UNIVERSITY OF MORATUWA

Faculty of Engineering

Department of Computer Science and Engineering

B.Sc. Engineering Honours Degree

Semester 4 Examination (2020 Intake)

CS 3513 Programming Languages

Time allowed: 2 Hours July 2023

Instructions to candidates

1. This paper consists of 4 questions in 10 pages.
2. Answer ALL 4 questions.
3. Start answering each of the 4 main questions on a new page.
4. The maximum attainable mark for each question is given in brackets.
5. This examination accounts for 60% of the module assessment.
6. This is a closed book examination.

   It is an offence to be in possession of unauthorized material during the examination.
7. Only calculators approved and labelled by the Faculty of Engineering are permitted.
8. Assume reasonable values for any data not given in or with the examination paper or Appendix. Clearly state such assumptions made on the script.
9. Appendices in pages 7-9 contains RPL Grammar rules, CSE machine rules and lambda calculus axioms.
10. In case of any doubt as to the interpretation of the wording of a question, make suitable assumptions and clearly state them on the script.
11. This paper should be answered only in English.

Q1. [25 marks]

Consider the following RPAL program which takes two equal length strings as arguments and returns the pairs of letters in the same position of the two strings. As an example, Pairs (‘abc’, ‘def’) will produce (ad,be,cf) as the result.

```
let rec Rev S =
    S eq '' -> ''
   | (Rev(Stern S)) @Conc (Stem S )
in let
    Pairs (S1,S2) = P (Rev S1, Rev S2)
    where rec P (S1, S2) =
      S1 eq '' & S2 eq '' -> nil
     | (fn L. P (Stern S1, Stern S2) aug ((Stem S1) @Conc (Stem S2)))
       nil
 in Print ( Pairs ('abc','def'))
```

(a) Construct the Abstract Syntax Tree (AST) for the above Program.

[5 marks]

(b) Construct the Standardize Tree (ST) for the above Program using the AST constructed in above (a). Transformational Rules are provided in the appendix. [5 marks]

(c) List down the Control Structures of the above program. [5 marks]

(d) Show the Control Stack Environment (CSE) machine evaluation for the above program. [10 marks]

Q2. [25 marks]

(a) Write an RPAL program which implements function Innerproduct. Function Innerproduct multiplies two N-tuples of integers element by element and returns the sum of the products.

Examples: Innedrproduct ((1,2,3), (4,5,6)) = 1*4 + 2*5 + 3*6 = 32

Innerproduct (nil, nil) = 0

The program should check for the erroneous inputs as well.

[6 marks]

(b) Construct the Abstract Syntax Tree (AST) for the program you wrote for part (a).

[6 marks]

(c) Write a (short) program in Pascal/C-like pseudo-code that will output 0 if copy-in,copyout is used, 1 if pass by reference is used, and 2 if pass-by-value is used. Show a legible trace of the program's execution. [7 marks]

(d) For the program shown below, draw a picture of the run-time environment, at the point marked "HERE", i.e., at the point when fantorial is about to return after executing " *if n = 1 then return tot* " statement. Show all temporaries, local variables, parameters, locations reserved for return values, and return addresses, as well as the base pointer, frame pointer(s), and the stack pointer. [6 marks]

```
function fantorial (int n, int tot) {
        int fan;
        if n = 1 then return tot // HERE
        else {
                fan = fantorial (n-1, n*tot);
                return fan;
        }
}
main( )
begin
        write(fantorial (5,1))
end;
```

Q3. [30 Marks]

The following Context-free grammar generates regular expressions, over the vocabulary Σ={a,b}.

Expression	-> Expression '|' Term
	-> Term
Term	-> List Term
	-> List
List	-> List 'list' Factor
	-> Factor
Factor	-> Factor '*'
	-> Factor '+'
	-> Factor '?'
	-> Primary
Primary	-> '(' Expression ')'
	-> 'a'
	-> 'b'

(a) Show that the above grammar is not LL(1). [1 Marks]

(b) Transform the above grammar into an equivalent grammar that is LL(1). Show the Select sets that make the grammar LL(1).

[5 Marks]

(c) Build a parse table from your resulting grammar in Part (b).

