E2.15

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2006

EEE/ISE PART II: MEng, BEng and ACGI

Corrected Copy

LANGUAGE PROCESSORS

Monday, 5 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 (30% each).

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

Examiners responsible	First Marker(s) :	Y.K. Demiris,
	Second Marker(s) :	J.V. Pitt,

1. [COMPULSORY]

(a) Chomsky's hierarchy of grammars defines four types of grammars; describe each of the four types discussing the restrictions that each type imposes on its grammar production rules. [3]

(b) Provide the formal definition of a Turing Machine. Explain the differences between a Linearly-bounded Automaton (LBA) and a Turing Machine. Describe the benefits of an LBA over a Turing Machine. [8]

(c) Given the following context-sensitive grammar, derive the string "aaabbbccc":

S -> aSBC | aBC
CB -> BC
aB -> ab
bB -> bb
bC -> bc
cC -> cc [4]

(d) Describe the formalism known as Syntax-directed Definition; provide an example syntax-directed definition for converting arithmetic expressions (consisting of single digits and the arithmetic operators "+" and "-") from infix to postfix notation. [8]

(e) Describe the formalism known as "translation scheme"; provide an example translation scheme for converting arithmetic expressions (consisting of single digits and the arithmetic operators "+" and "-") from infix to postfix notation. [8]

(f) Describe the steps involved in the DFA minimization algorithm. [5]

(g) Describe the data structures required for the LR parsing algorithm, and provide the steps performed with them during the execution of the algorithm. [4]

2. You are required to construct the minimal deterministic finite state automaton (DFA) for the regular expression *ab(c|d)*a* following the steps below.

(a) Construct a non-deterministic finite automaton (NFA) using Thompson's algorithm. [12]

(b) Construct the equivalent DFA using the subset construction algorithm. *Explain the intermediate steps you have taken.* [12]

(c) Apply the DFA minimization algorithm to the DFA you have constructed in (b), and showing whether your DFA was already minimal or not. *Explain the intermediate steps of the application of the DFA minimization algorithm* [6]

3. (a) Calculate the FIRST and FOLLOW sets for all non-terminal symbols for the grammar below [with {a, -, *, (,)} being terminals, {G', G, T, T', F} being non-terminals, ε being the empty string, and $ being the input right end marker]

(1)	G -> T G'
(2)	G' -> - T G'
(3)	G' -> ε
(4)	T -> F T'
(5)	T' -> * F T'
(6)	T' -> ε
(7)	F -> (G)
(8)	F -> a

[10]

(b) FIRST and FOLLOW sets are used in top-down parsing to assist in the construction of parsing tables; provide the algorithm for constructing a top-down parsing table, and construct the table for the grammar above. [20]

4. (a) In the context of shift-reduce parsing, provide the definition of the concepts "viable prefix" and "LR(0) items" and provide the algorithm for computing the canonical LR(0) collection of sets of items. [6]

(b) For the grammar below, construct the canonical set of LR(0) items and provide the DFA that can recognize viable prefixes for the grammar:

E' -> E
E -> E + T | T
T -> T * F | F
F -> (E) | id

[24]

E2.15: Language Processors
Sample Model answers to exam questions 2006

Question 1

(a) [bookwork] Type 0 (unrestricted grammars), 1 (for all productions α->β, we must have |α| ≤ |β|), 2 or context free grammars (only a single non-terminal may appear on the left-side of a production), and 3 or regular grammars (productions should all be left-linear or right linear)

(b) [bookwork] A Turing Machine M is defined as (Q, Σ, Γ, δ,q_0, B, F) where

- Q: a finite set of N states q_0, q_1, …, q_N of the finite control
- Σ: a finite input alphabet of symbols
- Γ: the complete set of tape symbols, Σ ⊆ Γ
- δ(q,X): the transition function between states. Given a state q ∈Q, and a tape symbol X ∈ Γ, the function δ(q,X) returns a triplet (p,Y, D), where p is the new state in Q, Y ∈ Γ is the symbol that replaces the current symbol on the tape, and D is either {left,right} indicating where will the head move next
- q_0: the start state
- B is the blank symbol (B∈Γ but B∉Σ) initially written in all tape cells except the ones holding the input
- F: the set of final states, F⊆ Q

[NB: the students need not use the symbols used above – any will do; however they are expected to be exact as to what constitutes a Turing Machine, and incorporate all of the elements above].

