06308

THE UNIVERSITY *of York*

B.Sc., B.Eng and M.Eng Examinations 1995

DEPARTMENT OF COMPUTER SCIENCE

Part IIb

Natural Language Processing

Time allowed: **Three hours**

Silent non-programmable calculators may be used.
Candidates should answer not more than **four** questions from this paper.

1 (25 marks)

(i) [6 marks]

(a) 2 In computational linguistics, we are interested in *linguistic competence*, not *linguistic performance*.

Explain what is meant by this statement.

(b) 2 Show, by giving an example, that English linguistic competence must be described by a grammar whose formal power is at least type 2 (context-free grammar).

(c) 2 Give two reasons why natural language processing researchers might interest themselves in linguistic performance as well as linguistic competence.

(ii) [5 marks]

(a) 3 Write a *Definite Clause* **Grammar** that generates the language $a^n b^n c^n$.

(b) 2 Your answer to part (ii)(a) shows that Definite Clause Grammars can generate context-*sensitive* languages.

Give two reasons why we generally prefer to use formalisms whose formal power is no higher than is needed to account for the syntactic constructions found in the language (i.e. context-*free* in the case of English).

(iii) [5 marks] Consider the following fragment of a grammar and a lexicon:

$$\begin{array}{rlr} \mathsf{S} \rightarrow \mathsf{NP\ VP} & & \textit{George} : \mathsf{NP} \\ \mathsf{VP} \rightarrow \mathsf{Vi} & & \textit{dreamt} : \mathsf{Vi} \\ \mathsf{VP} \rightarrow \mathsf{Vd\ NP\ NP} & & \textit{gave} : \mathsf{Vd} \end{array}$$

(a) $3\frac{1}{2}$ Axiomatise these grammar and lexicon fragments as clauses of *Definite Clause* **Logic**.

(b) $1\frac{1}{2}$ Explain the advantages of the axiomatisation.

(iv) [5 marks] Consider the following grammar and lexicon:

$$\begin{array}{ll} \mathsf{S} \rightarrow \mathsf{NP\ VP} & \mathsf{S}' = \mathsf{VP}'(\mathsf{NP}') \\ \mathsf{VP} \rightarrow \mathsf{Vi} & \mathsf{VP}' = \mathsf{Vi}' \\ \mathsf{VP} \rightarrow \mathsf{Vt\ NP} & \mathsf{VP}' = \mathsf{Vt}'(\mathsf{NP}') \end{array}$$

(where α' represents the semantic translation of syntactic category α)

$$\begin{array}{rc} \textit{Lennie} : \mathsf{NP} & l \\ \textit{George} : \mathsf{NP} & g \\ \textit{dreamt} : \mathsf{Vi} & \lambda x[dream1(x)] \\ \textit{shot} : \mathsf{Vt} & \lambda y[\lambda x[shot1(x, y)]] \end{array}$$

Give a direct Prolog implementation of this grammar and lexicon as a top-down parser, including interleaved, syntax-driven semantic translation (using an extra argument to each Prolog predicate). Define any auxiliary predicates that you use.

(v) [4 marks] State the advantages and disadvantages of interleaved, syntax-driven semantic translation.

2 (25 marks)

(i) [5 marks]

(a) 2 Explain, with examples, the difference between *term unification* and *graph unification*.

(b) 3 In a Definite Clause Grammar, the grammar still has a *monadic CF-PSG backbone*. Explain what this means and exemplify the *advantages* of formalisms that *abandon* such a backbone.

(ii) [10 marks] Consider the following grammar and lexicon:

$$\begin{array}{lrr} \mathsf{VP} \rightarrow \mathsf{Vaux\ VP} & \textit{could} : \mathsf{Vaux} & \textit{sleeps} : \mathsf{Vi(VFORM(finite))} \\ \mathsf{VP} \rightarrow \mathsf{Vi(VFORM(}x\mathsf{))} & \textit{have} : \mathsf{Vaux} & \textit{sleep} : \mathsf{Vi(VFORM(nonfinite))} \\ & \textit{has} : \mathsf{Vaux} & \textit{sleeping} : \mathsf{Vi(VFORM(pres\_participle))} \\ & \textit{been} : \mathsf{Vaux} & \textit{slept} : \mathsf{Vi(VFORM(past\_participle))} \\ & \textit{be} : \mathsf{Vaux} & \end{array}$$

The grammar overgenerates (e.g. it generates *"*have could been sleep*" and *"*has was sleep*" and many other inappropriate sequences of verbs).

