the one sketched below, using ? to stand in for the set of all possible characters. There is only a single state, but there are four types of edges: 26 for copying a character from input to output with cost 0 (i.e. no edit); 26 for deletion; 26 for insertion; and $26 \times 25 = 650$ for substitutions, for a total of 728 edges.



(b) The path through the transducer represents the edit string.



(c) The left edit fst looks like this:



The right edit fst is analogous. Each transducer has one state and 79 edges, for a total of 158 edges.

(d) This is not a problem, because whenever a character substitutes for itself with cost 1, there is another edge that gives the same substitution cost 0.

(e) The chart is below, with a path representing an alternative edit sequence in bold.

	c	a	t	s
c	**0**	1	2	3
u	1	**1**	2	3
r	2	2	**2**	3
t	3	3	2	**3**

(f) An alternative edit string is *cSSS*, shown above.

7. (a) Replace S → NP VB JJ with S → S' JJ and S' → NP VB.

(b) There is one alternative (for two possible binarisations): replace S → NP VB JJ with S → NP S" and S" → VB JJ.

(c) The chart contains items (0,DT,1), (1,NN,2), (1,JJ,2), (2,VB,3), (2,NN,3), (3,JJ,4), (0,NP,2), and (0,S,4)

	PREDICT	0, S → ∘ NP VB JJ, 0
	PREDICT	0, NP → ∘ DT NN, 0
	PREDICT	0, DT → ∘ the, 0
	SCAN	0, DT → the ∘, 1
	COMPLETE	0, NP → DT ∘ NN, 1
	PREDICT	1, NN → ∘ number, 1
	PREDICT	1, NN → ∘ prime, 1
	SCAN	1, NN → number ∘, 2
(d)	COMPLETE	0, NP → DT NN ∘, 2
	COMPLETE	0, S → NP ∘ VB JJ, 2
	PREDICT	2, VB → ∘ number, 2
	SCAN	2, VB → number ∘, 3
	COMPLETE	0, S → NP VB ∘ JJ, 3
	PREDICT	3, JJ → ∘ few, 3
	PREDICT	3, JJ → ∘ prime, 3
	SCAN	3, JJ → few ∘, 4
	COMPLETE	0, S → NP VB JJ ∘, 4

(e) The top-down prediction of Earley prevents the construction of the JJ analysis for *prime* and the NN analysis for *number*, since they are not predicted to appear in the positions where the words actually occur.

(f) Earley corresponds to a right-branching binarization of the grammar.

8. (a) L is actually a regular language. A suitable regular expression for it is

$$((\mathbf{decl} + \mathbf{use})\ \mathsf{char}\ \mathsf{char}^*)^*$$

where $\mathsf{char} = a + b + \cdots + z$.

[1 mark for saying it is regular, 2 marks for a regular expression or NFA.]

(b) Suppose we had a context-free grammar for J. Rebrand **decl** and **use** as non-terminals, add productions $\mathbf{decl} \to \epsilon$ and $\mathbf{use} \to \epsilon$, and we have a context-free grammar for W — contradiction.

(c) We may take R to be defined e.g. by the regular expression

$$\mathbf{decl}\ \mathsf{char}\ \mathsf{char}^*\ \mathbf{use}\ \mathsf{char}\ \mathsf{char}^*$$

Then clearly $J = K \cap R$. It is a standard fact that the intersection of a context-free language with a regular one is context-free. So if K were context-free, so would J be, contradicting the result of (b).

(d) Bookwork. A linear bounded automaton is an automaton with finitely many control states and a memory in the form of a tape of bounded length. Each square of the tape may be filled with a symbol drawn from a finite tape alphabet, and the input string is initially written onto the tape. Moreover, there is a constant k associated with the automaton such that the length of the tape is k times that of the input string (hence 'linear bounded'). There are also special symbols marking the two endpoints of the tape. At any time, the state of the automaton also includes a current 'read position' on the tape, and the possible transitions may depend on both the current control state and the current tape symbol. On a transition, the machine may replace the current tape symbol with another one,

optionally move the read position one place to the left or right (but not passing the endpoints), and jump to a new control state. Acceptance may be signalled e.g. by moving to a special 'accept' state, after which nothing further happens.

[There are about 7 points here to be mentioned — 1 mark each, not being too picky.]

(e) Recall from lectures that by augmenting the tape alphabet, we may in effect maintain *markers* for a finite number of significant positions in the input string. The following algorithm will do the job:

- Use a marker U to scan the whole input from left to right, looking for occurrences of **use**.
- For each such occurrence:
  - Use another marker D to scan the input from the left end up to U, looking for occurrences of **decl**.
  - For each such occurrence, check whether the variable following D matches the variable following U. (This may be done character-by-character, using two more markers for our positions in these strings, scrolling back and forth between them and using the control state to remember the current string symbol.)
  - If the variables match, advance U to the next **use**. If U reaches the right end of the tape, report acceptance.
  - If they don't match, advance D to the next **decl**. If D reaches U, report rejection.

[4 marks for the basic algorithm, accepting evidence of correct understanding and not being too fussy. 1 mark each for acceptance and rejection conditions.]

(f) This means that K is *context-sensitive*. [1 mark.]