LBA are a restricted class of TMs wrt the length of the tape, which in LBAs is not infinite, but proportional to the length of the input string. LBAs are deterministic, and are guaranteed to give a decision as to whether the input string is legal or not in an amount of time proportional to the length of the input string.

(c) [New computed example]
S
aSBC (using rule 1a)
aaSBCBC (using rule 1a)
aaaBCBCBC (using rule 1b)
aaaBBCCBC (using rule 2)
aaaBBCBCC (using rule 2)
aaaBBBCCC (using rule 2)
aaabBBCCC (using rule 3)
aaabbBCCC (using rule 4)
aaabbbCCC (using rule 4)
aaabbbcCC (using rule 5)
aaabbbccC (using rule 5)
aaabbbccc (using rule 5)

(d) [bookwork]

Grammar symbols can have an associated set of attributes
Each production rule can have an associated set of *semantic rules, which* are used to compute values of the attributes associated with the symbols appearing in that production
The grammar and the set of semantic rules constitute a formalism known as *syntax-directed definition.*

Syntax directed definition for infix to postfix translation:

Production	Semantic rule
Expr -> Expr1 + term	Expr.t = expr1.t || term.t || '+'
Expr -> Expr1 - term	Expr.t = expr1.t || term.t || '-'
Expr -> term	Expr.t = term.t
Term -> 0	Term.t = '0'
… for the rest of digits until:	… for the rest of digit until:
Term -> 9	Term.t = '9'

(e) [bookwork] A *translation scheme* is a CF grammar in which program fragments are embedded in right side of the productions: *the semantic actions.* These are added in the parse tree and are executed when they are accessed during the tree traversal.

Translation scheme for the infix to postfix translation:

```
 Expr -> expr + term {printf('+')}
Expr -> expr – term {printf('-')}
Expr -> term
Term-> 0 {printf('0')}
..
Term-> 9 {printf('9')}
```

(f) [bookwork] The DFA minimization algorithm:

Start by assuming that all the states in the DFA are equivalent
Work through the states, putting different states in separate sets
Two states are considered different if:

- One is a final state and the other one isn't
- The transition function maps them to different states, based on the same input character.

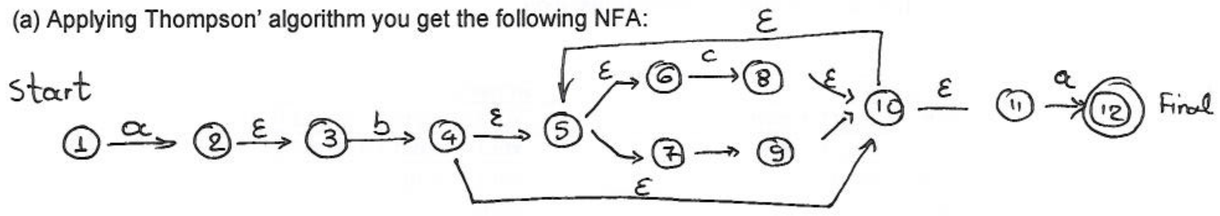
1. The minimization algorithm starts by initially creating two sets of states, final and non-final.
2. For each state-set created from (1), it examines the transitions for each state and for each input symbol. It the transition is to a different state for any two states, then they are put into different state-sets.
3. Repeat until no new state sets are being created by 2.

(g) LR parsing involves the use of a parsing table (containing *goto* and *action* entries), and a stack. Given an input string w, the algorithm proceeds as follows:

```
Set input pointer ip to the first symbol of w$
Repeat:
   Let s be the state on top of the stack, and a the symbol pointed by ip
   if action[s,a] = shift s'   then
       begin
           Push a then s' on top of the stack
           Advance ip to the next input symbol
       end
   else if action[s,a] = reduce A->β then begin
         Pop 2*length(β) items off the stack
         Let s' be the state now on top of the stack
         Push A then goto[s', A] on top of the stack
         Output the production A->β
         end
   else if action[s,a]=accept then return
   else error()
   end
```

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(StartState) = {1} = A
ε-closure(move(A, a)) = {2,3} = B
ε-closure(move(A, b)) = ε-closure(move(A, c)) = ε-closure(move(A, d)) = {}

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

ε-closure(move(C, a) = {12} = D

ε-closure(move(C, b)) = {}
ε-closure(move(C, c)) = ε-closure(move(C, d)) = {4,5,6,7,8,9,10,11} = C
ε-closure(move(D, a)) = ε-closure(move(D, b)) = ε-closure(move(D, c)) = ε-closure(move(D, d)) = {}
– no more new created states; we are done.