[6 Marks]

(d) Using your parse table from Part (c), show how the following input string is parsed.

```
(ab*) list (b|a) a?
```

[6 Marks]

(e) Using the table lookups from Part (d), build the derivation tree (top-down, left-most derivation) for the string parsed in Part (d). [6 Marks]

(f) Write (pseudo) code for Recursive Descent parser for the LL(1) version of the grammar you created in (b). Include "Write()" statements to build the Derivation Tree in bottom-up fashion. [6 Marks]

Q4. [20 marks]

(a) Using the RPAL grammar construct the abstract syntax tree (if possible) for each of the following programs: [1x7 = 7 Marks]

```
  i).x or y -> x | z -> x | y
 ii).x eq y -> z -> x | y | x
iii).x aug y , z , 3
 iv).x , y , z aug 3
  v).let f (x) = x * 2 + 1 in f (z) where z = 6
 vi).let x = z and y = 2 * z where z = 3 in x ** y ** y
vii).let x = z in x ** y ** y where z = 3 and y = 2 * z
```

(b) Show that the fixed-point finder $Y = \lambda F.(\lambda f.f\ f)(\lambda g.F(g\ g))$ is a fixed-point of the function $H = \lambda y.\lambda f.f(y\ f)$. [3 marks]

(c) List (you do not have to explain) an example binding that will happen in each of the following times.

I. Language design time
II. Language implementation time
III. Programme writing time
IV. Compile time
V. Run time [1 x 5 marks]

(d) Discuss the advantages and disadvantages of binding things as early as possible and delaying bindings in implementing programming languages. Use suitable examples for your explanation. [2 marks]

(e) Consider the following clauses:

```
studies(charlie, csc135).
studies(olivia, csc135).
studies(jack, csc131).
studies(arthur, csc134).

teaches(kirke, csc135).
teaches(collins, csc131).
teaches(collins, csc171).
teaches(juniper, csc134).

professor(X, Y) :- teaches(X, C), studies(Y, C).
```

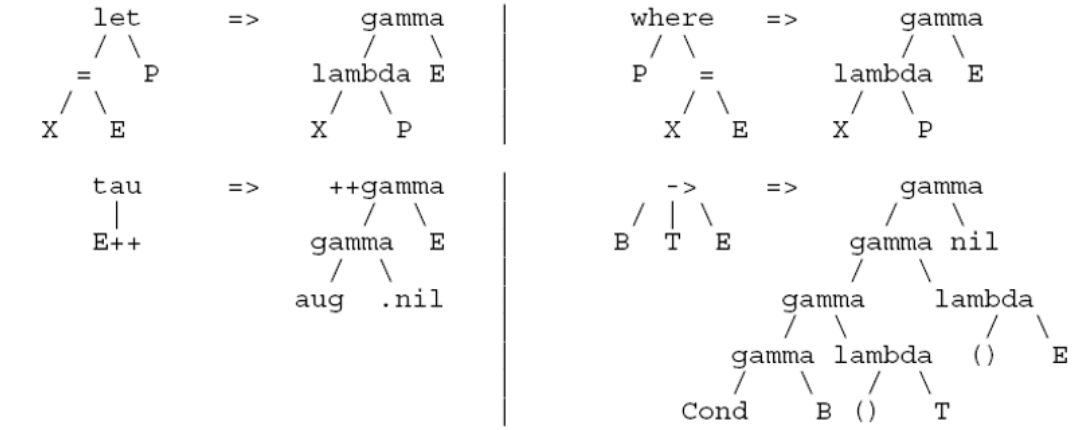
Write the prolog answers for each of the following query.

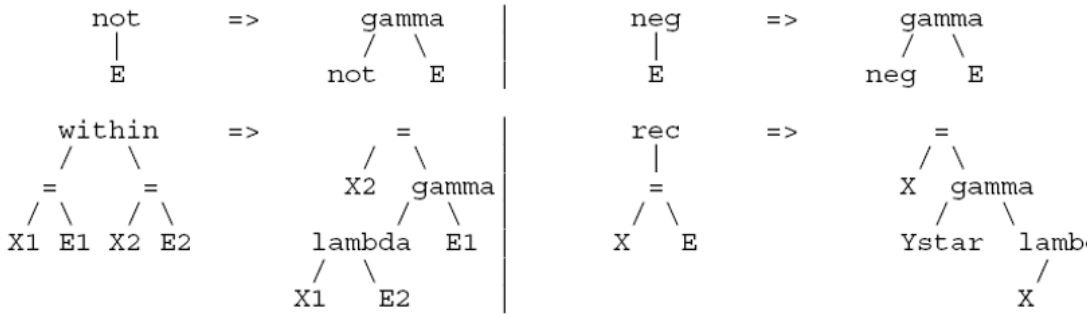
i. ?- studies(charlie, What) [1 mark]
ii. ?- professor(kirke, Students). [1 mark]
iii. ?- studies(Who, csc135) [1 mark]

