063173

THE UNIVERSITY of York

B.Sc., B.Eng. and M.Eng. Examinations 1997

DEPARTMENT OF COMPUTER SCIENCE

Part IIIb

Natural Language Processing

Time allowed: **Three hours**

Silent non-programmable calculators may be used.
Candidates should answer **four** questions from this paper.

1 (25 marks)

(i) [5 marks]

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

(b) [2 marks] Your answer to part (i)(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).

(ii) [20 marks] An incompetent computational linguist writes the following (illegal) Definite Clause Grammar:

$$
\begin{array}{rcl}
\mathsf{S} & \rightarrow & \mathsf{NP}\ \mathsf{VP} \\
\mathsf{NP} & \rightarrow & \mathsf{PName} \\
\mathsf{NP} & \rightarrow & \mathsf{Pro} \\
\mathsf{PP}(\mathsf{PFORM}(x)) & \rightarrow & \mathsf{P}(\mathsf{PFORM}(x))\ \mathsf{NP} \\
\mathsf{VP} & \rightarrow & \mathsf{V}(xs)\ xs
\end{array}
\qquad
\begin{array}{rl}
\textit{attacks}: & \mathsf{V}([\ ]) \\
\textit{destroys}: & \mathsf{V}([\mathsf{NP}]) \\
\textit{brings}: & \mathsf{V}([\mathsf{NP},\ \mathsf{PP}(\mathsf{PFORM}(\mathsf{to}))]) \\
\textit{to}: & \mathsf{P}(\mathsf{PFORM}(\mathsf{to})) \\
\textit{Mars}: & \mathsf{PName} \\
\textit{Death}: & \mathsf{PName} \\
\textit{she}: & \mathsf{Pro} \\
\textit{her}: & \mathsf{Pro}
\end{array}
$$

(a) [3 marks] The linguist claims that this grammar *lexicalises subcategorisation*. Explain what is meant by *subcategorisation* and what it means to have a grammar in which subcategorisation information is *lexicalised*.

(b) [2 marks] Why is the grammar not a legitimate Definite Clause Grammar?

(c) [12 marks] Revise the grammar and lexicon from above. Your revision should accomplish three goals (4 marks per goal).

Firstly, it should remedy the problem that you identified in part (ii)(b).

Secondly, it should incorporate changes that enforce a proper distribution of pronouns. For example, these are among the sentences that *should* be generated:

"She destroys Mars."
"Mars destroys her."
"Mars brings Death to her."

but these are among those strings that should *not* be generated:

*"Her destroys Mars."
*"Mars destroys she."
*"Mars brings Death to she."

And, thirdly, your revision should assign a *left-branching* phrase-structure to VPs (in which a verb is combined with one of its complements at a time).

(d) [3 marks] Draw the phrase-structure tree that your grammar assigns to the sentence

"Mars brings Death to her."

2 (25 marks)

This question refers to the following Prolog implementation of a *left-corner* parser for CF-PSGs encoded in Bridge&Harlow Grammar Rule Notation. Lines of the program are numbered for ease of reference from the question. You should also use them in your answers. It might help your understanding of this algorithm to know that, while this program does not use a chart, the way that it deals with the lexicon and the way that it uses the grammar when a complete constituent has been found are similar to these processes in bottom-up chart parsing.

```
100 parseCat(Cat, [Word|L0], L) :-
110    (LexCat ===> [Word]),
120    complete(LexCat, Cat, L0, L).
130
140 complete(Cat, Cat, L, L).
150
160 complete(C1, C2, L0, L) :-
170    (C ---> [C1 | Cs]),
180    parseList(Cs, L0, L1),
190    complete(C, C2, L1, L).
200
210 parseList([], L, L).
220
230 parseList([C|Cs], L0, L) :-
240    parseCat(C, L0, L1),
250    parseList(Cs, L1, L).
```

The parser is run by issuing a query such as

```
?- parseCat(s, [mars, attacks], []).
```

(i) [15 marks]

(a) [2 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.