Resulting DFA:

$$\begin{tikzcd}
\boxed{A} \arrow[r, "a"] & \boxed{B} \arrow[r, "b"] & \boxed{C} \arrow[loop above, "c"] \arrow[loop below, "d"] \arrow[r, "a"] & \boxed{\boxed{D}}
\end{tikzcd}$$

(c) The DFA we have derived is already minimal, and thus applying the DFA minimization algorithm is trivial: we start by assuming that all states of the DFA are equal, and we work through the states, putting different states in separate sets if (a) one is final and other is not (our case here) (b) the transition function maps them to different states, based on the same input character. The DFA minimization algorithm proceeds by initially creating two sets of states, final and non-final - non-final: {A, B, C} and final: {D}. For each state set created, the algorithm examines the transitions for each state and for each input symbol. In our case, all state sets contain only one state, so the algorithm terminates here, and we have (already) the minimal DFA.

Question 3:

(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 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".*

The resulting table for the grammar above 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)		

QUESTION 4:

(a) A *viable prefix* is a prefix of a right sentential form which can appear as the stack contents during a shift-reduce parse. An *LR(0) item* (or simply *item*) of a grammar is a production rule augmented with a position marker (a dot) somewhere within its right hand side.
The algorithm for computing the canonical LR(0) collection (lets say C) of sets of items :

```
We start with C = {closure({[S' -> .S]})};
Repeat
For each set of items I in C and each grammar symbol X such that Goto(I,X) is not empty and not in C
Do: add goto(I, X) to C
Until no more sets of items can be added to C
End
```

(b)

The canonical set of LR(0) items

I_0:	E' -> .E E- > .E+T E -> .T T -> .T*F T -> .F F -> .(E) F -> .id
I_1:	E' -> E. E -> E.+T
I_2:	E -> T. T -> T.*F
I_3:	T -> F.
I_4:	F -> (.E) E -> .E+T E -> .T T -> .T*F T -> .F F -> .(E) F -> .id
I_5:	F -> id.
I_6:	E -> E+.T T -> .T*F T -> .F F -> .(E) F -> .id
I_7:	T -> T*.F F -> .(E) F -> .id
I_8:	F -> (E.) E -> E.+T
I_9:	E -> E+T. T -> T.*F
I_{10}:	T -> T * F.
I_{11}:	F -> (E).

The DFA for recognizing viable prefixes for the grammar:

$$\begin{tikzcd}
I_0 \arrow[r, "E"] \arrow[ddr, "T"] \arrow[ddddr, "F"] \arrow[dddddr, "("] \arrow[ddddddr, "id"] & I_1 \arrow[r, "+"] & I_6 \arrow[r, "T"] \arrow[d, "id"] \arrow[dr, "F"] \arrow[drr, "("] & I_9 \arrow[r, "*"] & \text{to } I_7 \\
 & & \text{to } I_5 & \text{to } I_3 & \text{to } I_4 \\
 & I_2 \arrow[r, "*"] & I_7 \arrow[r, "F"] \arrow[d, "id"] \arrow[dr, "("] & I_{10} & \\
 & & \text{to } I_5 & \text{to } I_4 & \\
 & I_3 & & & \\
 & I_4 \arrow[loop above, "("] \arrow[r, "E"] \arrow[d, "id"] \arrow[dr, "F"] \arrow[drr, "T"] & I_8 \arrow[r, ")"] \arrow[drr, "+"] & I_{11} & \\
 & I_5 & \text{to } I_3 & \text{to } I_2 & \text{to } I_6
\end{tikzcd}$$

The set of items we computed gives rise to a DFA recognizing viable prefixes