IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2007

ISE PART II: MEng, BEng and ACGI

Corrected Copy

LANGUAGE PROCESSORS

Wednesday, 6 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, Y.K. Demiris
Second Marker(s) : J.V. Pitt, J.V. Pitt

The Questions

1. [COMPULSORY]

(a) Provide the transition diagram for a push-down automaton that can be used to recognise the language $\{a^n b^n, n>0\}$. Give an example of a language that cannot be recognised by a push-down automaton. [6]

(b) Provide the formal definition of a Push-Down Automaton (PDA) and describe the differences between a Push Down automaton and a Linearly-Bounded Automaton (LBA). [8]

(c) Describe the data structures that are involved in the LR parsing algorithm and provide a description of the algorithm's operation. [8]

(d) Describe the algorithm for register allocation via graph colouring, including a heuristic algorithm for determining whether a graph *G* is colourable using a number of colours, *K*. [8]

(e) A programming language imposes the following restrictions:

- Identifiers must start with a letter, followed by zero or more letters or digits. Examples include *Var1, HelloWorld, x121,* among others.
- Numbers must be written either in decimal or scientific notation; the format consists of the following two parts:
  - [A mandatory part] one or more digits
  - [An optional part] a decimal point, followed by one or more digits, optionally followed by "E", an optional plus or minus sign, and one or more digits.

  Examples include *12, 1.234, 12.3E4, 1.2E-34,* among others.

I. Provide the regular expressions for valid identifiers and numbers for this language; define all special characters you used. [4]

II. Provide a finite state automaton that, given an input string, will recognize it as either a valid identifier, or as a valid number. Clearly mark all final (accepting) states for the FSA. [6]

2. You are required to construct the minimal deterministic finite state automaton (DFA) for the regular expression *a(b|c)*d* 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 show whether your DFA was already minimal or not. *Explain the intermediate steps of the application of the DFA minimization algorithm.* [6]

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

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

where {a, -, *, (,)} are terminals, and {V', V, X, F} are the non-terminals. [12]

(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] [18]

4. (a) Calculate the FIRST and FOLLOW sets for all non-terminal symbols for the grammar below where {a, b, c, d, e } are terminals, and {A, B, C, D, E} are non-terminals:

(1) A -> C B
(2) B -> b C B | ε
(4) C -> E D
(5) D -> d E D | ε
(7) E -> e A c | a [15]

(b) You are given the following algorithm for the construction of a predictive parsing table M for a grammar:

For each production rule of the form 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. You should format your parsing table as shown below, where $ denotes the end of input marker. [15]

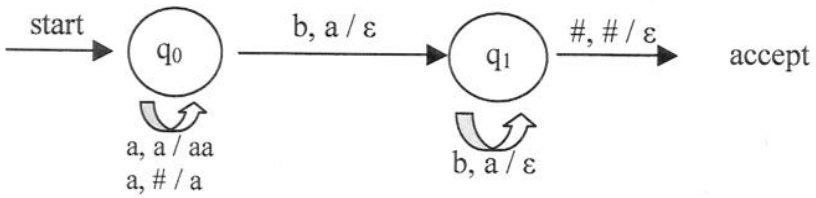
Non-terminal	Input Symbol					
	a	b	c	d	e	$
A						
B						
C						
D						
E						

E2.15: Language Processors
Sample Model answers to exam questions 2007

Question 1

(a) [New Computed Example]: The required PDA is the following:

(labels on the arrows: input symbol, stack symbol popped / symbols pushed, ε denotes empty string,)



An example of a language that cannot be recognised using a PDA is $\{a^n b^n c^n, n>=1\}$

(b) [Bookwork]: A PDA P is defined as $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

- Q: a finite set of N states $q_0, q_1, \ldots, q_N$
- Σ: a finite input alphabet of symbols
- Γ: a finite stack alphabet – the set of symbols
- $\delta(q,\alpha,X)$: the transition function between states. Given a state $q \in Q$, $\alpha \in \Sigma$ or $\alpha=\varepsilon$, and a stack symbol $X \in \Gamma$, the function $\delta(q,\alpha, X)$ returns a pair (p,γ), where p is the new state and γ is the string of stack symbols that replaces X at the top of the stack
- q_0: the start state
- Z_0: the start stack symbol
- F: the set of final states, $F \subseteq Q$

An LBA is a restricted class of Turing machines, and in contrast to the PDA contains a tape, and a bidirectional read-write head. It can be used to recognize context sensitive grammars in contrast to the PDA.

