Eastern University, Sri Lanka
Department of Mathematics
ial Degree Examination in Computer Science 2013/2014 (Mar/Apr, 2016)
CSS 08: Compiler Design

Answer **All** questions. (This paper has 6 questions on 6 pages.)

Time allowed: **Three Hours**.

Start a new page for each question.

Allocate your time wisely, the point value of each part is shown in square brackets.

At the bottom of the front page of your answer book, write the question numbers in the order you answered.

a) ***State*** clearly what you understand by *regular expression* and *nondeterministic finite automaton (NFA)*. [

b) Consider the alphabet {`a, b`}. ***Write*** shortest regular expressions for the following languages: [

i. All strings that contain exactly one "`a`" and at least one "`b`".
(E.g., `ab, bbabbb, bbba, .....`)

ii. All strings that contain 0 or more "`a`"s and an even number of "`b`"s. (Note: can contain 0 "`b`"s.)

c) ***State*** whether the language given below is regular or not. ***Explain*** the reason. [

$$\{\mathtt{a}^{\mathtt{p}}(\mathtt{bc})^{\mathtt{q}}(\mathtt{d})^{\mathtt{p}} \mid \mathtt{p}, \mathtt{q} \geq 0\}$$

d) ***Draw*** the *NFA fragments* for the following regular expressions: [

i. st ii. $s|t$ iii. s^*

e) Considering the alphabet {`a, b`}. ***Construct*** an NFA that is able to recognize the sentences generated by the regular the expression $(a^*ba^*b)^*a^*$. [

2. *Context-free grammars* are powerful enough to describe the syntax of most pro[gramming]
languages.

(a) Comparing with *regular expressions*, context-free grammars are capable of [describing]
much more complex languages.

With the help of suitable example, ***validate*** this statement.

(b) A language L has been defined with the following Grammar:

$$E \rightarrow E \otimes F$$
$$E \rightarrow F$$
$$F \rightarrow G \ominus F$$
$$F \rightarrow F \oplus G$$
$$F \rightarrow G$$
$$G \rightarrow \mathtt{id}$$

where, $\otimes$, $\ominus$, and $\oplus$ are mathematical operators.

i. By considering the example string $\mathtt{id} \ominus \mathtt{id} \oplus \mathtt{id}$, ***show*** that the ab[ove grammar]
is ambiguous.

ii. Suppose $\oplus$ is associative, and $\otimes$, $\ominus$ are left-associative and right-a[ssociative re]spectively. ***Explain*** how you rewrite the above grammar to an [unambiguous]
grammar.

iii. Consider the following production with a *right-associative* operator

$$Exp \rightarrow Exp \uparrow Exp \mid \mathtt{id}$$

Suppose $\uparrow$ has the *highest priority* than $\otimes$, $\ominus$, and $\oplus$,

(α) ***explain*** how you remove the ambiguity, and

(β) ***show*** how the above given language L can be extended wit[h this pro]duction.

The *LL(1) parsing* is a *predictive parsing* method, which is suitable for formal languages with unambiguous grammars.

(a) One of the cause for *conflicts* in LL(1) parsers is *left-recursion*. Briefly describe how you eliminate left-recursion from the following grammar: [25%]

$$
\begin{aligned}
E &\to E + T \mid T \\
T &\to T \times F \mid F \\
F &\to (E) \mid \texttt{id}
\end{aligned}
\tag{2}
$$

(b) ***State*** how the FIRST and FOLLOW sets can be used to construct an LL(1) parse table. [15%]

(c) Consider the following Grammar:

$$
\begin{aligned}
E &\to TE' \\
E' &\to +TE' \mid \epsilon \\
T &\to FT' \\
T' &\to \times FT' \mid \epsilon \\
F &\to \texttt{id} \mid (\ E\ )
\end{aligned}
\tag{3}
$$

where, "id", "+", "×", "(" and ")" are terminals.

i. Compute the FIRST and FOLLOW sets for each non-terminal. [30%]

ii. Construct the LL(1) parse table for the above given grammar. Explain each step clearly. [30%]

4. *SLR parser* is a type of *LR* parser with a relatively simple parser generator algo

   (a) ***Describe*** the *shift*, *go* and *reduce* actions associated with an SLR parser.

   (b) ***State*** and ***describe*** the steps for constructing an SLR parsing table from

   (c) Consider the following Grammar:

$$S \to a\ S\ b\ T$$

$$S \to a$$

   where, "a" and "b" are terminals.

   i. Compute the FIRST and FOLLOW sets for each non-terminal.

   ii. Construct the SLR parse table for the above given grammar. Explai clearly.

a) Programming languages use *names* to refer objects, such as variables, constants, types, and functions.

i. ***State*** what you understand by *scope* of a name. [15%]

ii. Scoping based on the structure of the syntax tree is called *static* or *lexical binding*. Briefly ***describe*** how this method finds the scopes for the names. [25%]

b) Consider the following portion of a grammar of an example language:

$$\begin{array}{lcl} Exp & \rightarrow & \textbf{num} \\ Exp & \rightarrow & \textbf{id} \\ Exp & \rightarrow & Exp + Exp \end{array}$$

where, "`num`" is a number, and "`id`" is an identifier.

The figure below shows the type-checking function $Check_{EXP}$ for the productions given above.

