UNIVERSITY OF MORATUWA

Faculty of Engineering

Department of Computer Science and Engineering

B.Sc. Engineering Honours Degree

Semester 5 Examination (2018 Intake)

CS 3512 Programming Languages

Time allowed: 2 Hours   November 2021

Instructions to candidates

1. This paper consists of 4 questions in 9 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 page 6-9 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]

(a) For $L_1 = \{a, bb, c\}$ and $L_2 = \{ac, ca\}$, calculate L_1L_2, $L_1 \cup L_2$, and L_1^3.

[5 Marks]

(b) Consider the following grammar.

Sentence → NP VP
NP → N
NP → Adj NP
N → boy
N → girl
Adj → the
Adj → tall
Adj → jealous
VP → V NP
V → hit
V → bit

produce the derivation of the sentence "jealous tall the girl bit the boy".

[5 Marks]

(c) State the main characteristic of productions in a regular grammar, that make those productions different from those in a context-free grammar. [2 Marks]

(d) Construct a regular grammar for simple floating-point constants (e.g. 196.32) in a typical programming language. [5 Marks]

(e) Construct a context-free grammar for the language $\{a^{3n+1} b^{2n} / n > 0\}$.

[3 Marks]

(f) Briefly explain the following phases in a compiler.

I. Scanner
II. Screener
III. Parser
IV. Constrainer
V. Code Generator [1 x 5 marks]

Q2. [25 marks]

Consider the following RPAL program which takes a tuple argument, and reverses it, i.e. returns a tuple with the same values as the original one, but in reverse order. In addition, tuple_reverse(nil)=nil.

```
let tuple_reverse t =
        not Istuple t ->  'error'
    |   tr  t  (Order t) 1 where
        rec  tr  t  n  i  =
                i > n   ->  nil
            |   (tr  t  n  (i+1)  aug  (t  i))
in Print ( tuple_reverse (1,2,3) )
```

(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]

Q3. [25 marks]

(a) Write a RPAL program that defines and uses a function Sqr_sum which takes two tuple arguments, squares their components, and computes the component-wise sum of the squared values. For example,

Sqr_sum(1,2,3) (3,4,5) should return (10,20,34).

This function must print the computed vector, or the string 'error'. Check for errors such as either argument not a tuple, one tuple longer than the other and any non-integer tuple elements. [6 marks]

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

[6 marks]

(c) For the program shown below, draw a picture of the run-time environment, at the point marked "HERE", i.e., **at the point when the constant value “1” has been placed on the stack**. 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. [7 marks]

```
function fib (int n) {
        int t;
        if n <= 1 then return 1 // HERE
        else {
                t = fib (n-1) + fib (n-2);
                return t;
        }
}
main( )
begin
        write(fib(4))
end;
```

(d) Show the output of the following program segment (written in C-like syntax) for each of the following parameter-passing mechanisms:

I. pass by value-result, [2 marks]
II. pass by value, and [2 marks]
III. pass by reference. [2 marks]

```
int a;
void f(int b) {
   print(b);
   b = 3;
   print(a+b);
}
main() {
   a = 8;
   f(a);
   print(a);
}
```

Q4. [25 marks]

Consider the following regular expression: (a*bc + d*e)*

(a) Transform this regular expression to an NFA, from there to a right-linear regular grammar. [3 marks]

(b) Transform the NFA from part (a) , to a DFA. [3 marks]

(c) Minimize the DFA obtained in Part (b). [3 marks]

(d) Write (in pseudo-code) a lexical analyzer for the language given by the above regular expression.
Write two versions of the lexical analyzer:
I. Table-driven [3 marks]

II. Hard-coded. [4 marks]

(e) Write the corresponding λ-expression of the below RPAL program. [4 marks]

```
let rec f x n i =
      ( i eq 0 ) -> ( x + n ) |
      ( n eq 0 ) -> ( i ls 3 ) -> ( i - 1 ) | x
            | ( f x ( f x ( n – 1 ) i ) ( i -1 ) )
in f 2 3 1
```

(f) Consider the following clauses:

```
male(james1).
male(charles1).
male(charles2).
male(james2).
male(george1).

female(catherine).
female(elizabeth).
female(sophia).

parent(charles1, james1).
parent(elizabeth, james1).
parent(charles2, charles1).
parent(catherine, charles1).
parent(james2, charles1).
parent(sophia, elizabeth).
parent(george1, sophia).
```

Write Prolog predicates for each of the following:

i. MotherOf(X,Y)
ii. FatherOf(X,Y)
iii. SisterOf(X,Y)
iv. BrotherOf(X,Y)
v. AuntOf(X,Y)

[1 x 5 marks]

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

Appendix

A. RPAL Subtree Transformational Gramma

```
     let      =>      gamma       |      where    =>      gamma
     / \              /   \       |       / \             /  \
    =   P          lambda  E      |      P   =        lambda  E
   / \              /  \          |          / \       /   \
  X   E            X    P         |         X   E     X     P

     tau      =>    ++gamma       |        ->     =>      gamma
      |               /  \        |      / | \            /   \
     E++          gamma   E       |     B  T  E       gamma  nil
                   /  \           |                     /   \
                 aug  .nil        |               gamma     lambda
                                  |                /  \       /  \
                                  |           gamma lambda   ()   E
                                  |            /    \  / \
                                  |         Cond    B ()  T

     not      =>      gamma       |       neg     =>      gamma
      |               /  \        |        |              /  \
      E             not   E       |        E            neg   E

    within    =>        =         |       rec     =>        =
    /    \             / \        |        |               / \
   =      =          X2  gamma    |        =              X  gamma
  / \    / \             /  \     |       / \                /   \
 X1 E1  X2 E2      lambda    E1   |      X   E           Ystar  lambda
                    /  \          |                              /  \
                  X1    E2        |                             X    E
```

```
    fcn_form  =>       =          |     lambda    =>      lambda
    /   |   \         / \         |      / \              /  \
   P    V+   E       P  +lambda   |     ,   E         Temp   ++gamma
                          /  \    |     |                     /   \
                         V    .E  |   X++i               lambda   gamma
                                  |                       /  \     /  \
                                  |                     X.i  .E Temp   \
                                  |                                  <INTEGER:i>

      lambda  =>     ++lambda     |       Op      =>      gamma
      /   \           /     \     |      / \              /   \
    V++    E         V      .E    |     E1  E2        gamma    E2
                                  |                    / \
                                  |                   Op  E1

      and     =>        =         |       @       =>      gamma
       |               / \        |     / | \             /  \
      =++             ,   tau     |   E1  N  E2      gamma   E2
      / \             |    |      |                   /  \
     X   E          X++   E++     |                  N    E1

      Uop     =>     gamma        |   Op in [aug,or,&,+,-,/,**,gr ...]
       |             /    \       |
       E           Uop     E      |   Uop in [not, neg]
```

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 .... $(V_1,\ldots,V_n)$	
CSE Rule 10 (tuple selection)	.... γ	$(V_1,\ldots,V_n)$ I .... V_I	
CSE Rule 11 (n-ary function)	.... γ .... $e_m\ \delta_k$	${}^c\lambda_k^{V_1,\ldots,V_n}$ 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 $=>_{\delta}$ 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,
 λ x.M $=>_{\alpha}$ λ y.subst[y,x,M]

Axiom Beta:

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

Axiom ρ (Fixed point identity):

Y F $=>_{\rho}$ F (Y F).