4190334

UNIVERSITY OF YORK

BA Degree Examinations 2006-2007

DEPARTMENT OF LANGUAGE & LINGUISTIC SCIENCE

L334: Computational Syntax and Semantics

Time allowed: ONE hour

Answer ALL questions

(1) For each of the following

(i) $\alpha \Rightarrow \beta$

(ii) $A \Rightarrow \alpha$

(a) explain what the expression means, and (4 marks)

(b) state what kind of language it defines. (2 marks)

Answer

(i) $\alpha \Rightarrow \beta$: (a) a (possibly empty) string of terminals and non-terminals expands as a (possibly empty) string of terminals and non-terminals. (b) A recursively enumerable language (Chomsky Type 0)

(ii) $A \Rightarrow \alpha$: (a) a single non-terminal rewrites as a a (possibly empty) string of terminals and nonterminals. (b) A context free language (Chomsky type 2).

(2) What is the meaning of the following expressions? (3 marks each)

(i) $G = \langle T, N, S, P \rangle$.

Answer:

This defines a grammar (G) as an ordered quadruple consisting of a set of terminal symbols (T), a start symbol (S), drawn from the non-terminals and a set of productions (P), consisting of ordered pairs of non-terminals and strings of terminals and non-terminals ($P =\subset N \times (N \cup T)$).

(ii) $\mathcal{L}(G) = \{w \in T^* \mid S \stackrel{*}{\Rightarrow} w\}$.

Answer:

The language (L) defined by a grammar (G) is the set of strings of terminals (w) for which there exists a derivation from the start symbol (S).

(3) (a) What kind of language is $a^n b^n$? (2 marks)

Answer:

A Context-free language (Chomsky Type 2)

(b) What kind of language is $a^n b^n c^n$? (2 marks)

Answer:

A Context-sensitive language (Chomsky Type 1)

(4) Given the following derivation:

S	⇒	NP VP	(1)
	⇒	NP V NP	(2)
	⇒	NP V *scotch*	(3)
	⇒	NP *drinks scotch*	(4)
	⇒	PName *drinks scotch*	(5)
	⇒	*Toby drinks scotch*	(6)

(a) Provide the grammar that was used to produce it. (2 marks)

Answer

S	⇒NP VP	Scotch	:	NP
VP	⇒V NP	drinks	:	V
NP	⇒Pname	Toby	:	Pname

(b) State what kind of derivation it is. (2 marks)

Answer:

A rightmost (top-down) derivation

(c) If the rule $VP \Rightarrow VS$ was added to the grammar (with an appropriate lexical entry for V), what problems would a parser using this algorithm encounter? (5 marks)

Answer:

This rule would make the grammar right-recursive; in the worst case, with a right-to-left, top-down derivation such as the one above, this would lead to infinite search and non-termination.

(d) Would this problem be alleviated by changing to a left-to-right top-down parser? Explain. (5 marks)

Answer:

Yes; a left-to-right parser would not encounter infinite search with a right-recursive grammar.

(5) The worst case recognition time for Context-free Phrase Structure Languages is $\mathcal{O}(n^3)$; the worst case parsing time for Context-free Phrase Structure Languages is exponential.

(i) State briefly what is meant by $\mathcal{O}(n^3)$. (2 marks)

Answer:

This is an example of “Big O” notation; it means that the time required to recognise or parse a string n words long is proportional to the cube of the length of the string.

(ii) What is responsible for this difference in complexity? (6 marks)

Answer:

Recognition simply means identifying that there is a derivation from the start symbol of the grammar to the string, i.e. that there is a parse of the string; *parsing* requires that *all* parses of the string are found – in the worst case, this may lead to infinite search and non-termination.