$Check_{Exp}(Exp, vtable, ftable)$ = `case` Exp `of`	
num	`int`
id	$t = lookup(vtable, getname(\textbf{id}))$ *if* $t = unbound$ *then* **error()**; `int` *else* t
$Exp_1 + Exp_2$	$t_1 = Check_{Exp}(Exp_1, vtable, ftable)$ $t_2 = Check_{Exp}(Exp_2, vtable, ftable)$ *if* t_1 = `int` *and* t_2 = `int` *then* `int` *else* **error()**; `int`

where, *vtable* and *ftable* are symbol tables for variables and functions respectively. The function *getname* extracts the name of an identifier, and the function *getvalue* returns the value of a number.

i. Briefly ***describe*** how each of the three cases handled by $Check_{EXP}$. [30%]

ii. Suppose the grammar has a production for a comparison operation

$$Exp \rightarrow Exp = Exp$$

Here the comparison requires that the arguments are of same type, and then the result is boolean.

Extend the type-checking function to handle these new constructions, and ***explain*** it clearly. [30%]

6. Many compilers generate a linearisation of the syntax tree as an *intermediate* co

(a) Briefly ***describe*** two advantages of generating an intermediate code rather ducing the machine code directly.

(b) Consider an example language with the following portion of a grammar:

$$\begin{aligned} Exp &\rightarrow \textbf{num} \\ Exp &\rightarrow \textbf{id} \\ Exp &\rightarrow \textbf{unop}\ Exp \end{aligned}$$

where, "num", "id" and "unop" are a number, an identifier and a una respectively.

The figure below shows the translation function $Trans_{EXP}$ for the produ above.

$Trans_{Exp}(Exp, vtable, ftable, place) =$ case Exp of	
num	$v = getvalue(\textbf{num})$ $[place := v]$
id	$x = lookup(vtable, getname(\textbf{id}))$ $[place := x]$
unop Exp_1	$place_1 = newvar()$ $code_1 = Trans_{Exp}(Exp_1, vtable, ftable, place_1)$ $op = transop(getopname(\textbf{unop}))$ $code_1 +\!+ [place := op\ place_1]$

where, $vtable$, $ftable$ are symbol tables for variables and functions res the "place" is the intermediate-language variable that the result of the ex be stored in. The *getname* is a function that extracts the name of an the function *getvalue* returns the value of a number. A function *new* generate new variable names in the intermediate language.

i. Suppose the grammar has a production for a binary operation

$$Exp \rightarrow Exp\ \textbf{binop}\ Exp$$

Extend the translation function to handle this binary operation, clearly.

ii. Assume a variable symbol table that binds x to v_0. The "place" t_0 and calls to *newvar* return the variables $t_1, t_2, t_3, \ldots$ in sequence ***Use*** the given translation functions and the one you have writte to ***generate*** code for the following expression:

$$-(x \times 3)$$

End

(b) Consider the following network with the indicated link costs. Use `Dijkstra's` shortest-path algorithm to compute the shortest path from `node X` to all network nodes. Show how the algorithm works by computing the table for the shortest distances from `node X` to all other nodes. [35%

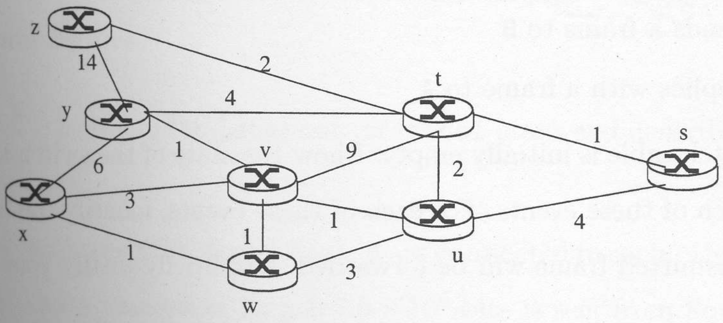

Figure 1: A network with eight nodes

(c) Consider the three node topology shown in the figure below. Compute the distance tables after the initialisation step and after each iteration of a synchronous version of the distance vector algorithm. [35%]

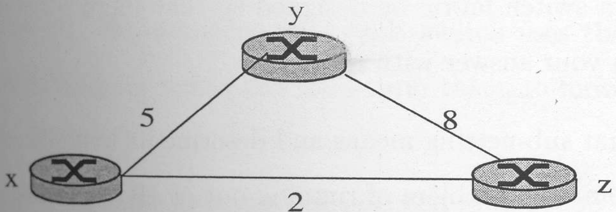

Figure 2: A network with three nodes

(a) Describe the top level architecture of a router. Use diagrams generously to further support your answer. [20%]

b) Describe what is meant by `HOL` blocking with the aid of suitable examples and diagrams. [20%]

c) Let's consider the operation of a self-learning Ethernet switch in the context of

a network in which 6 nodes labelled A through F are star conne
self-learning switch (one node connected to each link). Suppose th

- B sends a frame to E
- E replies with a frame to B
- A sends a frame to B
- B replies with a frame to A

The switch table is initially empty. Show the state of the switch ta after each of these events. For each of these events, identify the lir the transmitted frame will be forwarded, and briefly justify your a

(d) Consider a router with a switch fabric, 2 input ports (A and B) ports (C and D). Suppose the switch fabric operates at 1.5 times

i. If, for some reason, all packets from A are destined to D, and al B are destined to C, can a switch fabric be designed so that th port queuing? Explain your answer with reasons.

ii. Suppose now packets from A and B are randomly destined t D. Can a switch fabric be designed so that there is no input Explain your answer with reasons.

6. (a) Describe what sub-netting means and describe its benefits. Also d helps overcome the problem of running out of IP addresses.

(b) Suppose a class C network 200.138.10.0 has been sub-netted mask of 255.255.255.240. For this network calculate and list information:

i. the number of possible networks.

ii. number of possible hosts in each network.

iii. the full address range of each of these networks.

iv. the usable address range of the first three networks.

v. identify the broadcast addresses for the first three networks.

End