CSE 401/M501 23au Midterm Exam 11/3/23

Name _____ UW netid _____@uw.edu

There are 6 questions worth a total of 100 points. Please budget your time so you get to all of the questions. Keep your answers brief and to the point.

The exam is closed books, closed notes, closed electronics. However, you may have one 5x8 notecard for reference with any hand-written information you wish on both sides. Please turn off all cell phones, personal electronics, alarm watches, and pagers, and return your tray tables and seat backs to their full upright, locked positions. Sound or video recording, the taking of photographs, and time travel are prohibited.

If you have a question during the exam, please raise your hand and someone will come to help you.

There is an extra blank page at the end of the exam you can use if your answer(s) do not fit in the space provided. Please indicate on the original page(s) if your answer(s) is(are) continued on that last page.

After the blank page with extra space for answers is a copy of the MiniJava grammar. You should remove this from the exam and use it for reference as needed.

Please wait to turn the page until everyone is told to begin.

Score _____

1 _____ / 14

2 _____ / 14

3 _____ / 40

4 _____ / 14

5 _____ / 16

6 _____ / 2

Question 1. (14 points) Regular expressions and DFAs. Suppose we define this abbreviation for a regular expression describing a single decimal digit:

digit = [0-9]

Now consider the following regular expression:

digit+ (/ 0* [1-9] digit*)?

(In the above regular expression, / is a literal slash character / . The symbols (,), [,], *, +, −, and ? are regular expression grouping and operator symbols. Note that the entire regular expression after digit+ is optional, i.e., (…)?)

(a) (6 points) Describe the set of strings generated by this regular expression. For full credit you should give a description of the strings, not a description of how the regular expression operators are used to produce them (i.e., don't describe how the regular expression works, just what the set contains).

(b) (8 points) Draw a DFA that accepts the strings generated by this regular expression.

Question 2. (14 points) Scanners. Continuing our exploration of what a MiniJava scanner would do with various kinds of input text, we used our MiniJava scanner to read each of the following four input lines and turn them into tokens. (The lines are numbered for reference. The numbers before each `if` are not part of the input read by the scanner.)

```
1. if (whiletrue) [ x <> 0x2 ] /* $bash */ class[$bash]
2. if (while true) [ x < > 0 x2 ] /* $bash */ class[$bash]
3. if (whiletrue) [ x <> 0 x2 ]class[$bash]
4. if (whiletrue) [ x <> 0 x 2 ] /* $bash */ class[$bash]
```

The MiniJava scanner produced identical token streams for two of these four input lines. The output for the other two input lines produced different token streams.

(a) (4 points) Which two input lines produced the same sets of tokens?

(b) (10 points) Below, list in order the tokens that are returned by a MiniJava scanner for one of the two input lines that had the same output (i.e., what tokens were returned for the input lines that matched, but you only should write that sequence once). If there are any characters in the input line that produced an error, you should write ERROR(#) in the token stream to show where the illegal character was encountered, using the actual character instead of # of course. Your scanner output should include all tokens and errors present – i.e., don't stop when (or if) the first error is encountered. 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.

Question 3. (40 points) The "well, I guess we can't say we weren't expecting it" parsing question. Consider the following grammar. The extra *E'*::=*E* $ rule needed to handle end-of-file in an LR parser has been added for you.

0. *E'* ::= *E* `$` (`$` is EOF)
1. *E* ::= `a` *B*
2. *E* ::= `c`
3. *B* ::= *B* `y` *E*
4. *B* ::= `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, but you should number the states starting with 0, 1, 2, 3, ….

(continued on next page)

Question 3. (cont.) Grammar repeated from previous page for reference:

0. E' ::= E `$` (`$` is EOF)
1. E ::= `a` B
2. E ::= `c`
3. B ::= B `y` E
4. B ::= `x`

(b) (12 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 #	a	c	x	y	$	*E*	*B*
0							
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							

(continued on next page)

Question 3. (cont.) Grammar repeated from previous page for reference:

0. *E′* ::= *E* `$` (`$` is EOF)
1. *E* ::= `a` *B*
2. *E* ::= `c`
3. *B* ::= *B* `y` *E*
4. *B* ::= `x`

(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 `foo`", but with, of course, state numbers and correct details from your parser). If the grammar is LR(0), you should explain why (this can be brief).

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

	FIRST	FOLLOW	nullable
E			
B			

(e) (3 points) Is this grammar SLR? Explain why or why not.

Question 4. (14 points) Top-down parsing. Take another look at the grammar from the previous problem, but omitting the *E'* ::= *E* $ rule that was added for LR parsing:

1. *E* ::= `a` *B*
2. *E* ::= `c`
3. *B* ::= *B* `y` *E*
4. *B* ::= `x`

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 for the same language that is suitable for a top-down LL(1) predictive parser, or explain why this can't be done.

Question 5. (16 points) Semantics. One of our big customers would like to add a `?:` operator to MiniJava, just like the one in C and C++. The meaning of the expression *e1* `?` *e2* `:` *e3*, is that expression *e1* is evaluated first. If *e1* evaluates to true, the *e2* is evaluated and that is the value of the entire `?:` expression. If *e1* is false, then *e3* is evaluated, and that is the value of the expression. For example, this assignment statement will store the smaller of two values `x` and `y` in variable `min`:

```
min = x < y ? x : y ;
```

(a) (8 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 code.

(b) (8 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.

Question 6. (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?

(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 _____.

Extra space for answers, if needed. Please be sure to label which question(s) are answered here, and be sure to put a note on the question page so the grader will know to look here.