CSE 401/M501 24sp Midterm Exam 5/3/24 Sample Solution

Question 1. (18 points) Regular expressions and DFAs. *Gaussian integers* are complex numbers whose real and imaginary parts are integers. They have the same form as ordinary complex numbers *a*+*bi*. Integers *a* and *b* are decimal integers and *i* is a symbol that represents the square root of -1. More precisely, for this problem, Gaussian integers are strings *a*+*bi* with the following properties:

- *a* and *b* are non-negative decimal integers containing one or more decimal digits 0 through 9, and they can start with any digit including 0.
- A number can have both a real (*a*) and an imaginary (*bi*) part or just a real part or just an imaginary part. If it has both parts, the real part must come first. If it has only one part, the + symbol connecting the parts is omitted.
- If both the real and imaginary parts are present, they are connected by the symbol +. (To simplify the problem, it is not possible to use – to connect the parts.)
- The coefficient *b* of the imaginary term can be omitted. The symbol *i* by itself means the same as 1*i*.

Examples of legal Gaussian integers using these rules: 3+14i, 0078+01i, 3+i, 42, 17i, 0+17i, i, 0, 0i, 0+0i, 3+0i .

Examples of strings that are not Gaussian integers using these rules: +1+2i (leading + symbol), 5-6i (- symbol), -5 (leading -), +i (+ sign used when no real part), 3+4 (multiple real parts), 3i+6i (multiple imaginary parts).

(continued

on

next

page)

⇩

Question 1 (cont.) (a) (8 points) Give a regular expression that generates all strings that are legal Gaussian integers according to the rules on the previous page.

Fine print: You must restrict yourself to the basic regular expression operations covered in class and on homework assignments: *rs* , *r*|*s* , *r** , *r+* , *r*?, character classes like `[a-cxy]` and `[^aeiou]`, abbreviations *name*=*regexp*, and parenthesized regular expressions. No additional operations that might be found in the "regexp" packages in various Unix programs, scanner generators like JFlex, or programming language libraries are allowed.

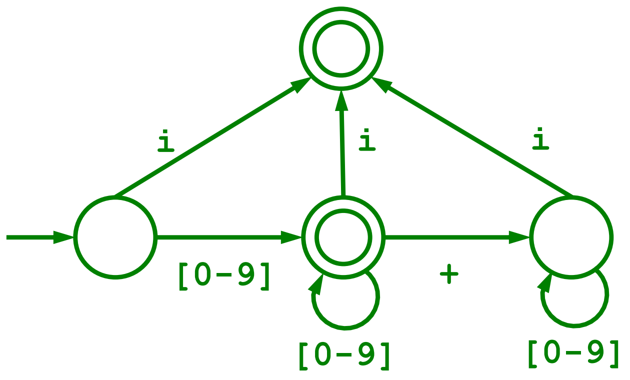
digits = [0-9]+

digits | (digits +)? digits? i

Note: The + in the digits abbreviation above is the regular expression + operator. The one in the (digits +) regular expression is a literal + symbol. As long as your notation was clear it received full credit.

(b) (10 points) Draw a DFA that accepts all valid Gaussian integers described above and generated by the regular expression from part (a).

Here is one possibility:



Question 2. (10 points) Scanners. A scanner ought to be able to process any sort of text input, regardless of whether the file contents are actually a program in the language the scanner was intended to process.

What happens if we use a scanner for MiniJava to process the following input?

```
this is !a#bin@ryString 011101[1101]
```

Below, list in order the tokens that would be returned by a scanner for MiniJava as it reads this input. If there is a *lexical* error in the input, indicate where that error is encountered by writing a short explanation of the error in between the valid tokens that appear before and after the error(s) (something brief like "illegal character %" would be fine). The token list should include any tokens found after any error(s) in the input, i.e., scanning should continue after discovering an error. You may use any reasonable token names (e.g., LPAREN, ID(x), etc.) as long as your meaning is clear.

