IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2004

EEE/ISE PART II: MEng, BEng and ACGI

LANGUAGE PROCESSORS

Friday, 4 June 2:00 pm

Time allowed: 2:00 hours

There are FOUR questions on this paper.

Q1 is compulsory.
Answer Q1 and any two of questions 2-4.

Q1 carries 40% of the marks. Questions 2 to 4 carry equal marks.

Corrected Copy

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible First Marker(s) : Y.K. Demiris, Y.K. Demiris
Second Marker(s) : G.A. Constantinides, G.A. Constantinides

QUESTION 1:

(a) Describe two advantages for including an intermediate code generation phase in a compiler. [2]

(b) Within Chomsky's hierarchy of grammars, describe the main difference between a type-1 and a type-2 grammar, and provide two example production rules (one for each type of grammar) that illustrate this difference. [3]

(c) Provide the regular expression for valid identifiers in PASCAL [3]

(d) Provide a deterministic finite state automaton (DFA) for recognizing valid strings derived from the regular expression you provided in question 1(c). Clearly mark the start and final states of the DFA. [3]

(e) Explain why the grammar below is not LL(1), and use left-factoring to transform it to its LL(1) equivalent.

A -> aAb
A -> aAc
A -> d [4]

(f) Provide an algorithm for partitioning three-address statement sequences into basic blocks [3]

(g) Provide the definition of L-attributed grammars [2]

QUESTION 2:

Construct the deterministic finite state automaton (DFA) for the regular expression *a(b|c)*a* by:

(a) constructing a non-deterministic finite automaton (NFA) using Thompson's algorithm. [8]

(b) constructing the minimal equivalent DFA using the subset construction algorithm, followed by a DFA minimization step (*if required*). Explain the intermediate steps you have taken. [12]

QUESTION 3:

(a) For the augmented grammar below, compute the canonical LR(0) collection of sets of items

G' -> G
G -> G – X | X
X -> X * F | F
F -> (G) | a

where {a, -, *, (,)} are terminals, and {G', G, X, F} are the non-terminals. [8]

(b) The computation of the collection of sets of LR(0) items

$$C = \{I_0, I_1, I_2, \ldots, I_n\}$$

is the first step in the construction of an SLR parsing table for an augmented grammar G'.
Provide the remaining steps of the algorithm.
[Hint: You will need to provide the rules for constructing parsing actions and goto transitions for each state *i* constructed from I_i] [12]

QUESTION 4:

(a) For the grammar below, calculate the FIRST and FOLLOW sets for all non-terminal symbols [where $ represents the input right end marker, {a, -, *, (,)} are terminals, and {G', G, T, T', F} are non-terminals]

G -> T G'
G' -> -TG' | ε
T -> FT'
T' -> *FT' | ε
F -> (G) | a [10]

(b) The algorithm for the construction of a predictive parsing table M for a given grammar is as follows:

For each production rule A -> α do:

- *For each terminal x in FIRST(α), add A -> α to M[A, x]*
- *If FIRST(α) contains ε, add A -> α to M[A, b] for each terminal b in FOLLOW(A)*
- *If FIRST(α) contains ε, and FOLLOW(A) contains $, add A -> α to M[A, $]*
- *Mark all undefined entries of M as "error".*

Use this algorithm to construct the parsing table for the grammar above. Format your parsing table as shown below. [10]

Non-terminal	Input symbol					
	a	-	*	(	)	$
G						
G'						
T						
T'						
F						

E2.15: Language Processors
Sample Model answers to exam questions 2004

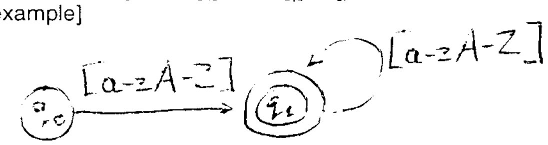
Question 1

(a) [bookwork] (1) Front-end/back-end separation enables easy porting to new architectures/languages (2) Machine-independent optimisations can be applied to the intermediate code representation.

(b) [bookwork] Main difference: type-2 grammars are context-free grammars (type-1 are context sensitive); Type1-example: AB->xyz, Type-2 example: A->xyz

(c) [new computed example] id: [a-zA-Z]([a-zA-Z]|[0-9])*

(d) [new computed example]



(e) [bookwork/new computed example] Its not LL(1) since FIRST(A) contains "a" in both the first and second rules; applying left-factoring will give you

