IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2011

ISE PART II: MEng, BEng and ACGI

Corrected Copy G 3

LANGUAGE PROCESSORS

Friday, 3 June 2:00 pm

Time allowed: 2:00 hours

There are THREE questions on this paper.

Answer ALL questions.
Q1 carries 40% of the marks. Questions 2 and 3 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

The Questions

Answer ALL questions

1.

(a) Provide the formal definition of a Push-Down Automaton (PDA). [4]

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

(c) Describe the formalism known as *Syntax-Directed Definition*, and provide an example syntax-directed definition for conversing arithmetic expressions (consisting of single digits and the arithmetic operators "*" and "+") from infix to postfix notation. [6]

(d) In the context of lexical analysis, once a minimal Deterministic Finite Automaton (DFA) has been constructed (following a conversion from Non-deterministic Finite Automaton (NFA) -> DFA -> Minimal DFA), a lexical analyser can use it to process input strings to check their lexical validity. Provide pseudo-code for simulating the execution of the DFA. [4]

(e) In the context of shift-reduce parsing, and in order to construct the parsing table, the functions closure(I) and goto(I, X) need to be defined (where I is a set of items and X is a grammar symbol). Provide the definition of these functions. [4]

(f) Using an example string, explain why the grammar

X -> X + X
X -> a

is ambiguous, rewrite it to remove its ambiguity, and explain why the resulting grammar is unambiguous. [6]

(g) 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 K B
(2) B -> b C B | ε
(3) C -> E D
(4) D -> d E D | ε
(5) E -> e A c | a [10]

2. You are required to construct the minimal deterministic finite state automaton (DFA) for the regular expression *a*b*c* following the steps below. You should clearly mark all final states in all the automata you construct.

(a) Construct a non-deterministic finite automaton (NFA) using Thompson's algorithm. [10]

(b) Construct the equivalent DFA using the subset construction algorithm. *Explain the intermediate steps you have taken.* [10]

(c) Describe the DFA minimization algorithm, and subsequently apply it to the DFA you have constructed in (b). Show whether your DFA was already minimal or not. *Explain clearly the intermediate steps of the application of the DFA minimization algorithm* [10]

3.

(a) For the augmented grammar below,

0. E' -> E
1. E -> E + T
2. E -> T
3. T -> T * F
4. T -> F
5. F -> (E)
6. F -> id

compute the canonical set of LR(0) items [Hint: set contains 12 items] [10]

(b) Each set I_i from the ones computed above constructs a state i. The parsing entries for state i are determined by the following three rules:

(Rule 1) 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
(Rule 2) If [A -> α.] is in I_i, then set action[i,a] to "reduce A -> α for all a in FOLLOW(A) – here A may not be S'
(Rule 3) If [S'->S.] is in I_i, then set action[i,$] to "accept".

The goto transitions for state i are constructed for all nonterminals using the rule:
(Rule 4) if goto(I_i, A) = I_j, then goto[i,A]=j

All entries not defined by the rules above are made "error", indicated by blank entries.

Using the canonical set of LR(0) computed in (3a), and the table construction algorithm above, construct the parsing table for this grammar using the table format below (three entries already completed) [20]

State	Action entries						Goto entries		
	id	+	*	(	)	$	E	T	F
0									
1									
2									
3									
4	S5								
5	S5								
6								9	
7									
8									
9									
10									
11						r5			

(NB: Make sure you copy this table to your exam booklet!)

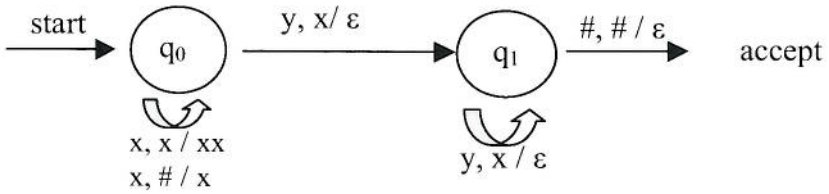
E2.15: Language Processors
Sample answers to exam questions 2011

Question 1

(a) [Bookwork]:
A PDA P is defined as (Q, Σ, Γ, δ,q_0, Z_0, F)
- Q: a finite set of N states q_0, q_1, …, q_N
- Σ: a finite input alphabet of symbols
- Γ: a finite stack alphabet – the set of symbols
- δ(q,α,X): the transition function between states. Given a state q ∈Q, α ∈ Σ or α=ε, and a stack symbol X ∈ Γ, the function δ(q,α, 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⊆ Q

