Module Title: Informatics 2A
Exam Diet (Dec/April/Aug): December 2013 2013–14
Brief notes on answers:

PART A

1. (a) Bookwork. An LP pipeline is an ordered sequence of processing stages that are applied to a language text or utterance in order to perform some task of interest. (i) For Java, typical stages (in order) would include lexing, parsing, typechecking, code generation, linking and execution. (ii) For spoken English, typical stages would include phonological analysis, word segmentation, POS tagging, parsing, agreement checking, meaning extraction.

   [2 marks for the general concept of a pipeline; 2 marks each for (i) and (ii). Any reasonable stages in a reasonable order will be accepted.]

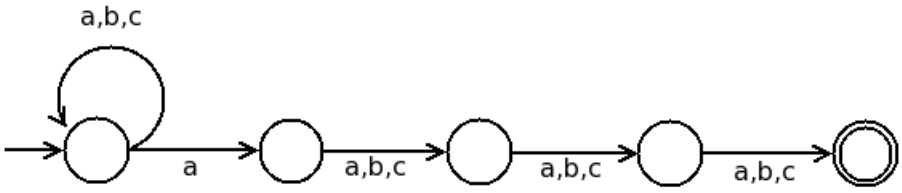
   (b) Again bookwork. Any two of the following (or any other reasonable points) will be accepted.

   - Natural languages are riddled with *ambiguity* at all levels; in formal languages this is generally avoidable by design. This means that in NL it is more a case of determining the most probable interpretation than the single 'correct' one—hence the common use of probabilistic methods in NLP.
   - In formal languages, the pipeline often works in a pure, linear way: each processing stage depends only on the output from the previous stage. In NL, there is less of a clean separation between stages, since information from later stages may often used to resolve ambiguities arising from earlier stages.
   - Natural languages are evolving all the time, and no exhaustive description of a language is possible. NL systems should therefore be designed to be work (as far as possible) in the presence of new words and other previously 'unknown' elements.

   [For each point: 1 mark for the difference, 1 mark for the implications for LP.]

2. [A similar question to this was scrutinised last year but wasn't used because of the ITO burglary]

   (a) One possible NFA:



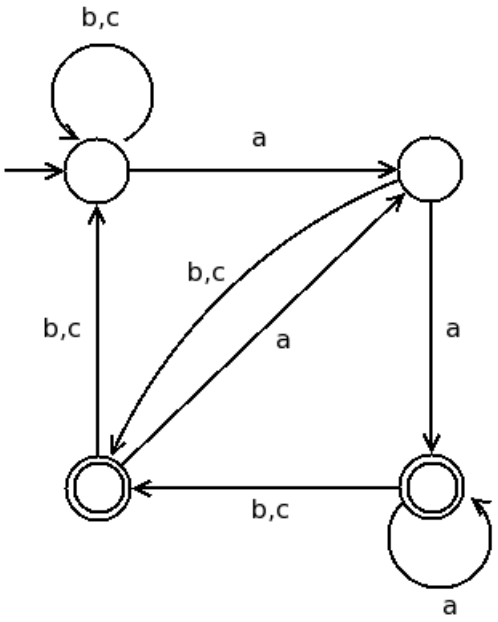
   [4 marks: in proportion to correctness]

   [The students have seen an example very similar to this in lectures.]

   Common error in solution:

   - Including a garbage state. NFAs never need a dedicated garbage state. (Exercise: understand why not!)

   (b) The DFA is below.



[5 marks: in proportion to correctness]

(c) 2^n .

[1 mark]

[This was mentioned in lectures as an example for which determinization is necessarily exponential.]

3. (a) A typical equivalent CNF grammar is:

$$\begin{array}{rcl} \mathsf{Com} & \rightarrow & \mathsf{G\ \ TwoNP} \\ \mathsf{G} & \rightarrow & \text{give} \\ \mathsf{TwoNP} & \rightarrow & \mathsf{NP\ NP} \\ \mathsf{NP} & \rightarrow & \mathsf{DT\ Nom} \\ \mathsf{Nom} & \rightarrow & \text{dog} \mid \text{bone} \mid \mathsf{A\ N} \end{array}$$

plus the existing productions for DT, N, A.