A copy of the MiniJava grammar is attached as the last page of the test. You should remove it from the exam and use it for reference while you answer this question. You should assume that the scanner processes MiniJava syntax as defined in that grammar, with no extensions or changes to the language. Also recall that the MiniJava project defines an <IDENTIFIER> as a sequence of letters, digits, and underscores, starting with a letter, and uppercase letters are distinguished (different) from lowercase; and an <INTEGER_LITERAL> is a sequence of decimal digits not starting with 0, or the number 0 by itself, denoting a decimal integer value.

THIS ID(is) NOT ID(a)

Illegal character #

ID(bin)

Illegal character @

ID(ryString) INT(0) INT(11101) LBRACK INT(1101) RBRACK

Question 3. (12 points) Ambiguity. We've been hired by the departments of tourism of the Carolinas and the Dakotas to write a grammar to generate slogans to attract visitors to their states. Below is a grammar that our Grammar Committee came up with in response to the request:

```
G ::= Visit A D P | Visit D P
A ::= Spectacular | Wonderful | Scenic
D ::= North | South
P ::= Dakota | Carolina
```

Notes: whitespace in the grammar is only for readability and is not part of the grammar or the strings generated by it. Each of the words like `Wonderful` in a grammar production should be treated as a single terminal symbol, not as individual letters.

(a) (5 points) Give a rightmost derivation of the sentence `Visit Scenic South Dakota`.

***G* => Visit *A D P* => Visit *A D* Dakota => Visit *A* South Dakota =>**

Visit Scenic South Dakota

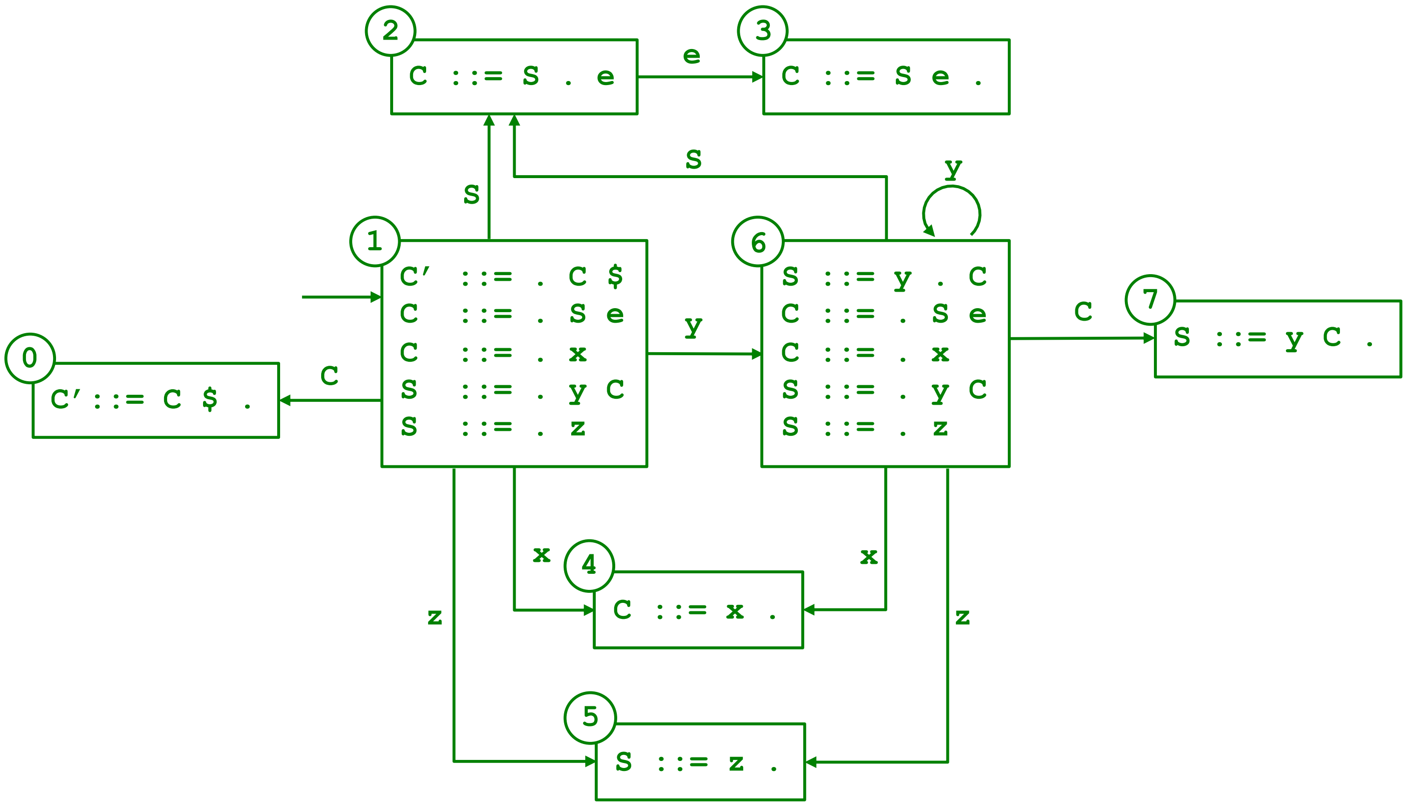
(b) (7 points) Is this grammar ambiguous? If so, give a proof that it is by showing two distinct leftmost derivations for some string generated by the grammar. If not, give an informal, but precise, argument why it is not ambiguous.