(b) 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 $\{x^n y^n z^n, n>=1\}$

(c) [Bookwork + new computed example]

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'

(d) [Bookwork]

The state transition table defines function move(s,ch) which returns the next state from s given ch as input character. Then:

```
S:=S0
c:=nextchar();
while c <> eof do
    s:= move (s, c)
    c:= nextchar();
    end;
if s is in FinalStates then
   return "valid";
else return "not valid";
```

(e) [Bookwork]

The *closure(I)* of a set of items I for a grammar G is the set of items constructed from I using two rules:
1. Initially, every item in I is added in closure(I)
2. If A->α.Bβ in closure(I) and B->γ is a production, then add item B->.γ to I if its not already there. We apply this rule until no more new items can be added to closure(I)

For a set of items I and a grammar symbol X, the function goto (I, X) is defined as the closure of the set of all items [A -> αX.β] such that [A -> α.Xβ] is in I

(f) The grammar is ambiguous because there are sentences that do not have a unique leftmost and rightmost derivation, and do not result into a unique parse tree. For example, x+x+x has two different leftmost and rightmost derivations (for example, S+S -> x+ S -> x + S + S -> x+x+S -> x+x+x, and S+S -> S + S + S -> x + S + S -> x + x + S -> x + x + x are two different possible leftmost derivations resulting into different parse trees). Rewriting it as

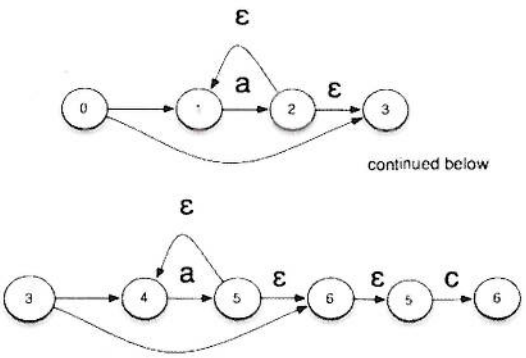
(1) S -> S + x
(2) S- > x

results into a grammar that is non-ambiguous, and has a unique leftmost and rightmost derivation (e.g. S + x -> S + x + x -> x + x + x is the unique leftmost derivation, similarly for the rightmost)

(g) [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, $}

Question 2 [new computed example]

(a) Applying Thompson' algorithm you get



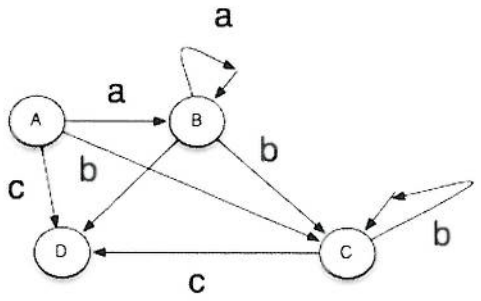
(b) Applying the subset construction algorithm on this NFA we get:

ε-closure(StartState) = ε-closure{0} = {0,1, 3, 4, 6, 7} = A
ε-closure(move(A, a)) = ε-closure{2} = {1, 2, 3, 4, 6, 7} = B
ε-closure(move(A, b)) = ε-closure{5} = {4, 6, 7} = C
ε-closure(move(A, c)) = ε-closure{8} = {8} = D

ε-closure(move(B, a)) = ε-closure{2} = {1, 2, 3, 4, 6, 7} = B
ε-closure(move(B, b)) = ε-closure{5} = {4, 6, 7} = C
ε-closure(move(B, c)) = ε-closure{8} = {8} = D

ε-closure(move(C, a)) = {}
ε-closure(move(C, b)) = ε-closure{5} = {4, 6, 7} = C
ε-closure(move(C, c)) = ε-closure{8} = {8} = D

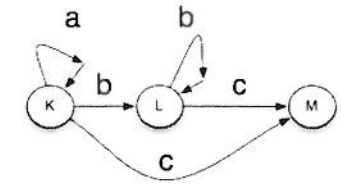
ε-closure(move(D, a)) = ε-closure(move(D, b)) = ε-closure(move(D, c)) = {}
No more new states, we are done.



(c) In order to minimise the DFA we obtained from the subset construction algorithm, 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 (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 the first set is further subdivided in two sets, one with {A, B}, and one with {C} since the two first ones go to the same states for the same symbols. If we name the first set {A, B} as K, second set {C}) as L, and M = {D}, and draw all transitions, we get the following minimal DFA:



Question 3: [New computed example]

(a) The set of 12 LR(0) items is:

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).

(b)

State	Action						Goto		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			