[Roughly, 2 marks for being in CNF, 2 marks for being equivalent to the original grammar.]

(b) For the above CNF grammar, the CYK parse chart is:

	give	the	dog	a	bone
give	G				Com
the		DT	NP		TwoNP
dog			N,Nom		
a				DT	NP
bone					N,Nom

(Different answers to part (a) will yield slightly different charts.) For each entry corresponding to a non-singleton phrase, there should also be pointers leftward and downward to its two immediate children. The down pointer from each NP should point to Nom.

[Roughly 0.5 marks per correct entry; 1.5 marks for pointers and general format.]

4. (a) The parse tree is given by

(BigS (S there is (NP a (Nom (A serious)(N problem)))) if
(S it (WeatherV snows)))

The raw lambda-expression associated with the root is

$$\mathit{Snows} \;\Rightarrow\; \exists\,\mathrm{x}.\,(\lambda\mathrm{y}.\,(\lambda\mathrm{z}.\;\mathit{Serious}(\mathrm{z}))(\mathrm{y}) \wedge (\lambda\mathrm{z}.\;\mathit{Problem}(\mathrm{z}))(\mathrm{y}))(\mathrm{x})$$

with various sub-expressions of this being associated with other nodes in the obvious way.

[1 mark for the right parse tree. 1 mark for evidence of understanding, and about 0.5 marks per correctly annotated node.]

(b) The above expression β-reduces in three steps to

$$\mathit{Snows} \;\Rightarrow\; \exists\,\mathrm{x}.\,\mathit{Serious}(\mathrm{x}) \wedge \mathit{Problem}(\mathrm{x})$$

Various reduction orders are possible.

[2 marks for the correct normal form, 2 marks for displaying the reduction sequence.]

5. (a) Underlining the sequence to be expanded on the next line:

$$\begin{aligned}
\underline{\mathsf{S}} &\Rightarrow\ \mathsf{!}\ \underline{\mathsf{T}} \\
&\Rightarrow\ \underline{\mathsf{!\ A}}\ \mathsf{a\ T} \\
&\Rightarrow\ \mathsf{a\ !\ a}\ \underline{\mathsf{T}} \\
&\Rightarrow\ \mathsf{a\ !}\ \underline{\mathsf{a\ B}}\ \mathsf{b} \\
&\Rightarrow\ \mathsf{a}\ \underline{\mathsf{!\ B}}\ \mathsf{a\ b} \\
&\Rightarrow\ \mathsf{a\ b\ !\ a\ b}
\end{aligned}$$

[7 marks: 1 per correct step, including starting with S]

(b) The language is:

$$\{x\ \mathsf{!}\ x \mid x \in \{\mathsf{a},\mathsf{b}\}^*, |x| \geq 1\}$$

[2 marks: award 1 if idea right but some error in presentation]

(c) Context sensitive.

[1 mark]

PART B

6. (a) (i). Regular languages. E.g., $\{a^n \mid n \geq 0\}$, or even $\emptyset$.
   (ii). Context-free languages. E.g., $\{a^n b^n \mid n \geq 0\}$.
   (iii). Context-sensitive languages. E.g., $\{a^n b^n c^n \mid n \geq 0\}$.
   (iv). Recursively enumerable languages. E.g., the set of encodings of Turing machines that halt when run on an empty tape.

   [8 marks: 1 mark per point]

   Common errors in solutions:

   - Not including an example for (i). Because there are no machine types higher on the list than (i), to answer the question you just need to give an example of a regular language.
   - Saying "grammars" instead of "languages". The question asks for classes of languages.
   - Saying "unrestricted languages" in (iv). The class of languages is *recursively enumerable languages* (which are defined by unrestricted grammars).