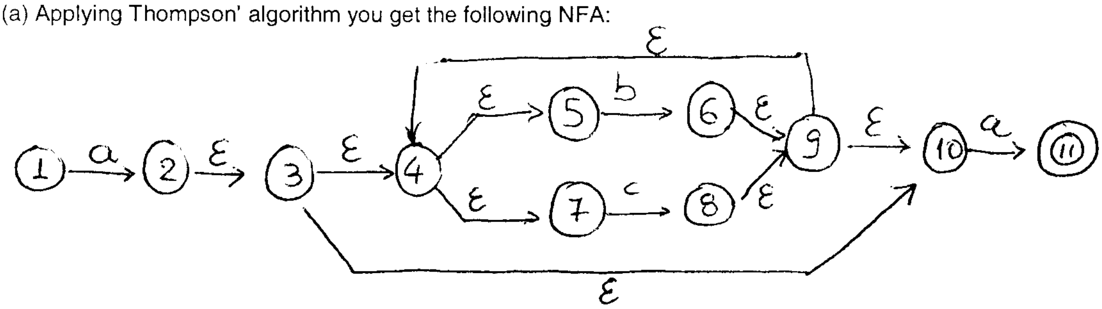
A -> aAX
X -> b | c
A -> d

(f) [bookwork]: First we need to determine the set of leaders, i.e. the first statements of basic blocks; we use the following two rules for that: (1) The first statement in the sequence is a leader (2) any statement that is the target of a conditional or unconditional goto is a leader (3) any statement that immediately follows a goto or conditional goto statement is a leader. Once the leaders are determined, for each leader, its basic block consists of the leader, and all the statements up to but not including the next leader.

(g) [bookwork]: L-attributed grammars are grammars where each inherited attribute of X_j ($1 <= j <= n$) in a production rue of the form A -> $X_1X_2 \ldots X_n$ depends only on the attributes of the symbols X_1, X_2, ..., X_{j-1} to the left of X_j in the production, and the inherited attributes of A.

Question 2
[new computed example]

(a) Applying Thompson' algorithm you get the following NFA:



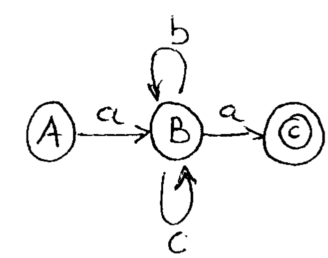
(b) Applying the subset construction algorithm on this NFA we get:

ε-closure(Start-State) = {1} = A
ε-closure(move(A, a)) = {2,3,4,5,6,7,8,9,10} = B
ε-closure(move(A, b)) = ε-closure(move(A, b)) = {}

ε-closure(move(B, a)) = {2,3,4,5,6,7,8,9,10, 11} = C
ε-closure(move(B, b)) = {2,3,4,5,6,7,8,9,10} = B
ε-closure(move(B, c)) = {2,3,4,5,6,7,8,9,10} = B

ε-closure(move(C, a)) = ε-closure(move(C, b)) = ε-closure(move(C, c)) = {}
– no more new created states; we are done.

DFA:



Question 3:

(a) [new computed example]

I_0:

G' -> .G
G -> .G –X
G -> .X
X -> .X*F
X -> .F
F -> .(G)
F -> .a

I_1:

G' -> G.
G -> G.-X

I_2:

G ->X.
X -> X.*F

I_3:

X->F.

I_4:

F -> (.G)
G -> .G-X
G -> .X
X -> .X*F
X -> .F
F -> .(G)
F -> .a

I_5:

F -> a.

I_6:

G -> G-.X
X-> .X*F
X->.F
F->.(G)
F->.a

I_7:

X-> X*.F
F->.(G)
F->.a

I_8:

F-> (G.)
G->G.-X

I_9:

G-> G-T.
X -> X.*F

I_{10}:

X->X*F.

I_{11}:

F->(G).

(b) [Bookwork]

The remaining steps after the construction of the collection of sets of LR(0) items are:

- Construct the parsing entries for state i are determined as follows:
  - For a terminal a, if [A->α.aB] is in I_i and goto(I_i,a) = I_j then set action[i,a] to "shift j"
  - If [A->α.] is in I_i, then set action[i,a] to "reduce A->α for all a in FOLLOW(A)
  - If [S'->S.] is in I_i then set action[i,$] to "accept"
- The goto transitions for state i are constructed for all nonterminals A using the rule: if goto(I_i, A)= I_j then goto[i,A]=j
- All entries not defined by the two rules above, are made "error".

Question 4:

(a) FIRST(G)=FIRST(T) = FIRST(F) = {(,a}; FIRST(G') = {-, ε}; FIRST(T') = {*,ε}

FOLLOW(G) = FOLLOW{G') = {), $}; FOLLOW(T) = FOLLOW(T') = {-,), $};
FOLLOW(F) = {+, *,), $}

(b) The parsing table for the grammar is:

	Input symbol					
	a	-	*	(	)	$
G	G→TG'			G→TG'		
G'		G'→-TG'			G'→ε	G'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→a			F→(G)		