NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

SEMESTER I (2009-2010) EXAMINATION FOR

CS4212 COMPILER DESIGN

Nov/Dec 2009 Time Allowed: **2 Hours**

INSTRUCTIONS TO CANDIDATES

1) This examination paper contains **FIVE** questions and comprises **TWENTY-TWO** printed pages.
2) Answer ALL the questions in the spaces provided, even though you might not have time to finish all questions.
3) This is an **OPEN-BOOK** examination.
4) The maximum mark of this paper is **100%**.
5) The questions are **not** ordered in increasing difficulty.

Matriculation Number : ______________________________

For Examiner's Use Only							
Question	**Maximum**	**Marks**	**Check**	**Question**	**Maximum**	**Marks**	**Check**
1(a(i))	6			Q3(a)	6		
1(a(ii))	6			Q3(b)	6		
1(b(i))	4			4(a(i))	6		
1(b(ii))	4			4(a(ii))	6		
2(a)	6			4(b)	6		
2(b)	8			5(a)	5		
2(c)	8			5(b)	4		
2(d)	4			5(c(i))	2		
2(e)	8			5(c(ii))	5		
Subtotal	**54**			**Subtotal**	**46**		
				Total	**100**		

1) Let's begin with questions on the compiler's front end.
   a) Let $\sum$ be an alphabet containing only **a** and **b**.
      i) **[6%]** Write a **regular definition** (ie., **regular grammar**) that generates strings containing only **a**'s and **b**'s such that there are an **even** number of **a**'s and an **odd** number of **b**'s.

         Ans:

      ii) **[6%]** Draw a **deterministic finite state automaton** corresponding to the regular definition you have just defined.

         Ans:

b) In some programming languages, such as Python, statements are **not separated** by delimiters such as ';'. Instead, **indentation** is used to provide separation and nesting of statements. The following Python program contains a conditional statement which spans across lines (1) – (3). Lines (2) and (3) form the then-branch, and are indented.

```
1)    if (bundle == None ):
2)        print mailman.lastErrorText()
3)        sys.exit()
4)    success = mailman.UnlockComponent("30-day trial")
```

i) **[4%]** What are the **possible complications** such practice of using indentation can bring to the compiler's front end?

Ans:

ii) **[4%]** Describe how a compiler front end can handle programs such as the above that use indentation to separate and nest statements.

Ans:

2) Consider the following context-free grammar:

(1) $S \rightarrow U$

(2) $U \rightarrow T$ **a** U

(3) $U \rightarrow T$ **a** T

(4) $T \rightarrow$ **a** T **b** T

(5) $T \rightarrow$ **b** T **a** T

(6) $T \rightarrow \varepsilon$

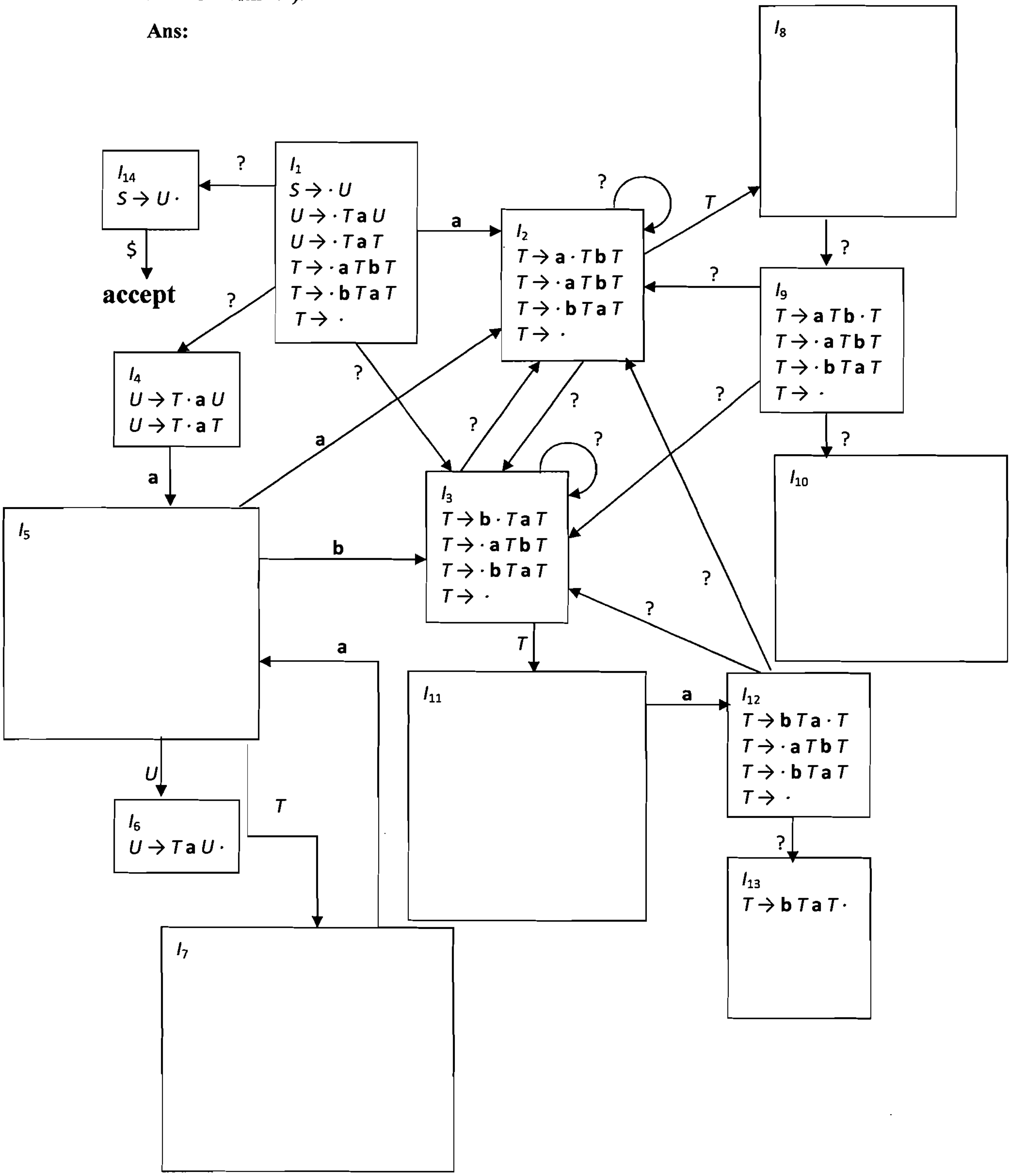
(a) **[6%]** Generate the ***FIRST*** and ***FOLLOW*** **sets** for the non-terminals.