No, this grammar is not ambiguous (in fact it is regular). There is only one way to derive each of the possible strings generated by the language.

Question 4. (36 points) The "OMG! Not this again!!" parsing question. Consider the following grammar. The nonterminal *C* is the start symbol of the grammar, and the extra *C'*::=*C* \$ rule needed to handle end-of-file in an LR parser has been added for you.

0. *C'* ::= *C* \$	(\$ is EOF)	3. *S* ::= y *C*
1. *C* ::= *S* e		4. *S* ::= z
2. *C* ::= x		

(a) (16 points) Draw the LR(0) state machine for this grammar. When you finish, you should number the states in the final diagram in whatever order you wish so that you can use the state numbers in later parts of this question. The state numbers should be successive integers starting with 0, 1, 2, 3, ….



(continued on next page)

Question 4. (cont.) Grammar repeated from previous page for reference:

0. $C' ::= C$ `$` (`$` is EOF)
1. $C ::= S$ `e`
2. $C ::=$ `x`
3. $S ::=$ `y` C
4. $S ::=$ `z`

(b) (8 points) Write the LR(0) parser table for the LR parser DFA shown in your answer to part (a). To save time, an empty table is provided below. However, it probably has more rows than you need. Use only as many rows as needed and leave the rest blank.

State #	e	x	y	z	$	*C*	*S*
0					**acc**		
1		**s4**	**s6**	**s5**		**g0**	**g2**
2	**s3**						
3	**r1**	**r1**	**r1**	**r1**	**r1**		
4	**r2**	**r2**	**r2**	**r2**	**r2**		
5	**r4**	**r4**	**r4**	**r4**	**r4**		
6		**s4**	**s6**	**s5**		**g7**	**g2**
7	**r3**	**r3**	**r3**	**r3**	**r3**		
8							
9							
10							
11							
12							
13							
14							
15							

(continued on next page)

Question 4. (cont.) Grammar repeated from previous pages for reference:

0. $C' ::= C$ `$` ($ is EOF)
1. $C ::= S$ `e`
2. $C ::=$ `x`
3. $S ::=$ `y` C
4. $S ::=$ `z`

(c) (3 points) Is this grammar LR(0)? Explain why or why not. Your answer should describe **all** of the problems that exist if the grammar is not LR(0) by identifying the relevant state number(s) in your answers to parts (a) and (b) and the specific issues in those state(s) (i.e., something like "state 47 has a shift-reduce conflict if the next input is `blah`", but with, of course, correct state numbers and details from your parser). If the grammar is LR(0), you should give a technical explanation why it is (this can be brief).

