IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2005

EEE/ISE PART II: MEng, BEng and ACGI

LANGUAGE PROCESSORS

Friday, 3 June 2:00 pm

Corrected Copy

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) : R. Lent, R. Lent

1. [COMPULSORY]

(a) The language $\{a^n b^n, n \geq 1\}$ can be generated by the following grammar:

S -> ab
S -> aSb

Within the context of Chomsky's hierarchy of grammars, explain why this grammar is *not* a regular (type-3) grammar, describe what type of grammar it is, and why. [6]

(b) Provide the formal definition of a Push Down Automaton. [8]

(c) Explain the main two differences between Deterministic Finite Automata (DFA) and Non-deterministic Finite Automata (NFA) in terms of allowable transitions. [4]

(d) In the context of converting NFAs to DFAs, describe the subset construction algorithm by providing the definitions of the two functions required, as well as the steps involved in the algorithm. [6]

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

S -> S + S
S -> x

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

(f) In the context of parsing table construction, provide the two rules used for the computation of the function *closure(L)* of a set of items L for a grammar G. [6]

(g) Provide an example for each of the following code optimisation techniques: (i) Flow of control optimisation (ii) copy propagation. [4]

2. Construct the minimal deterministic finite state automaton (DFA) for the regular expression $k^*x(y|z)^*$ by:

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

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

(c) applying 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 {z, p, t, o, c} being terminals, and {X, K, Y, L, F} being non-terminals, and $ being the input right end marker]

(1) X -> Y K
(2) K -> p Y K
(3) K -> ε
(4) Y -> F L
(5) L -> t F L
(6) L -> ε
(7) F -> o X c
(8) F -> z [15]

(b) Consider the grammar

(1) X -> X + X
(2) X -> X * X
(3) X -> (X)
(4) X -> a

Given the valid input string (a+a*a), simulate *one* of the possible sequences of steps of the shift-reduce parser, showing at each step the remaining input string, the contents of the stack, the action performed, and if it is a reduce action, the production rule used. Draw the resulting parse tree.

(Note that since the grammar above has two rightmost derivations for the input (a+a*a) there are more than one sequences of steps that the parser can make) [15]

4. (a) In the context of register allocation, describe the use of interference graphs, and how graph-colouring theory can be used for the allocation of registers. [9]

(b) Assume the following code fragment:

```
prod = x * y;
sum1 = x + x;
sum2 = y + y;
total1 = prod + sum1 * sum2;
total2 = prod – sum1 * sum2;
```

is to be compiled on a machine that has 4 registers, {R0, R1, R2, R3}

Assume that:

- x and y are live on entry to this block, and dead upon exit
- total1 and total2 are live upon exit
- prod, sum1, and sum2 are dead upon exit

Then

i. draw the interference graph for the code segment above, explaining your decisions at each step. [14]

ii. explain why the interference graph you created in (i) is 4-colourable, and using the method of register allocation through graph colouring, allocate registers to the variables above. [7]

E2.15: Language Processors
Sample Model answers to exam questions 2005

Question 1

(a) [bookwork] The grammar is not regular (type 3) since the second production rule is neither left-linear or right-linear. It is context-free grammar (type-2), since only a single non-terminal appears on the left-side of all production rules.

(b) [bookwork]
A pushdown automaton P is defined as $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
where:
- Q is a finite set of N states $q_0, q_1, \ldots, q_N$
- Σ is a finite input alphabet of symbols
- Γ is a finite stack alphabet
- $\Delta(q, \alpha, X)$ is the transition function between states. Given a state q in Q, α in Σ, or $\alpha = \varepsilon$, and a stack symbol X in Γ , 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 is the start state
- Z_0 is the start stack symbol
- F is the set of final states (F subset of 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 PDA, and incorporate all of the elements above].

(c) [bookwork] DFAs allow only one transition from one state to another for a given input. NFAs allow more than one transition possibilities for a given input, and also allow ε-transitions (transition between states without reading any input)

(d) [bookwork] The subset construction algorithm requires the definition of two functions:
- the ε-closure function takes as input a state and returns the set of states reachable from it based on one or more ε-transitions. This set will always include the state itself.
- The function move(state, char) which take as input a state and a character and returns the set of states reachable from it, with one transition using this character.

The algorithm then proceeds as follows:
(1) The start state of the NFA is the ε-closure of the start state of the NFA
(2) For the created state in (1), and for any created state in (2) do:
  For each possible input symbol:
    Apply the move function to the created state with the input symbol
    For the resulting set of states, apply the ε-closure function; this might or might not result in a new set of states.
(3) Continue (2) until no more new states are being created.
(4) The final states of the DFA are the ones containing any of the final states of the NFA

(e) [Bookwork/New computed example] 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)

(f) [Bookwork] The closure(L) of a set of items L for a grammar G is the set of items constructed from L using the following two rules:
- Initially, every iterm in L is added in closure(L)
- If A -> α.Bβ in closure(L) and B-> γ is a production, then add item B -> .γ to L if its not already there. Apply this rule until no more new items can be added to closure(L).