Ans:

	FIRST	*FOLLOW*
S		
U		
T		

(b) **[8%]** Part of a **LR(0)** automaton is shown in the figure below, where I_1 is the initial state. Using the canonical LR(0) collection of items, complete the figure by filling in the missing items in **5 of the states**, and by filling the missing labels on **13 of the transitions (those marked with '?')**.

Ans:



(c) **[8%]** Complete the **parse table** below from the DFA built in part (b) above.

Ans:

	a	b	$	*S*	*U*	*T*
1	s2,r6		r6			
2			r6			8
3			r6			11
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						

(d) **[4%]** The grammar defined in part (b) specifies a language where each string has more **a**'s than **b**'s. There exist some **shift/reduce conflicts** in the parse table you have just built. One such conflict is in state 1 on input symbol '**a**'. **Provide an (informal) argument** whether a shift **or** a reduce should be favoured in this conflict. (Tip: use examples to help your argument; no proof required.)

Ans:

(e) **[8%]** Consider the following grammar and its corresponding shift-reduce parsing table:

0: $S' \rightarrow S\,\$$

1: $S \rightarrow \mathbf{a}\ S\ \mathbf{b}\ \mathbf{b}$

2: $S \rightarrow \mathbf{c}\ S$

3: $S \rightarrow \varepsilon$

	a	**b**	**c**	$	*S*
0	s1	r3	s2	r3	3
1	s1	r3	s2	r3	4
2	s1	r3	s2	r3	5
3				acc	
4		s6			
5	r2	r2	r2	r2	
6		s7			
7	r1	r1	r1	r1	

Trace the parsing steps for the input string **acacbbbb.** For each step, you should show the current stack content, the remaining input string, and the next action. **Is the string accepted**?

Ans:

Stack	Input	Action
0	**acacbbbb**$	

3) Part of the syntax-directed definitions (**SDD**) for translating a Boolean expression involving the AND operator (&&) is given in the table below.

   a) **[6%]** Using the same set of attributes already provided in the first rule, **complete the missing (second) rule** for the expression involving the **IMPLY operator (=>)**, which has the following meaning: *P* => *Q* is true if and only if either *P* is false, or both *P* and *Q* are true.

Ans:

Production	**Semantic Rules**
$B \rightarrow$ true	*B.code = gen(* '**goto**' *B.true)*
$B \rightarrow$ false	*B.code = gen(* '**goto**' *B.false)*
$B \rightarrow B_1$ && B_2	$B_1.true = newlabel()$ $B_1.false = B.false$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \parallel label(B_1.true) \parallel B_2.code$
$B \rightarrow B_1 \Rightarrow B_2$	

b) **[6%]** Repeat (a) for the expression involving the **exclusive-OR operator (^^)**, which evaluates to true if **exactly one** of its two operands is evaluated to true. (Tip: you can create new local temporary variables by calling *newtemp()*.)

