E2.15: Language Processors
Sample answers to exam questions 2014

Question 1

(a) [bookwork] Type 0 (unrestricted grammars), 1 (for all productions α->β, we must have $|\alpha| \leq |\beta|$), 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).

[2 points for each correct condition for types 1,2,3] [6]

(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, $\Sigma \subseteq \Gamma$
- $\delta(q,X)$: the transition function between states. Given a state q $\in$Q, and a tape symbol X $\in \Gamma$, the function $\delta(q,X)$ returns a triplet (p,Y, D), where p is the new state in Q, Y $\in \Gamma$ 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 ($B\in\Gamma$ but $B\notin\Sigma$) initially written in all tape cells except the ones holding the input
- F: the set of final states, $F\subseteq Q$

[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.
[4 points for specifying the turing machine, 2 points for specifying the differences][6]

(c) [bookwork] A *translation scheme* is a CF grammar in which program fragments are embedded in the 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')}
```

[3 points for the definition and 3 points for the example] [6]

(d) [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

[3 points for each function] [6]

(e) [Bookwork] Basic blocks are sequences of consecutive statements in which flow of control enters at the beginning and leaves at the end without a halt or possibility of branching.
The algorithm first determine the set of *leaders*, the first statements of basic blocks, using the following rules:

a. The first statement is a leader
b. Any statement that is the target of a conditional or unconditional goto is a leader
c. Any statement that immediately follows a goto or conditional goto statement is a leader.

For each leader, its basic block consists of the leader and all statements up to but not including the next leader. Once the basic blocks have been defined, a number of transformations can be applied to them to improve the quality of code without worrying about accidentally introducing flow of control errors.

[2 points for the definition of basic block, 4 points for the algorithm] [6]

(f) [Bookwork] *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".*

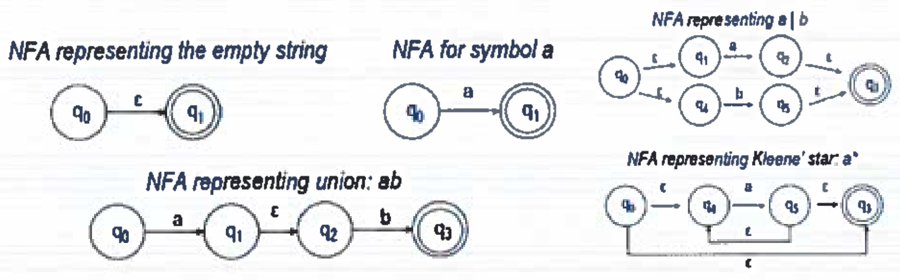
[3 points for each of the first 3 rules, 1 point for the last one] [10]

Question 2 [new computed example]

(a)

(i)

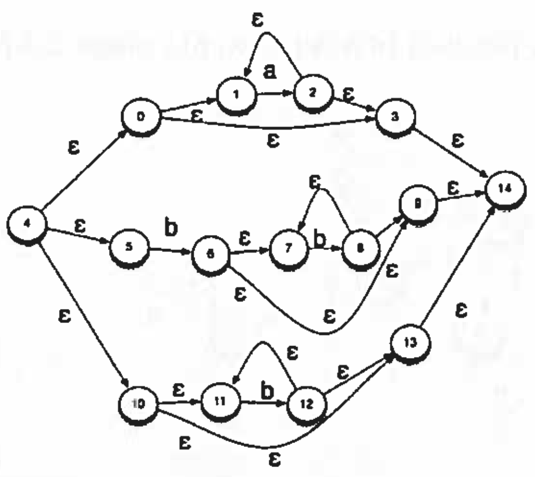
Start by constructing an NFA for each symbol and for each operator (|, union, Kleene *); join the resulting NFA with ε-transitions (in precedence order). The NFA blocks are:


NFA representing the empty string
q0
ε
q1
NFA for symbol a
q0
a
q1
NFA representing a | b
q0
q1
q2
q3
q4
q5
a
b
ε
NFA representing union: ab
q0
a
q1
ε
q2
b
q3
NFA representing Kleene' star: a*
q0
q4
q5
q3
ε
a

[4]

PCE: Again, variable level of description, but the majority got this one right. People lost marks by forgetting one or more of the NFA blocks above, and describing the overall process.

(ii) Applying Thompson' algorithm you get



End states: 2, 8, 12, 14 (3,9,13 can also be considered final too) **[6]**

(b) (i)

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 takes 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:
The start state of the DFA is the ε-closure of the start state of the NFA
2. For the created state of (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. **[1 point for each function, 2 points for the algorithm]**

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

ε-closure(StartState) = ε-closure{4} = {4, 0, 1, 3, 14, 5, 10, 11, 13} = A

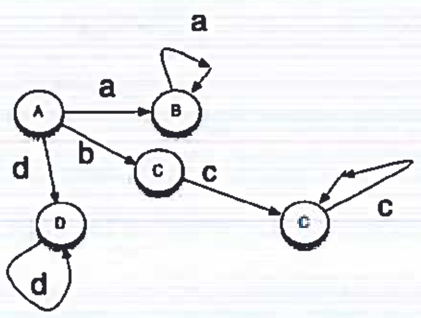
ε-closure(move(A, a)) = ε-closure{2} = {1, 2, 3, 14} = B
ε-closure(move(A, b)) = ε-closure{6} = {6, 7, 9, 14} = C
ε-closure(move(A, c)) = {}
ε-closure(move(A, d)) = ε-closure{12} = {11,12,13,14} = D

ε-closure(move(B, a)) = {1, 2, 3, 14} = B
ε-closure(move(B, b)) = ε-closure(move(B, c))= ε-closure(move(B, d)) = {}

ε-closure(move(C, a)) = ε-closure(move(C, b)) = ε-closure(move(C, d)) = {}
ε-closure(move(C, c)) = {7,8,9,14} = E

ε-closure(move(D, a)) = ε-closure(move(D, b)) = ε-closure(move(D, c)) = {}
ε-closure(move(D, d)) = ε-closure{12} = {11,12,13,14} = D

No more new states, we are done. The resulting DFA is below. ALL states are final states.



[6]

(c)

i. 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 – in our case they are all final. For each state set created, the algorithm examines the transitions for each state and for each input symbol. We continue until no more new states can be created. **[5]**

ii. In our case this set is further subdivided in 5 sets, A, B, C, D, and E, since all of them go to different states with different symbols. Thus our DFA is already minimal.

[5]

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).

[1 point for each correct set of LR(0) items constructed]

(b)

(Rule 1) For a terminal *a*, if [A -> α.aB] is in Ii and goto(Ii, a) = Ij, then set action[i, a] to shift j
(Rule 2) If [A -> α.] is in Ii, 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 Ii, then set action[i,$] to "accept".

The goto transitions for state i are constructed for all nonterminals using the rule:

(Rule 4) if goto(Ii, A) = Ij, then goto[i,A]=j All entries not defined by the rules above are made "error", indicated by blank entries.

[2 points for each correct rule provided]

(c)

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			

[2 points for getting the S action entries correct, 2 points for getting the R entries correct, 1 point for getting the acc state correct, 5 point for getting the goto entries correct (1 per entry)]