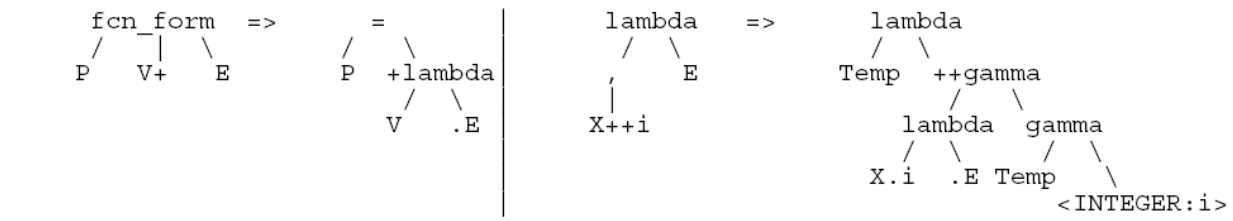
---------- End of the Examination Paper ----------

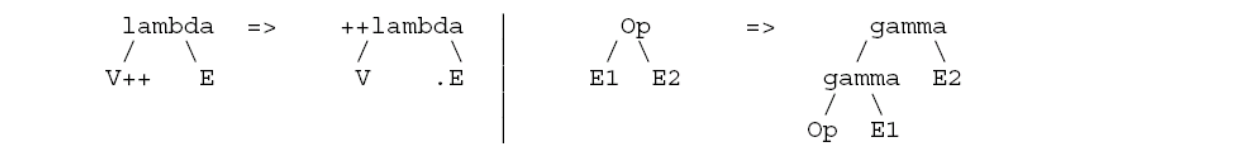
Appendix

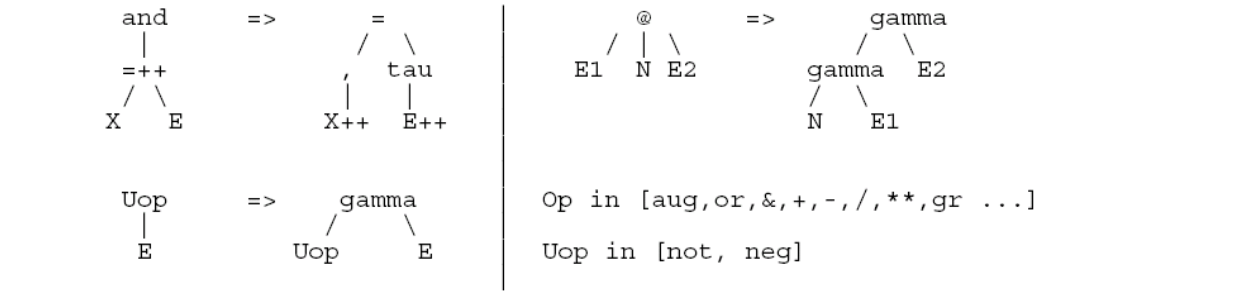
A. RPAL Subtree Transformational Gramma











B. CSE Machine Rules

	CONTROL	STACK	ENV
Initial State	$e_0\ \delta_0$	e_0	e_0 = PE
CSE Rule 1 (stack a name)	.... Name	.... Ob	Ob=Lookup(Name,e_c) e_c:current environment
CSE Rule 2 (stack λ)	.... λ_k^x	.... $^c\lambda_k^x$	e_c:current environment
CSE Rule 3 (apply rator)	.... γ	Rator Rand .... Result	Result=Apply[Rator,Rand]
CSE Rule 4 (apply λ)	.... γ .... $e_n\ \delta_k$	$^c\lambda_k^x$ Rand .... e_n	$e_n = [Rand/x]e_c$
CSE Rule 5 (exit env.)	.... e_n	value e_n .... value	
CSE Rule 6 (binop)	.... binop	Rand Rand .... Result	Result=Apply[binop,Rand,Rand]
CSE Rule 7 (unop)	.... unop	Rand .... Result	Result=Apply[unop,Rand]
CSE Rule 8 (Conditional)	.... $\delta_{then}\ \delta_{else}\ \beta$	true	
CSE Rule 9 (tuple formation)	.... τ_n	V_1 ... V_n .... $(V_1,...,V_n)$	
CSE Rule 10 (tuple selection)	.... γ	$(V_1,...,V_n)$ I .... V_I	
CSE Rule 11 (n-ary function)	.... γ .... $e_m\ \delta_k$	$^c\lambda_k^{V_1,...,V_n}$ Rand .... e_m	e_m=[Rand 1/V_1]... [Rand n/V_n]e_c
CSE Rule 12 (applying Y)	.... γ	Y $^c\lambda_i^v$ .... $^c\eta_i^v$	
CSE Rule 13 (applying f.p.)	.... γ .... $\gamma\ \gamma$	$^c\eta_i^v$ R .... $^c\lambda_i^v$ $^c\eta_i^v$ R	