Yes. There are no shift-reduce or reduce-reduce conflicts in any states of the LR(0) state machine (or, equivalently, in the table).

(d) (6 points) Complete the following table showing the FIRST and FOLLOW sets and nullable for each of the nonterminals in this grammar. You should include $ (the end-of-file marker) in the FOLLOW set for any nonterminal where it is appropriate.

	FIRST	FOLLOW	nullable
C	**x, y, z**	**e, $**	**no**
S	**y, z**	**e**	**no**

(e) (3 points) Is this grammar SLR? Give a brief technical explanation why or why not.

Yes. Because the grammar is LR(0) it is also SLR. Or, equivalently, because there are no conflicts in the LR(0) state machine or parse tables, the grammar is SLR.

Question 5. (8 points) Top-down parsing. Take another look at the grammar from the previous problem, but omitting the *C'* ::= *C* \$ rule that was added for LR parsing:

1. *C* ::= *S* `e`	3. *S* ::= `y` *C*
2. *C* ::= `x`	4. *S* ::= `z`

(Recall that *C* is the start symbol for this grammar.)

Is this grammar suitable for constructing a top-down LL(1) predictive parser? If so, explain why. If not, explain why not, and, if possible, construct a different grammar that generates the same language that is suitable for a top-down LL(1) predictive parser, or explain why this can't be done.

Yes, it can be used directly to construct a predictive parser. To verify this, we need to look at the FIRST sets for the right-hand sides of the productions for each non-terminal.

For *C*, FIRST(*S* `e`) is the set { `y`, `z` }. FIRST(`x`) is the set { `x` }. These sets are disjoint so the *C* productions satisfy the necessary criteria.

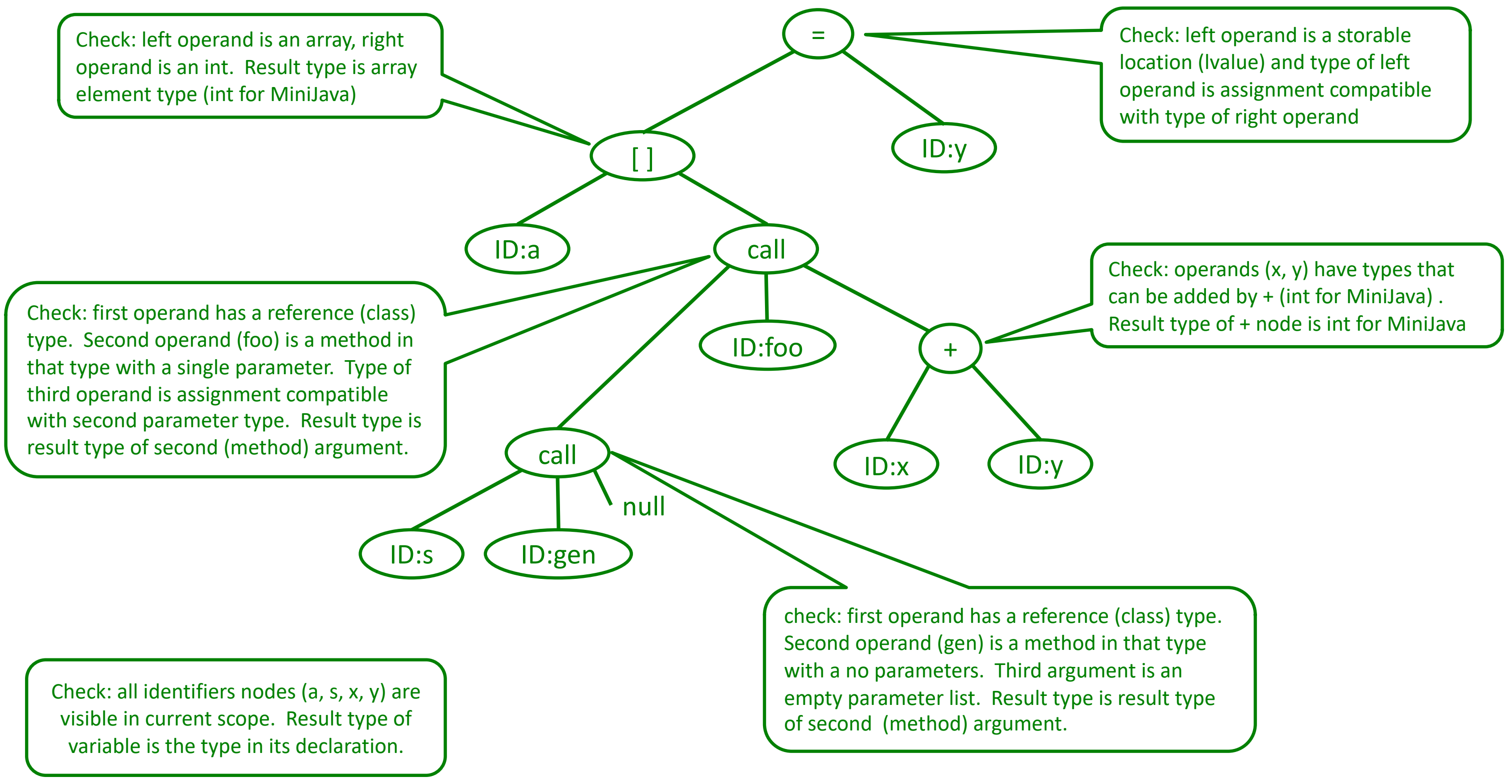
For *S*, FIRST(`y` *C*) is the set { y } and FIRST(`z`) is the set { `z` }. These sets also have no common elements so the *S* productions satisfy the criteria.