Use the information given in the lexicon about the feature VFORM to add features to the lexical entries for the Vauxs and to the rules of the grammar to overcome this source of overgeneration. (Specifically, you will want to enforce the constraint that the *form* of the VP that follows a Vaux is the form that the Vaux *requires*.) Make your answer as concise as possible.

(iii) [10 marks] Consider the following grammar and lexicon for the (made up) language Examinese:

$$
\begin{aligned}
\mathsf{S(GAPIN}(x)\mathsf{, GAPOUT}(y)\mathsf{)} &\rightarrow \mathsf{NP(GAPIN}(x)\mathsf{, GAPOUT}(x)\mathsf{, PLU}(z)\mathsf{)} \\
&\ \mathsf{VP(GAPIN}(x)\mathsf{, GAPOUT}(y)\mathsf{)} \\
\mathsf{S(GAPIN}(x)\mathsf{, GAPOUT}(x)\mathsf{)} &\rightarrow \mathsf{NP(GAPIN}(x)\mathsf{, GAPOUT}(x)\mathsf{, PLU}(z)\mathsf{)} \\
&\ \mathsf{S(GAPIN([NP}|x\mathsf{]), GAPOUT}(x)\mathsf{)} \\
\mathsf{NP(GAPIN([NP}|x\mathsf{]), GAPOUT}(x)\mathsf{, PLU}(z)\mathsf{)} &\rightarrow \epsilon \\
\mathsf{VP(GAPIN}(x)\mathsf{, GAPOUT}(y)\mathsf{)} &\rightarrow \mathsf{Vt(GAPIN}(x)\mathsf{, GAPOUT}(x)\mathsf{, PLU}(z_1)\mathsf{)} \\
&\ \mathsf{NP(GAPIN}(x)\mathsf{, GAPOUT}(y)\mathsf{, PLU}(z_2)\mathsf{)}
\end{aligned}
$$

$$
\begin{aligned}
\textit{Lennie}: &\ \mathsf{NP(GAPIN}(x)\mathsf{, GAPOUT}(x)\mathsf{, PLU(-))} \\
\textit{laps}: &\ \mathsf{NP(GAPIN}(x)\mathsf{, GAPOUT}(x)\mathsf{, PLU(+))} \\
\textit{amt}: &\ \mathsf{Vt(GAPIN}(x)\mathsf{, GAPOUT}(x)\mathsf{, PLU(-))} \\
\textit{amts}: &\ \mathsf{Vt(GAPIN}(x)\mathsf{, GAPOUT}(x)\mathsf{, PLU(+))}
\end{aligned}
$$

The grammar generates Examinese sentences such as the following (shown with their English translations):

1. *"Lennie amts laps."* (Lennie likes rabbits.)
2. *"Laps, Lennie amts!"* (Rabbits, Lennie likes!)

(Obviously, the grammar does not generate the punctuation in (1) and (2) above; the punctuation is included only to help you make sense of what the sentences say.)

(a) 2 Draw a phrase-structure tree (including the features and their values for each category in the tree) that would result from parsing sentence (2) (*"Laps, Lennie amts!"*) for root category S(GAPIN([ ]), GAPOUT([ ])). (Ignore the punctuation again.)

(b) 4 *Briefly* explain *what* the GAPIN and GAPOUT features achieve and *how* they achieve it. (Your explanation may refer to your answer to part (iii)(a) if you wish.)

(c) 4 In Examinese, the value of the PLU feature of the transitive verb, Vt, should agree with the value of the PLU feature of the *object* NP (this being the NP that is ordinarily dominated by the VP node). (This *is* the case in sentences (1) and (2) above, and they are therefore both grammatical; but the following are not grammatical: *"*Lennie amt laps*" and *"*Laps, Lennie amt*". Note that this agreement pattern is different from that found in English, where the transitive verb would agree with the *subject* NP.)

Leaving the lexicon unchanged, modify the grammar so that this agreement is enforced.

3 (25 marks)

The following incomplete Prolog program implements a simple shift-reduce for BH-GRN grammars. (A small BH-GRN grammar is also shown.)

```
parse(Stack, Stack, Input, Input).

parse(CurrentStack, DesiredStack, CurrentInput, DesiredInput) :-
   shift(CurrentStack, NewStack, CurrentInput, NewInput),
   parse(NewStack, DesiredStack, NewInput, DesiredInput).

parse(CurrentStack, DesiredStack, CurrentInput, DesiredInput) :-
   reduce(CurrentStack, NewStack, CurrentInput, NewInput),
   parse(NewStack, DesiredStack, NewInput, DesiredInput).

s  ---> [np, vp].
vp ---> [v, np].
np ===> [george].
np ===> [lennie].
v  ===> [shot].
```

(i) [5 marks] Complete the program by giving definitions of `shift/4` and `reduce/4`. Give definitions of any auxiliary predicates that you use. Make clear how you are assuming that the stack and input buffer are implemented.