Ans:

Production	Semantic Rules
$B \rightarrow B_1$ ^^ B_2	

4) This question raises some issues pertaining to run-time environments.

a) (You should review both the following sub-questions (i) and (ii) together in order to better prepare your answers.) Consider the following function declaration written in a language that treats **functions as first-class citizens.**

```
1.    function first(): int
2.       var x: int
3.       var f: (void -> int)
4.       function second() : int
5.         var x : int
6.         function third(a: int) : int
7.            { return (x+a) }
8.         {
9.          x = 5 ;
10.         return third
11.        }
12.      {
13.        x = 10 ;
14.        f = second() ;
15.        return f(3)
16.      }
```

The declaration above declares three functions (presented in different fonts), **`first`**, **second** and `third`, with the latter function declaration nested within the declaration of the former. The call **`first( )`** at line 14 will invoke the call **second( )**, which returns the function `third`. Line 15 invokes function `third` with argument 3. Under static scoping rules, this will evaluate (x+a) in line 6 to integer 8, and not 13. Thus, **`first( )`** returns integer 8.

i) **[6%]** Describe how you will prepare the activation record for `third` when it is returned as a value at line 10 so that `third` can be called and executed correctly at line 15.

Ans:

ii) **[6%]** The function declaration is repeated here for easy reference.

```
(1)   function first(): int
(2)       var x: int
(3)       var f: (void -> int)
(4)       function second() : int
(5)         var x : int
(6)         function third(a: int) : int
(7)            { return (x+a) }
(8)         {
(9)          x = 5 ;
(10)         return third
(11)        }
(12)      {
(13)        x = 10 ;
(14)        f = second() ;
(15)        return f(3)
(16)      }
```

Show in diagram, and explain in words, **the content of the control stack** at the point when call is made at line 15 and the control is in the body (line 7) of `third`.

Ans:

<Extra Workspace>

b) **[6%]** List down the circumstances under which a compiler designer will choose to **ignore** the generation and storage of **access links** in the activation records during the construction of a compiler.

Ans:

5) Consider a hypothetical target machine with two general-purpose register R0 and R1. It has **two-address** instructions of the form: ***op*** ***source, destination***

in which ***op*** is an op-code, and ***source*** and ***destination*** are data fields. It has the following op-codes (among others):

MOV (move *source* to *destination*)

ADD (add *source* to *destination*: *destination* = *store*+ *destination*)

SUB (subtract *source* from *destination*: *destination* = *destination* – *source*)

While both the ***source*** and ***destination*** can be either memory location or register, or even constant, it is desirable that **only MOV operation involves memory location, and memory access is used in arithmetic operation only when register spilling is required.**

A ***source*** can also be a constant *c*, which is expressed as **#c**

In addition to these instruction forms, the target machine also has an unconditional and a conditional jump instruction:

GOTO *label* (Jump to *label*)

BGTZ *register, label* (if content(*register*) > 0 jump to *label*)

a) **[5%]** In the following (possibly faulty) machine code sequence generated from the intermediate code, complete the associated **register and address descriptor entries** as code is being generated.

Ans:

R0	R1	a	b	C	d	t	u	v
Empty	Empty	{a}	{b}	{c}	Empty	Empty	Empty	Empty

Intermediate instruction sequence	Code Sequence Generated

t := a - b	MOV a, R0
	MOV b, R1
	SUB R1, R0

u := b + c	ADD c, R1

v := t - u	SUB R1, R0

d := v - u	SUB R1, R0 MOV R0, d

b) **[4%]** Show the result of **liveness analysis** at each line in the following code segment (from line 1 to line 5), given that **only variable d** will be used after the code (This is indicated after line 5 as the set **{ d }**, and before line 6.).

Ans:

```
1.        d = 0 ;

2.        e = a ;

     do {
3.        d = a + d ;

4.        e = e -  b ;

       }
5.        while (e > 0)

          { d }
6.        return d ;
```

c) The code segment appearing in part (b) is reproduced here.

```
1.          d = 0 ;
2.          e = a ;
      do {
3.          d = a + d ;
4.          e = e - b ;
      }
5.          while (e > 0)
6.          return d ;
```

This question relates to code generation of the code segment above, given the following conditions:

(a) There are only three registers available for use: R0, R1 and R2.

(b) Variables a and b are initially kept in registers R0 and R1 respectively.

(c) The statement "**return d**" will be translated to storing the final value of d in R2.

i) **[2%]** Which **variables need to be spilled** during your code generation?

Ans:

ii) **[5%] Show and explain the code generated.**

Ans:

--------------------- Extra Workspace --------------

--- END OF PAPER ---