Question 6. (14 points) Semantics. Suppose we have the following statement in a MiniJava program:

```
a[s.gen().foo(x+y)] = y;
```

(a) (7 points) At the bottom of this page, draw an abstract syntax tree (AST) for this assignment statement. You should use appropriate names for the AST nodes, and have an appropriate level of abstraction and structural detail similar to the AST nodes in the MiniJava project AST classes, but don't worry about matching the exact names or details of classes or nodes found in the MiniJava starter code.

(b) (7 points) Annotate your AST by writing next to the appropriate nodes the checks or tests that should be done in the static semantics/type-checking phase of the compiler to ensure that this statement does not contain errors. If a particular check or test applies to multiple nodes, you can write it once and indicate which nodes it applies to, as long as your meaning is clear and readable. You may assume that `int` is the only numeric type in MiniJava, but remember that MiniJava also has `boolean` and object (class) types.


=
[ ]
ID:y
ID:a
call
ID:foo
+
ID:x
ID:y
call
null
ID:s
ID:gen
Check: left operand is an array, right operand is an int. Result type is array element type (int for MiniJava)
Check: left operand is a storable location (lvalue) and type of left operand is assignment compatible with type of right operand
Check: operands (x, y) have types that can be added by + (int for MiniJava) . Result type of + node is int for MiniJava
Check: first operand has a reference (class) type. Second operand (foo) is a method in that type with a single parameter. Type of third operand is assignment compatible with second parameter type. Result type is result type of second (method) argument.
check: first operand has a reference (class) type. Second operand (gen) is a method in that type with a no parameters. Third argument is an empty parameter list. Result type is result type of second (method) argument.
Check: all identifiers nodes (a, s, x, y) are visible in current scope. Result type of variable is the type in its declaration.

Question 7. (2 free points) (All reasonable answers receive the points. All answers are reasonable as long as there is an answer. ☺)

(a) (1 point) What question were you expecting to appear on this exam that wasn't included?

<this space intentionally left blank>

(b) (1 points) Should we include that question on the final exam? (circle or fill in)

Yes

No

Heck No!!

[✓ $!@$^*% No !!!!!]

Yes, yes, it *must* be included!!!

No opinion / don't care

None of the above. My answer is ________________________________.