(ii) [4 marks]

Give four uses of compilation that are peculiar to computational linguistics.

(iii) [5 marks]

Give the Prolog program that results from *partial execution* of the shift-reduce parser whose definition you completed in part (i) over the BH-GRN grammar. (You should carry out as much partial execution as you can, to produce a grammar-specific shift-reduce parser.)

(iv) [7 marks]

The Prolog representation of the canonical LR(0) characteristic finite state machine (FSM) of the BH-GRN grammar given earlier is:

```
transition(push, 0, succ, s).
transition(shift, 0, 1, george).
transition(shift, 0, 2, lennie).
transition(push, 0, 3, np).
transition(reduce, 1, np, [george]).
transition(reduce, 2, np, [lennie]).
transition(shift, 3, 4, shot).
transition(push, 3, 5, v).
transition(push, 3, 6, vp).
transition(reduce, 4, v, [shot]).
transition(shift, 5, 1, george).
transition(shift, 5, 2, lennie).
transition(push, 5, 7, np).
transition(reduce, 7, vp, [np,v]).
transition(reduce, 6, s, [vp,np]).
```

Give the *full* Prolog definition of a table-driven shift-reduce parser that could traverse an FSM expressed in this form. Again make clear how you are assuming the stack and input buffer are represented.

(v) [4 marks]

(a) 2 Define, with examples, what is meant by the terms *local ambiguity* and *global ambiguity*.

(b) 2 In what sense is the use of an FSM to drive a shift-reduce parser a form of partial execution?

4 (25 marks)

This question uses the Prolog implementation of a chart parser for BH-GRN grammars shown on the next page. (It is simpler but less flexible than the one covered in the lectures.)

Lines of the program are numbered for ease of reference from the question. You should also use them in your answers.

Although it has not been shown, you should assume a standard definition of `append/3`, *viz* `append(L1, L2, L3)` is true if list `L3` is the result of concatenating list `L2` onto the end of list `L1`.

```
parse(Input) :-
    process_each_word(0, Input).

process_each_word(VtxJ, []).

process_each_word(VtxI, [Word|Words]) :-
    VtxJ is VtxI + 1,
    create_lex_edges(VtxI, VtxJ, Word),
    process_each_word(VtxJ, Words).

create_lex_edges(VtxI, VtxJ, Word) :-
    findall(edge(VtxI, VtxJ, Cat, []),
             (Cat ===> [Word]),
             Edges),
    add_to_chart(Edges).

add_to_chart([]).

add_to_chart([edge(VtxI, VtxJ, Cat, Need)|Edges]) :-
    edge(VtxI, VtxJ, Cat, Need), !,
    add_to_chart(Edges).

add_to_chart([edge(VtxJ, VtxK, CCat, [])|Edges0]) :-
    assert(edge(VtxJ, VtxK, CCat, [])),
    findall(edge(VtxJ, VtxJ, LHS, [CCat|RHS]),
             (LHS ---> [CCat|RHS]),
             Edges1),
    findall(edge(VtxI, VtxK, ICat, NCats),
             edge(VtxI, VtxJ, ICat, [CCat|NCats]),
             Edges2),
    append(Edges0, Edges1, Edges3),
    append(Edges3, Edges2, Edges),
    add_to_chart(Edges).

add_to_chart([edge(VtxI,VtxJ,ICat,[NCat|NCats])|Edges0]):-
    assert(edge(VtxI, VtxJ, ICat, [NCat|NCats])),
    findall(edge(VtxI, VtxK, ICat, NCats),
             edge(VtxJ, VtxK, NCat, []),
             Edges1),
    append(Edges0, Edges1, Edges),
    add_to_chart(Edges).
```

(i) [3 marks] State three advantages of chart parsing.

(ii) [3 marks] Explain in detail why it can be argued that a bottom-up chart parser does not work exclusively bottom-up (i.e. why it has a top-down element to its processing).

(iii) [3 marks] Is the Prolog parser given above a left-to-right parser or a right-to-left parser? Explain your answer in detail with reference to the code.