(b) [3 marks] The Prolog parser given above is primarily a bottom-up parser but it has a small degree of top-down processing. Explain this claim in detail with reference to the code.

(c) [2 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.

(d) [8 marks] A category `LC` is a left-corner of some other category `C` if there is a rule whose left-hand side is `C` and whose right-hand side has `LC` as its leftmost category, `C ---> [LC | Cs]`. The reflexive, transitive closure of the left-corner relation gives what is called a *link table*.

For example, the link table for the following (incomplete) grammar and lexicon

```
a ---> [b, c].  b ---> [f, g, h].
a ---> [d, e].  f ===> [i].
```

is

```
link(b, a).  link(f, b).
link(d, a).  link(i, b).
link(f, a).  link(i, f).
link(i, a).  link(X, X).
```

One source of inefficiency in the left-corner parser can be eliminated by making use of a link table.

- Explain carefully what the inefficiency is;
- give, in Prolog, the modifications you would make to the program to make it use the link table; and
- explain how your modifications eliminate the inefficiency.

(ii) [10 marks] A natural language processing expert wants to build a system to give the general public the ability to interrogate, e.g., telephone directories and rail timetables using *spoken* natural language input.

He has been working on modules that hypothesise what words there are in a user's spoken input signal, and he has also decided that semantic translation must be interleaved with parsing. He is now considering what kind of parser to use.

(a) [3 marks] State **three** criteria that he might use for judging different parsers, **briefly** justifying their relevance to the task in hand.

(b) [7 marks] Three parsers are under consideration: a shift-reduce parser, the left-corner parser given above, and a *bidirectional* variant of the left-corner parser. The bidirectional parser does not parse exclusively left-to-right nor exclusively right-to-left. Instead, it can scan over some words of the input, to make a more informed choice about which word to look up in the lexicon next, and then parse both to the left and to the right of this word.

For each of the three criteria you identified in part (ii)(a), compare the three parsers.

3 (25 marks)

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

(ii) [4 marks] The *Fundamental Principle* in chart parsing creates *new* edges; it does not modify existing edges. Explain carefully what goes wrong if, instead of spawning new edges, the Fundamental Principle *modifies* incomplete edges. (Use diagrams in your answer but restrict your attention to CF-PSG parsing.)

(iii) [8 marks] For the following CF-PSG

$$\begin{array}{rl} \mathsf{S} \rightarrow \mathsf{NP}\ \mathsf{VP} & \quad \textit{Mars} : \mathsf{NP} \\ \mathsf{NP} \rightarrow \mathsf{NP}\ \mathsf{Conj}\ \mathsf{NP} & \quad \textit{attacks} : \mathsf{V} \\ \mathsf{VP} \rightarrow \mathsf{V} & \\ \mathsf{VP} \rightarrow \mathsf{V}\ \mathsf{NP} & \end{array}$$

draw the chart that would be produced by *bottom-up*, *left-to-right* chart parsing of the string

"Mars attacks"

(If you number each edge in the chart, the numbering need not reflect the order in which edges were created or the order in which edges were added to the chart.)

(iv) [5 marks] Would the chart be different had you parsed *top-down* instead of bottom-up? Explain your answer **in detail**.

(v) [5 marks] Chart parsers avoid duplicated effort. In some implementations, this is done by recording what 'seeks' have been carried out and using this record to avoid 're-seeking'. In other implementations, an edge that is to be added to the chart is discarded if it is a duplicate of one that is already on the chart. Compare these two approaches. (Restrict your attention to CF-PSG parsing.)

4 (25 marks)

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

(a) [$1\frac{1}{2}$ marks] Truth-conditional semantics;

(b) [$1\frac{1}{2}$ marks] Model-theoretic semantics;

(c) [$1\frac{1}{2}$ marks] Compositionality; and

(d) [$1\frac{1}{2}$ marks] The rule-to-rule hypothesis.