(6) (a) Briefly describe the steps taken by a left-to-right shift-reduce parser parsing the sentence *"Toby drinks scotch"* using the grammar you provided in your answer to question (4) above. (5 marks)

Answer:

A shift-reduce parser uses two data structures: a *buffer* containing the sequence of words to be parsed and a —emphstack consisting of a last-in first-out queue of grammatical categories. The parser uses two operations: *shift*, which moves a word from the front of the buffer onto the top of the stack, and *reduce*, which matches the right-hand side of a grammar rule against the categories on the top of the stack and replaces them with the category on the left-hand side of the rule. The parse succeeds when the buffer is empty and there is a single item on the stack. In the case of *"Toby drinks scotch"*, the parser would execute the following steps:

Operation	Stack	Buffer	Rule
start	[ ]	[Toby drinks Scotch]	
shift	[Toby]	[drinks Scotch]	
reduce	[$Pname$]	[drinks Scotch]	Toby:$Pname$
reduce	[NP]	[drinks Scotch]	$NP \Rightarrow Pname$
shift	[NP drinks]	[Scotch]	
reduce	[$NP\ V$]	[Scotch]	drinks:V
shift	[$NP\ V$ Scotch]	[ ]	
reduce	[$NP\ V\ NP$]	[ ]	Scotch:NP
reduce	[$NP\ VP$]	[ ]	$VP \Rightarrow VNP$
reduce	[S]	[ ]	$S \Rightarrow NPVP$

(b) Comment briefly on the advantages and disadvantages of bottom-up vs. top-down parsing. (5 marks)

Answer:

Top-down parsing is susceptible to non-termination in the face of left (or right) recursion; bottom-up parsing is not. Bottom-up parsing is susceptible to non-termination if the grammar contains epsilon productions. With a bottom-up parser, it is easy to parse any kind of grammatical sequence; with a top-down parser, the category to be parsed must be specified in advance.

(c) What steps can be taken to avoid the disadvantages? (4 marks)

Answer:

For both kinds of parser, it is possible to compile the grammar into a weakly equivalent grammar which does not contain the offending kind of recursion, or epsilon productions. There are grammar formalisms which do not employ epsilon productions at all, raising no problem for bottom-up parsers. A further possibility for top-down parsers is to impose a depth bound, to prevent infinite recursion (but this may mean that some legitimate parses are missed).

(7) The following rule is one that occurs in one of the Prolog grammars used in the course.

```
np(P^(P*Z), [stored(NPmx, Z) | Store0]-Store) --->
  [det(Detmx, Store0-Store1), nbar(Nbarmx, Store1-Store)],
  {beta_reduce(Detmx*Nbarmx, NPmx)}.
```

(a) Is this a Context-free Phrase Structure Rule? Explain your answer. (2 marks)

Answer:

No; the categories in the rule (`np, det, nbar`) have arguments; this makes it a Definite Clause Grammar. DCGs are capable of parsing Recursively Enumerable languages.

(b) Explain the role in parsing of the arguments to the predicates used in the rule. (6 marks)

Answer:

This rule forms part of a grammar which uses quantifier storage. Each rule takes two arguments, the first of which carries the non-quantificational semantic information for the construction being parsed and the second carries the quantificational information. In this case, the non-quantificational information is represented by the expression `P^(P*Z)`, which is the Prolog version of the lambda calculus translation of an NP ($\lambda P[P(z)]$).

The second argument is the store, which is defined as a difference list. The first item on the store in this rule (consisting of the functor `stored`, with two arguments is the result of applying the translation of the determiner daughter to the translation of the Nbar daughter. E.g., for 'some soldier' this would be $\lambda Q[\lambda R[\exists x[Q(x) \wedge R(x)]]](\lambda z[soldier(z)]) = \lambda R[\exists x[soldier(x) \wedge R(x)]]$; this (in its Prolog form) would be the value of the variable `NPmx`.

The second item stored is the variable (Z) introduced by the non-quantificational `NP` translation. This plays a role in quantifier retrieval by forming a λ-abstract over the non-quantificational translation of the sentence being parsed: `Quantifier`(λ z[`SentenceTranslation`]).

Total marks: 60

End of Exam