C. RPAL's Phrase Structure Grammar

```
# Expressions ###############################################

E    -> 'let' D 'in' E                          => 'let'
     -> 'fn'  Vb+ '.' E                         => 'lambda'
     ->  Ew;
Ew   -> T  'where' Dr                           => 'where'
     -> T;

# Tuple Expressions #########################################

T    -> Ta ( ',' Ta )+                          => 'tau'
     -> Ta ;
Ta   -> Ta 'aug' Tc                             => 'aug'
     -> Tc ;
Tc   -> B '->' Tc '|' Tc                        => '->'
     -> B ;

# Boolean Expressions #######################################

B    -> B 'or' Bt                               => 'or'
     -> Bt ;
Bt   -> Bt '&' Bs                               => '&'
     -> Bs ;
Bs   -> 'not' Bp                                => 'not'
     -> Bp ;
Bp   -> A ('gr' | '>' ) A                       => 'gr'
     -> A ('ge' | '>=') A                       => 'ge'
     -> A ('ls' | '<' ) A                       => 'ls'
     -> A ('le' | '<=') A                       => 'le'
     -> A 'eq' A                                => 'eq'
     -> A 'ne' A                                => 'ne'
     -> A ;

# Arithmetic Expressions ####################################

A    -> A '+' At                                => '+'
     -> A '-' At                                => '-'
     ->   '+' At
     ->   '-' At                                => 'neg'
     -> At ;
At   -> At '*' Af                               => '*'
     -> At '/' Af                               => '/'
     -> Af ;
Af   -> Ap '**' Af                              => '**'
     -> Ap ;
Ap   -> Ap '@' '<IDENTIFIER>' R                 => '@'
     -> R ;

# Rators And Rands ##########################################

R    -> R Rn                                    => 'gamma'
     -> Rn ;
Rn   -> '<IDENTIFIER>'
     -> '<INTEGER>'
     -> '<STRING>'
     -> 'true'                                  => 'true'
     -> 'false'                                 => 'false'
     -> 'nil'                                   => 'nil'
     -> '(' E ')'
     -> 'dummy'                                 => 'dummy' ;

# Definitions ###############################################

D    -> Da 'within' D                           => 'within'
     -> Da ;
Da   -> Dr ( 'and' Dr )+                        => 'and'
     -> Dr ;
Dr   -> 'rec' Db                                => 'rec'
     -> Db ;
Db   -> Vl '=' E                                => '='
     -> '<IDENTIFIER>' Vb+ '=' E                => 'fcn_form'
     -> '(' D ')' ;

# Variables #################################################

Vb   -> '<IDENTIFIER>'
     -> '(' Vl ')'
     -> '(' ')'                                 => '()';
Vl   -> '<IDENTIFIER>' list ','                 => ','?;
```

D. Lambda Calculus Axioms

Axiom Delta:

Let M and N be AE's that do not contain λ-expressions.
Then $M \Rightarrow_{\delta} N$ if Val(M) = Val(N).

Axiom Alpha:

Let x and y be names, and M be an AE with no free occurrences of y. Then, in any context,
$\lambda\ x.M \Rightarrow_{\alpha} \lambda\ y.subst[y,x,M]$

Axiom Beta:

Let x be a name, and M and N be AE's. Then, in any context,
$(\lambda x.M)\ N \Rightarrow_{\beta} subst[N,x,M]$.

Axiom ρ (Fixed point identity):

$Y\ F \Rightarrow_{\rho} F\ (Y\ F)$.