   (b) The PDA execution:

action	state	unread input	stack
	$q1$	$aaabba$	$\bot$
$q1 \xrightarrow{a,\bot : a\,\bot} q1$	$q1$	$aabba$	$a\,\bot$
$q1 \xrightarrow{a,a : a\,a} q1$	$q1$	$abba$	$a\,a\,\bot$
$q1 \xrightarrow{a,a : a\,a} q1$	$q1$	bba	$a\,a\,a\,\bot$
$q1 \xrightarrow{b,a : \epsilon} q2$	$q2$	ba	$a\,a\,\bot$
$q2 \xrightarrow{\epsilon,a : \epsilon} q1$	$q1$	ba	$a\,\bot$
$q1 \xrightarrow{b,a : \epsilon} q2$	$q2$	a	$\bot$
$q2 \xrightarrow{\epsilon,\bot : b\,\bot} q1$	$q1$	a	$b\,\bot$
$q1 \xrightarrow{a,b : \epsilon} q1$	$q1$	ϵ	$\bot$
$q1 \xrightarrow{\epsilon,\bot : \epsilon} q1$	$q1$	ϵ	ϵ

   [8 marks: in proportion to completeness/correctness]

   Common errors in solutions:

   - Consuming the current input symbol when an ϵ-transition is applied (leading to, e.g., an unread input of a in row 6 of the table).
   - Considering $\bot$ as the empty stack. It is not, it is a stack with one item. The final transition is needed to empty the stack.

   (c) The language L:

$$\{x \in \{a,b\}^* \mid \#_a(x) = 2 \times \#_b(x)\}$$

   where $\#_c(x)$ denotes number of occurrences of character c in string x.

   [2 marks: award 1 if idea right but some error in presentation]

   (d) We use the pumping lemma to show the language is not regular.

   [1 mark]

   We show $\neg$ P (the negation of the pumping property).

Suppose $k \geq 0$.

Consider $x = \epsilon$, $y = b^k$ and $z = a^{2k}$. Then $xyz = b^k a^{2k} \in L$ and clearly $|y| \geq k$.

Suppose $y = uvw$ where $|v| \geq 1$.

Then $uv^0w = uw = b^m$ for some $m < k$. Whence $xyv^0wz = b^m a^{2k} \notin L$ since $\#_a(b^m a^{2k}) = 2k > 2m = 2 \times \#_b(b^m a^{2k})$.

Thus the pumping property fails for $i = 0$.

[6 marks: in proportion to completeness of argument]

[Direct proofs not using the pumping lemma are also acceptable here. In any case, split marks as: 1 for setting scene for proof, plus 6 for proof details.]

[Correctness of answer will be assessed on basis of proving the language the student gives as answer to part (c) cannot be recognised by a DFA. Thus, full marks can be obtained for part (d) even if part (c) is incorrect.]

Common errors in solutions:

- Choosing fixed strings for x, y, z; e.g., $x = aaa$, $y = bb$, $z = a$. For every $k \geq 0$, the proof needs to work with xyz where y satisfies $|y| \geq k$.
- Choosing specific values for u, v, w; e.g., $u = b^{k-1}$, $v = b$, $w = \epsilon$. For the argument to be correct, it has to work for every uvw for which $y = uvw$ and $|y| > 0$.
- Saying $y = uvw$ in one line and then $y = uv^iw$ or $y = uw$ in another. If $|v| \geq 1$ then $y = uvw$ and $y = uw$ cannot both be true.

7. [A similar question to this was set last year but was not used because of the ITO burglary.]

   (a) The parse trees are given by:

   - (S place (NP (NP (Noun chips))(PP (Prep on)(NP (Noun tray))))
     (PP (Prep in)(NP (Noun oven))))
   - (S place (NP (Noun chips)) (PP (Prep on)(NP (NP (Noun tray))
     (PP (Prep in)(NP (Noun oven))))))

   [1 mark each.]

   (b) The execution of Earley parsing is as follows. (There are also other spurious steps they might choose to show.)