(ii) [3 marks]

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

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

(iii) [16 marks] Here are three sentences with their desired logic translations:

"Mars attacks"	*attack1(m)*
"Mars fights Neptune"	*fight1(m, n)*
"Mars brings Death to Neptune"	*bring1(m, d, n)*

You are also given the following incomplete tables of information:

Syntax rule	Semantic translation rule	Semantic type of LHS
S $\rightarrow$ NP VP		$\mathbf{t}$
NP $\rightarrow$ PName	NP′ = PName′	$\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$
VP $\rightarrow$ Vi	VP′ = Vi′	$\langle\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle, \mathbf{t}\rangle$
VP $\rightarrow$ Vt NP		
VP $\rightarrow$ Vd NP PP		
PP $\rightarrow$ *to* NP	PP′ = NP′	$\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$

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

Word : Category	Logic translation	Semantic type
Mars : PName		
Neptune : PName		
Death : PName		
attacks : Vi	$\lambda\mathcal{P}[\mathcal{P}(\lambda x[attack1(x)])]$	
fights : Vt		
brings : Vd		

Complete all the missing entries. Show your working. (Warning! Do not attempt to take a short-cut by borrowing material from Question 5. The two questions do not necessarily use the same semantic types for verbs and verb phrases.)

5 (25 marks)

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

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

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

$$\begin{array}{rc} \mathit{Mars} : \mathsf{NP} & m \\ \mathit{Neptune} : \mathsf{NP} & n \\ \mathit{explodes} : \mathsf{Vi} & \lambda x[\mathit{explode}1(x)] \\ \mathit{attacks} : \mathsf{Vt} & \lambda y[\lambda x[\mathit{attack}1(x, y)]] \end{array}$$

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.

(ii) [14 marks] The semantics in part (i) only handles simple NPs. In this part of the question, we consider some of the problems of handling quantified NPs.

The standard logic translation into logic for the NP *"every starship"* is

$$\lambda P_1[\forall x_1(ss1(x_1) \supset P_1(x_1))],$$

for *"some planet"* it is

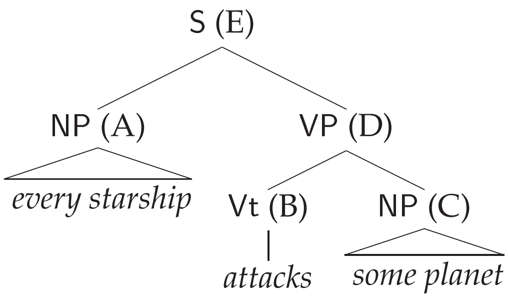
$$\lambda P_2[\exists x_2(pl1(x_2) \wedge P_2(x_2))]$$

and for the Vt *"attacks"* it is

$$\lambda \mathcal{P}[\lambda y[\mathcal{P}(\lambda x[\mathit{attack1}(y, x)])]]$$

(taking *ss*1, *pl*1 and *attack1* to be the predicates that denote starships, planets and attacking, respectively).

Given that the standard translation for an S is computed by S′ = NP′(VP′), and that the standard translation for a VP that contains a transitive verb is computed by VP′ = Vt′(NP′), give the *matrix* and *store* that would be associated with each of the *labelled* nodes in the following tree, where computation of matrixes and stores proceeds by means of a simple version of *Cooper storage*. (Simplify by lambda-conversion as much as possible.)



Once you have computed the matrix and store for node (E), pull the NP translations out of the store and apply them successively to the matrix of the S to give **one** of the fully scoped translations of the S. Simplify by lambda-conversion as much as possible. (There is no need to repeat this to obtain the other reading.)

(iii) [6 marks] We now consider one further extension to the semantics.

