$Id: cmps104a-2011q4-exam2.mm,v 1.33 2011-11-10 16:20:44-08 - - $

page 1 page 2 page 3

Total / 31

Please print clearly:

Name: SOLUTION

Login: @ucsc.edu

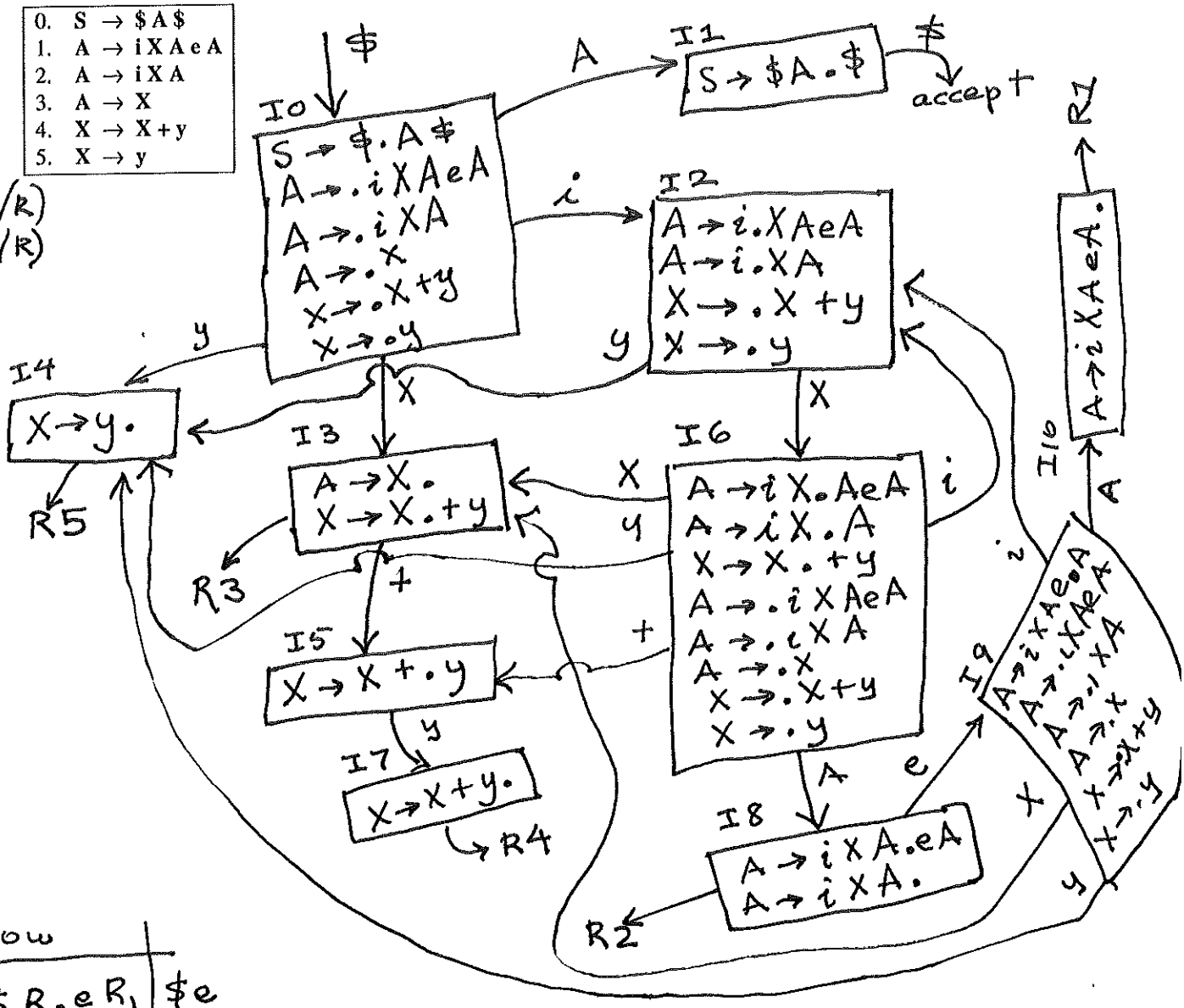
No books; No calculator; No computer; No email; No internet; No notes; No phone. Neatness counts! Do your scratch work elsewhere and enter only your final answer into the spaces provided.