(g) [New computed example]:
- Flow of control optimization example – removal of unnecessary jumps:

  Replace jump sequences of the form

Goto L1		Goto L2
...	with	...
L1: Goto L2		L1: Goto L2

- Copy propagation:

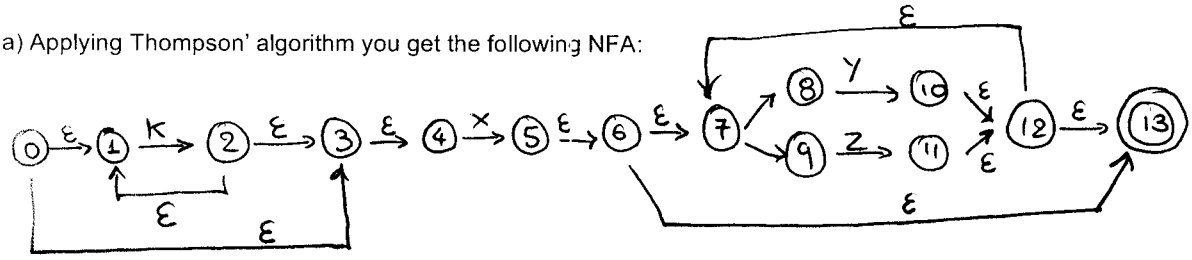
  If X and X are not subsequently modified:

X : = Y		
A[1] = X	can be replaced with	A[1] = Y
A[2] = Y		A[2] = Y

  Saving the cost of one assignment.

Question 2
[new computed example]

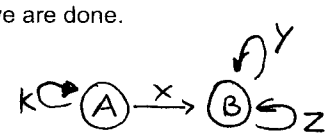
(a) Applying Thompson' algorithm you get the following NFA:



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

ε-closure(StartState) = {0,1,2,3,4} = A
ε-closure(move(A, k)) = {0,1,2,3,4} = A
ε-closure(move(A, y)) = ε-closure(move(A, z)) = {}
ε-closure(move(A, x)) = {5, 6, 7, 8, 9, 10, 11, 12, 13} = B
ε-closure(move(B, x)) = ε-closure(move(B, k)) = {}
ε-closure(move(B, y)) = {5, 6, 7, 8, 9, 10, 11, 12, 13} = B
ε-closure(move(B, z)) = {5, 6,7, 8, 9, 10, 11, 12, 13} = B
– no more new created states; we are done.

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} and final: {B}. 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:

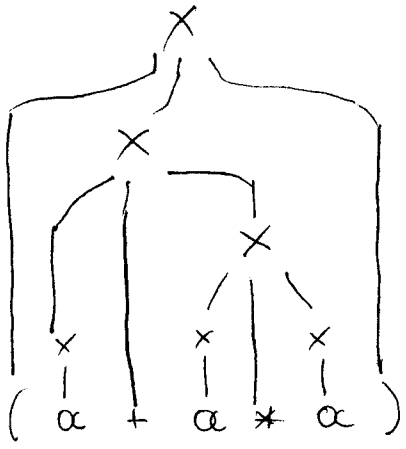
(a) [New computed example]: FIRST(X) = FIRST(Y) = FIRST(F) = {o,z}; FIRST(K) = {p, ε}; FIRST(L) = {t,ε}

FOLLOW(X) = FOLLOW(K) = {c, $}; FOLLOW(Y) = FOLLOW(L) = {p, c, $}; FOLLOW(F) = {p, t, c, $}

(b) [New computed example]

Step	Input	Stack	Action
(1)	(a+a*a)$		Shift
(2)	a+a*a)$	(	Shift
(3)	+a*a)$	(a	Reduce (r4)
(4)	+ a*a)$	(X	Shift
(5)	a*a)$	(X +	Shift
(6)	*a)$	(X + a	Reduce (r4)
(7)	*a)$	(X + X	Shift
(8)	a)$	(X + X*	Shift
(9)	)$	(X + X * a	Reduce (r4)
(10)	)$	(X + X * X	Reduce (r2)
(11)	)$	(X + X	Reduce (r1)
(12)	)$	(X	Shift
(13)	$	(X)	Reduce (r3)
(14)	$	X	Accept

Parse tree:



Question 4:

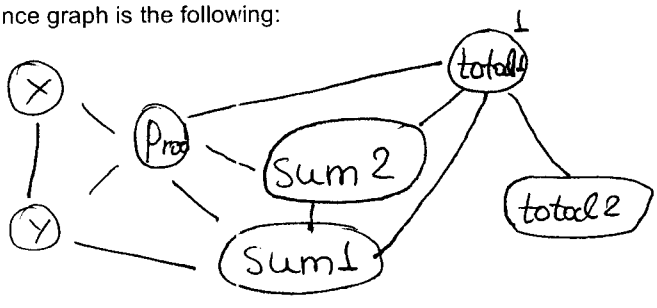
(a) [Bookwork] For a given code segment, an interference graph is an undirected graph where each variable is a node, and arcs represent nodes that cannot share a register (because they are live at the same time i.e. they interfere with each other wrt register allocation).

We can map this register allocation to graph colouring theory, transforming it to the problem of how to colour the nodes of a graph with the lowest possible number of course, such that for each arc, the nodes at its ends have different colours.

(b) In an interference graph an edge connects two nodes if one is live at a point where the other is defined. Thus

- In the first statement, prod, x and y must be connected, since x and y are both live when prod is defined.
- Following the second statement, x is not needed anymore, but y is (next statement), so sum1 and y must be connected.
- In the third statement, sum1 and prod are both live when sum2 is defined, so they must be connected
- In the fourth statement, prod, sum1 and sum2 are live when total1 is defined, so they should all be connected.
- Following the fifth statement, only total1 is live when total2 is defined, so total1 and total2 should be connected.

Therefore, the interference graph is the following:



ii. This interference graph is 4-colourable since the nodes for x, y, sum1, total1 and total2 all have fewer than 4 neighbours, which means that they can be removed along with their edges. In the new graph prod now has no neighbours, and can also be removed. We have resulted into an empty graph, which means that 4-colouring of the original intereference graph is possible.

Allocating registers: we have to make sure that no two connected nodes share the same register (share the same "colour")

Several equivalent possibilities exist – one of them being:

X: R2, Y: R1, prod:R0, sum1: R2, sum2: R1, total1: R3, total2: R1