(iv) [3 marks] Is the Prolog parser given above a bottom-up parser or a top-down parser? Explain your answer in detail with reference to the code.

(v) [3 marks] Is the Prolog parser given above one whose search strategy is depth-first or breadth-first? Explain your answer in detail with reference to the code.

(vi) [3 marks] We want to change the parser's search strategy so that it is the opposite of what you said it was in part (v).

Give the line numbers and the revised Prolog for each line that must change.

(vii) [4 marks] What does a *shaper test* test for? How would you modify the Prolog parser to incorporate shaper tests? (You need not give any Prolog code in this answer, but you should use the line numbers to make it clear which parts of the program need changing.)

(viii) [3 marks] In the Prolog program, edges contain four pieces of information. List three other types of information that can be held in edges. For each that you list, state why it might be useful.

5 (25 marks)

(i) [6 marks] Briefly explain what is meant by each of the following terms:

(a) $1\frac{1}{2}$ Truth-conditional semantics

(b) $1\frac{1}{2}$ Model-theoretic semantics

(c) $1\frac{1}{2}$ Compositionality

(d) $1\frac{1}{2}$ The rule-to-rule hypothesis

(ii) [3 marks]

(a) $1\frac{1}{2}$ What is the role of *logic* in assigning semantics to expressions of natural languages?

(b) $1\frac{1}{2}$ What is the role of *type theory* in assigning semantics to expressions of natural languages?

(iii) [14 marks]

You are given the following tables of information:

Syntax rule	Semantic translation rule	Semantic type of LHS
S → NP VP	S′ = NP′(VP′)	**t**

(where α' represents the semantic translation of syntactic category α)

Word : Category	Logic translation	Semantic type
George : NP	$\lambda P[P(g)]$	$\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$
Lennie : NP	$\lambda P[P(l)]$	$\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$
worked : VP	$\lambda x[work1(x)]$	$\langle \mathbf{e}, \mathbf{t}\rangle$
dreamt : VP	$\lambda x[dream1(x)]$	$\langle \mathbf{e}, \mathbf{t}\rangle$

(a) 4 Complete the following entries:

Syntax rule	Semantic translation rule	Semantic type of LHS
VP → VP CVP		
CVP → Conj1 VP		

Word : Category	Logic translation	Semantic type
and : Conj1		

(b) 4 Complete the following entries:

Syntax rule	Semantic translation rule	Semantic type of LHS
S → S CS		
CS → Conj2 S		

Word : Category	Logic translation	Semantic type
and : Conj2		

(c) 6 Complete the following entries:

Syntax rule	Semantic translation rule	Semantic type of LHS
NP → NP CNP		
CNP → Conj3 NP		

Word : Category	Logic translation	Semantic type
and : Conj3		

Here is some help with this one: an example of a typical lambda abstraction over an expression of type $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$ is:

$$work1(l) \equiv \lambda\mathcal{P}[\mathcal{P}(\lambda x[work1(x)])](\lambda Q[Q(l)])$$

(iv) [2 marks]

State the main weakness of your treatment of conjunction in part (iii) above.

6 (25 marks)

(i) [3 marks] Briefly state, with examples, what you understand by the following three types of lexical ambiguity:

(a) 1 Categorial ambiguity

(b) 1 Homonymy

(c) 1 Polysemy

(ii) [5 marks] Some theorists consider *scope ambiguities* to be subtypes of structural ambiguities or of lexical ambiguities. Describe the positions of these two groups of theorists. What disadvantages are there for these two positions?

(iii) [12 marks] You are given the following incomplete tables of information:

Syntax rule	Qfsem rule	Store rule	Semantic type of LHS
S → NP VP	S′ = NP′(VP′)		**t**
NP → PName			
NP → Det N			
VP → Vt NP	VP′ = Vt′(NP′)		

(where α' represents the semantic translation of syntactic category α)

Word : Category	Logic translation	Store	Semantic type of translation
Lennie : PName	$\lambda P[P(l)]$		$\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$
every : Det			
some : Det			
worker : N			
mouse : N			
killed : Vt	$\lambda\mathcal{P}[\lambda y[\mathcal{P}(\lambda x[killed1(y,x)])]]$		$\langle\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle, \langle \mathbf{e}, \mathbf{t}\rangle\rangle$

Complete all entries in the table, with the store rules of the grammar and the stores of the lexicon implementing a simple version of *Cooper storage*.

(iv) [5 marks] Briefly **describe** and **compare** the use of selection restrictions and semantic association (as implemented using marker passing) as strategies for disambiguation.