1. Given the grammar presented here, and using the style from the LALR(1) handout:
   (a) Construct the characteristic finite state machine (CFSM), sets of items and transition diagram, showing shifts, reductions, and acceptance. **[6✓]**
   (b) Construct the FOLLOW sets. **[3✓]**
   (c) Answer *yes* or *no* to each of the following questions: **[1✓]**

   Is the grammar LR(0)? NO Is the grammar SLR(1)? NO

0.	S → A
1.	A → iXAeA
2.	A → iXA
3.	A → X
4.	X → X+y
5.	X → y

LR(0)
State 3 (S/R)
State 8 (S/R)



Follow	
A	$ R_2 e R_1	$e
X	R_3 + y i	+ y i $e

State 3 R_3 { $e } ✓ OK is SLR(1)
State 8 R_2 { $e } X SR on (e)

2. Using **bison**, define a grammar for **expr**. An **expr** is a pair of **exprs** connected by one of the operators '+', '-', '*', or '/', or may be an **IDENT** or an **expr** in between parentheses. The operators '+' and '*' have the same precedence as each other and are right associative. The other operators '-' and '/' have the same precedence as each other but lower then the other two, and are left associative. Provide two answers, each using as few rules as possible. Both must be acceptable to **bison**. Do not show semantic actions.

(a) An unambiguous grammar. [2✓]

```
%token IDENT
%start s

%%
expr : expr '-' term
     | expr '/' term
     | term
     ;
term : fac '+' term
     | fac '*' term
     | fac
     ;
fac : '(' expr ')'
    | IDENT
    ;
%%
```

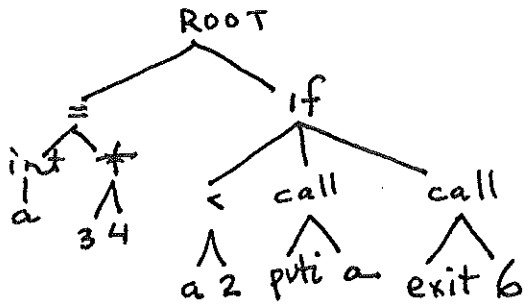
(a) An ambiguous grammar using precedence and associativity declarations. [2✓]

```
%left '-' '/'
%right '+' '*'
%start expr
%token IDENT
%%
expr : expr '+' expr
     | expr '-' expr
     | expr '*' expr
     | expr '/' expr
     | '(' expr ')'
     | IDENT
     ;
%%
```

3. Using the specifications for project 3, draw abstract syntax trees for the following programs. Show only tokens, not lexical information. Draw the ASTs as a tree, not as a data structure, i.e., all lines should be from parent to child, not sibling to sibling.

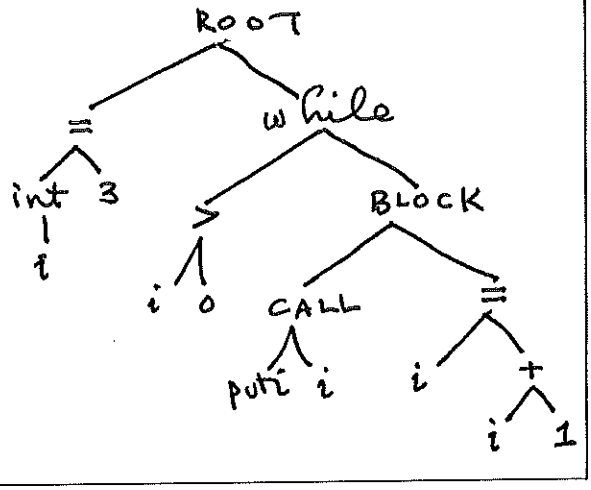
(a) // [2✓]

```
int a = 3 + 4;
if (a < 2) puti (a); else exit (6);
```



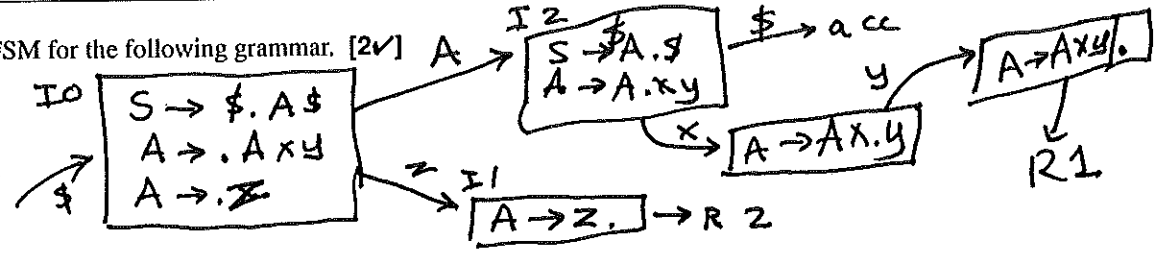
(a) // [2✓]

```
int i = 3;
while (i > 0) { puti (i); i = i + 1; }
```