S → • place NP PP	[0,0]	P
S → • remove NP from NP	[0,0]	P (spurious)
S → place • NP PP	[0,1]	S
NP → • Noun	[1,1]	P
NP → • NP PP	[1,1]	P (spurious)
NP → Noun •	[1,2]	S
NP → NP • PP	[1,2]	C (spurious)
S → place NP • PP	[0,2]	C
PP → • Prep NP	[2,2]	P
PP → Prep • NP	[2,3]	S
NP → • Noun	[3,3]	P
NP → • NP PP	[3,3]	P (spurious)
NP → Noun •	[3,4]	S
PP → Prep NP •	[2,4]	C
S → place NP PP •	[0,4]	C (successful parse)

[9 marks: 3 for understanding of basic method; 6 for the details.]

(c) Probabilistic CFGs wouldn't help for this pair of trees. The same rules feature the same number of times but in a different order, so whatever the rule probabilities were, the two trees would have the same probability.

[2 marks; either they see it or they don't.]

(d) Slightly more challenging. An example of an equivalent LL(1) grammar is:

S → place Noun PP | remove NP from NP
NP → Noun PPopt
PPopt → ϵ | PP
PP → Prep NP

Many other solutions are possible.

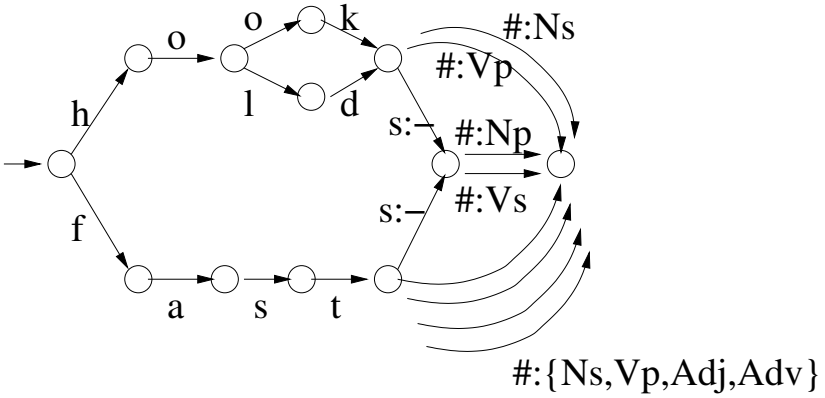
[2 marks for an LL(1) grammar not equivalent to the given one. 3 marks for an equivalent grammar that isn't quite LL(1). 5 marks for a fully correct solution.]

(e) This will depend on their solution to the previous part. For the above LL(1) grammar, the parse table is

	place	remove	from	Noun	Prep	$
S	place Noun PP	remove NP from PP				
NP				Noun PPopt		
PPopt			ϵ		PP	ϵ
PP					Prep NP	

[Up to 7 marks. Full credit is available for a solution that's correct relative the grammar given in (d), if it's LL(1). 3 marks for evidence of understanding; 4 marks for the details.]

8. (a) The required transducer is:



(b) The Viterbi matrix is as follows. Pointers are to cells in the previous column.

	hook	hold	fast
N	0.6x0.5 = 0.3	0.3x0.4x0.3 = 0.036 (points to N)	0.036x0.4x0.2 = 0.00288 (points to N)
V	0.2x0.3 = 0.06	0.3x0.3x0.5 = 0.045 (points to N)	0.036x0.3x0.2 = 0.00216 (points to N)
Adj	0	0	0.045x0.2x1.0 = 0.009 (points to V)
Adv	0	0	0.045x0.3x1.0 = 0.0135 (points to V)

So the optimal tagging is N V Adv.

[2 marks for the correct result. For the Viterbi matrix, roughly 1 mark per correct non-zero entry, being somewhat lenient towards minor clerical errors.]

(c) To tag a sequence such as 'hook holds fast': first run the transducer on each word to extract the stem and a set of possible fine-grained tags. Then run Viterbi on the sequence of stems to obtain a coarse-grained tagging. Finally, for each word, select the (unique possible) fine tag that is compatible with the coarse tagging. E.g. for 'holds', the stem is 'hold', the possible fine tags are Np and Vs, the coarse tag is V, so the chosen fine tag is Vs.

[2 marks for the method; 1 mark for sensible reference to the example.]