A computational linguist has to decide what logic translations (and, hence, model-theoretic denotations) to assign to a number of sentences. She is comparing two options, illustrated below:

Sentence	Option 1	Option 2
"Mars attacks Neptune"	$attack1(m, n)$	$\exists e\, attack1(e, m, n)$
"Mars attacks Neptune on Tuesday"	$attack2(m, n, tue)$	$\exists e(attack1(e, m, n) \wedge day(e, tue))$
"Mars attacks Neptune on Tuesday at 3p.m."	$attack3(m, n, tue, 3p.m.)$	$\exists e(attack1(e, m, n) \wedge day(e, tue)$ $\wedge time(e, 3p.m.))$

(a) [3 marks] Option (2) is said to use *event semantics*. Explain what is meant by this term.

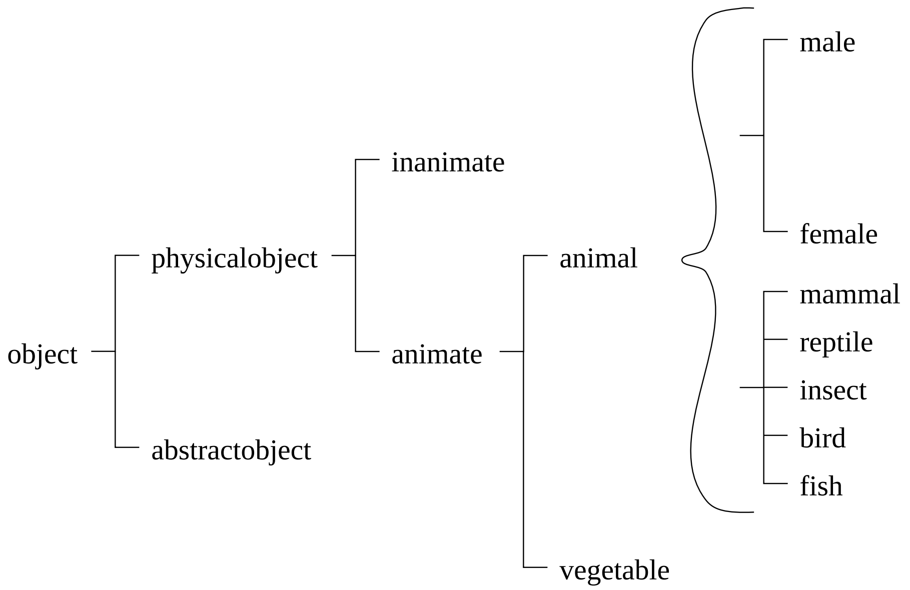
(b) [3 marks] Describe, **in detail**, **two** of the advantages of option (2) over option (1).

6 (25 marks)

(i) [5 marks] Say what is meant by a *generate-and-test* approach to disambiguation. List **three** of the problems of this approach.

(ii) [4 marks] What is a *sortal restriction* (also called a *selection restriction*)? Give examples of how such restrictions are used in natural language processing systems.

(iii) [9 marks] A Prolog natural language processing system is to make use of the *sortal hierarchy* that is depicted below.



In the diagram, square brackets denote *exclusive* choice and curly braces denote *non-exclusive* choice.

Explain in detail how a Prolog implementation would enforce sortal restrictions based on this sortal hierarchy. Your explanation need not include Prolog programs, but it should describe and illustrate how an entity's sort is represented by a Prolog term, how the expected sorts of the arguments of a predicate are represented using a list of terms, and how unification is used to test for restriction violation in a Prolog representation of a well-formed formula of predicate logic.

(iv) [4 marks] Enforcement of sortal restrictions using the method you described in (iii) is fairly efficient. But there are problems.

What problems does the method have, and what problems does sortal restriction enforcement have in general? (Full marks can be obtained by giving a total of *four distinct* problems of either kind.)

(v) [3 marks] Give, with examples, **two** other distinct sources of constraints that can be used in disambiguation in a natural language interface to a database.