(c) [Bookwork}: 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
```

(d) [Bookwork]:

By constructing the *interference graph*, where :

1. Variables are nodes in the graph
2. An arc drawn between two nodes indicates that the two nodes cannot share a register, because they are live at the same time.

we map the problem of register allocation to the graph colouring problem in graph theory: how to colour the nodes of a graph with the lowest possible number of colours, such that for each arc the nodes at its ends have different colours.

Heuristic algorithm for determining whether a graph is K-colourable:

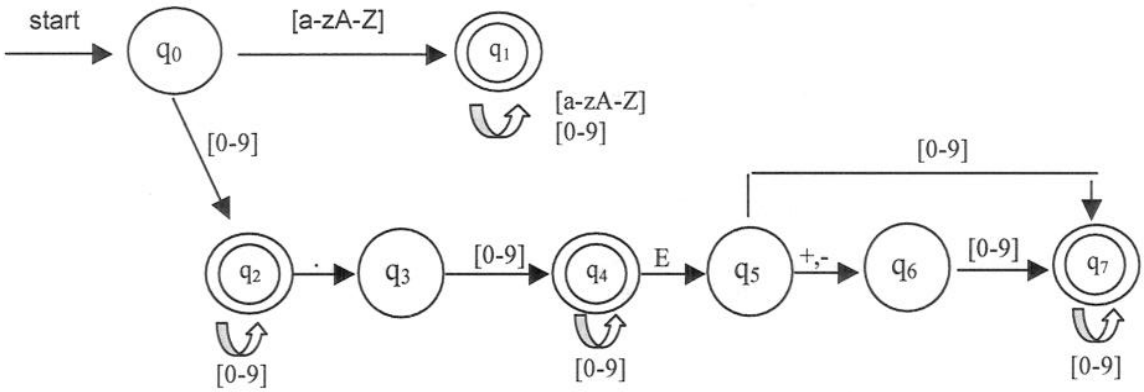
For each node n in the graph G that has fewer than k-neighbours, we remove n along with its edges. This results in a graph G' and the problem has been reduced to k-colouring of G' (since G can be coloured by assigning to n one of the colours note assigned to any of its neighbours).
Process is repeated until you get either:

- An empty graph (which means that k-colouring of G is possible)
- A graph in which each node has k or more adjacent nodes (which means that k-colouring may not be possible, and spilling code may be needed).

(e) [New computed example]:

(a) identifier: [a-zA-Z]([a-zA-Z]|[0-9])*
number: [0-9]+(.[0-9]+)?(E[+-]?[0-9]+)?
[Defining [a-zA-Z] as *letter* and [0-9] as *digit*, and using those in the regular expressions above is fine.]

(b) [accepting states are marked with double circle]

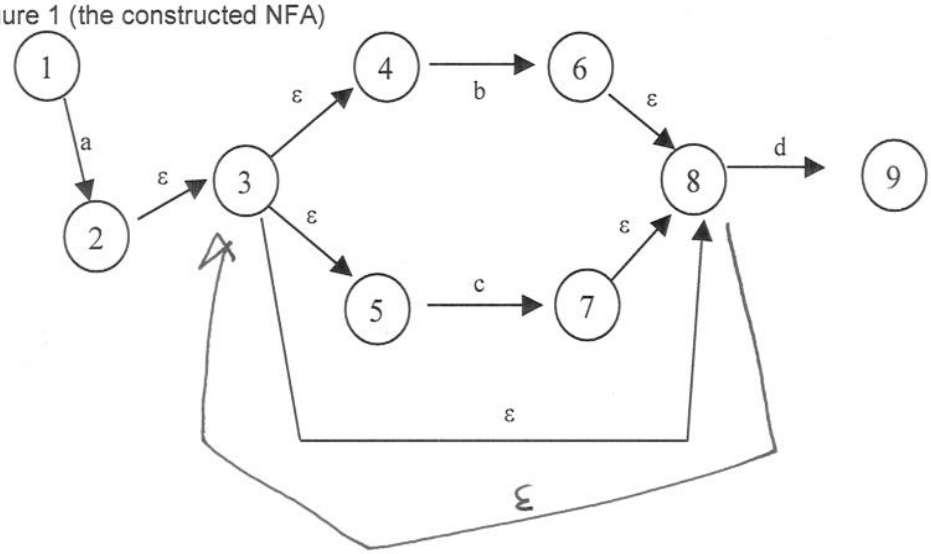


Question 2
[new computed example]

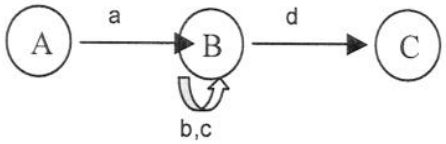
(a) Applying Thompson' algorithm you get the NFA of figure 1.
(b) Applying the subset construction algorithm on this NFA we get:

ε-closure(StartState) = {1} = A
ε-closure(move(A, a)) = {2,3,4,5,6,7,8} = 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, b) = ε-closure(move(B, C) = {2,3, 4,5,6,7,8} = B
ε-closure(move(B, d) = {9} = C
ε-closure(move(C, a)) = ε-closure(move(C, b)) = ε-closure(move(C, c)) = ε-closure(move(C, d)) = {}
– no more new created states; we are done.

Figure 1 (the constructed NFA)



The resulting DFA;



(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} and final: {C}. 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) [New Computed Example]

I_0:	I_1:	I_2:	I_3:	I_4:	I_5:
V' -> .V	V' -> V.	V ->X.	X->F.	F -> (.V)	F -> a.
V -> .V –X	V -> V.-X	X -> X.*F		V -> .V-X	
V -> .X				V -> .X	
X -> .X*F				X -> .X*F	
X -> .F				X -> .F	
F -> .(V)				F -> .(V)	
F -> .a				F -> .a	

I_6:	I_7:	I_8:	I_9:	I_{10}:	I_{11}:
V -> V-.X	X-> X*.F	F-> (V.)	V-> V-T.	X->X*F.	F->(V).
X-> .X*F	F->.(V)	V->V.-X	X -> X.*F		
X->.F	F->.a				
F->.(V)					
F->.a					

(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) [New Computed Example]:

FIRST(A) = FIRST(C) = FIRST(E) = {e,a}; FIRST(B) = {b, ε}; FIRST(D) = {d,ε}
FOLLOW(A) = FOLLOW(B) = {c, $}; FOLLOW(C) = FOLLOW(D) = {b, c, $};
FOLLOW(E) = {b, d, c, $}

(b) [New Computed Example]: The parsing table for the grammar is:

Non-terminal	Input Symbol					
	a	b	c	d	e	$
A	A -> c B				A -> c B	
B		B -> b C B	B -> ε			B -> ε
C	C -> E D				C -> E D	
D		D -> ε	D -> ε	D -> d E D		D -> ε
E	E -> a				E-> e A c	