4. Draw the LR(0) CFSM for the following grammar. [2✓]

0.	S → $ A $
1.	A → A x y
2.	A → z



Multiple choice. To the *left* of each question, write the letter that indicates your answer. Write *Z* if you don't want to risk a wrong answer. Wrong answers are worth negative points. **[11✓]**

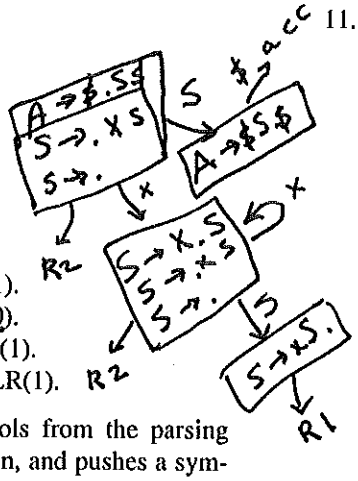
number of correct answers		× 1 =	$= a$
number of wrong answers		× ½ =	$= b$
number of missing answers		× 0 =	0
column total $c = \max(a-b, 0)$	11		$= c$

1. If an LR(0) machine created from a grammar has n states, then the size of the SLR(1) parse table will be: what will be the size of the parse table?
   (A) $(|V_N| \times |V_T|) + n$
   (B) $(|V_N \cup V_T|) \times n$
   (C) $(|V_N \cap V_T|) \times n$
   (D) $|V_N| \times |V_T|^n$

2. This `bison` grammar has:
```
s : a | b ;
a : 'x' ;
b : 'x' ;
```
   (A) a reduce/reduce conflict.
   (B) a shift/reduce conflict.
   (C) a shift/shift conflict.
   (D) no conflict of any kind.

3. Which grammar will recognize one `'x'` followed by zero or more `'y'`s?
   (A) `a : 'x' a | 'y' ;`
   (B) `a : 'y' a | 'x' ;`
   (C) `a : a 'x' | 'y' ;`
   (D) `a : a 'y' | 'x' ;`

4. The following grammar:
   1. $S \rightarrow x\ S$
   2. $S \rightarrow$
   (A) is LR(0) but not SLR(1).
   (B) is SLR(1) but not LR(0).
   (C) is both LR(0) and SLR(1).
   (D) is neither LR(0) nor SLR(1).



5. When a parser pops symbols from the parsing stack, calls a semantic action, and pushes a symbol onto the stack, what kind of action is it?
   (A) accept
   (B) error
   (C) reduce
   (D) shift

6. The `flex` expression `ab|c*` is the same as
   (A) `((ab)|c)*`
   (B) `(a(b|c))*`
   (C) `(ab)|(c*)`
   (D) `a((b|c)*)`

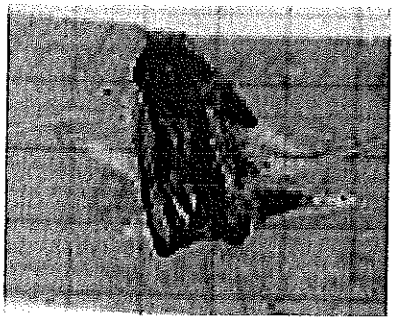
7. When `yylex` has scanned a token, during the execution of the appropriate semantic action, what variable points at the lexeme?
   (A) `yyin`
   (B) `yylval`
   (C) `yyparse`
   (D) `yytext`

8. What is used to convert an NFA into a DFA?
   (A) Hopcroft's construction
   (B) Kleene's construction
   (C) subset construction
   (D) Thompson's construction Reg → NFA

9. Which `flex` pattern will recognize a Java double-slash comment?
   (A) `/\/\[^/n]*` ✗
   (B) `/\/\[^\n]*` ✗
   (C) `\/\/[^/n]*` ✗
   (D) `\/\/[^\n]*`

10. What kind of Chomsky automaton does `bison` simulate?
    (A) Turing machine ✗
    (B) linear-bounded automaton ✓
    (C) pushdown automaton ✓
    (D) finite state automaton ✗

11. The largest class of grammars whose characteristic finite state machines are the same size as the LR(0) CFSM is:
    (A) LR(0)
    (B) SLR(1)
    (C) LALR(1)
    (D) LR(1)



Grace Hopper was working on the Harvard University Mark II Aiken Relay Calculator (a primitive computer). On the 9th of September, 1947, when the machine was experiencing problems, an investigation showed that there was a moth trapped between the points of Relay #70, in Panel F.
[http://en.wikipedia.org/wiki/Grace_Hopper]