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

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:

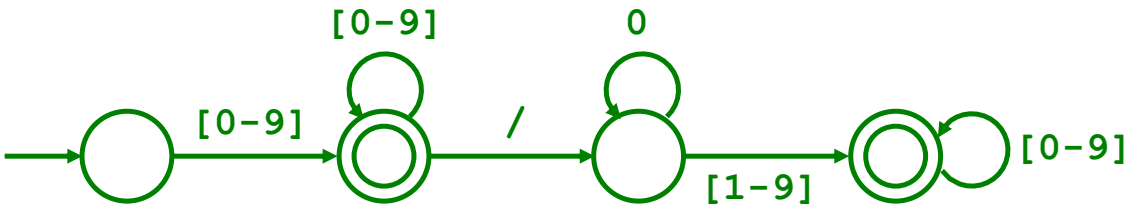
digit+ (`/ 0*` `[1-9]` digit*)?

(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).

The generated strings are unsigned decimal rational numbers *num*/*denom*, where *num* is a sequence of one or more decimal digits (including 0), and *denom* is any sequence of one or more decimal digits except for the number 0 by itself. The "/ *denom*" part is optional.

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



Note: There are many possible solutions to this problem, of course. Other DFAs that accept the same set of strings also received full credit.

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?

1 and 3

(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.

IF LPAREN ID(whiletrue) RPAREN LBRACK ID(x) LESS ERROR(>)

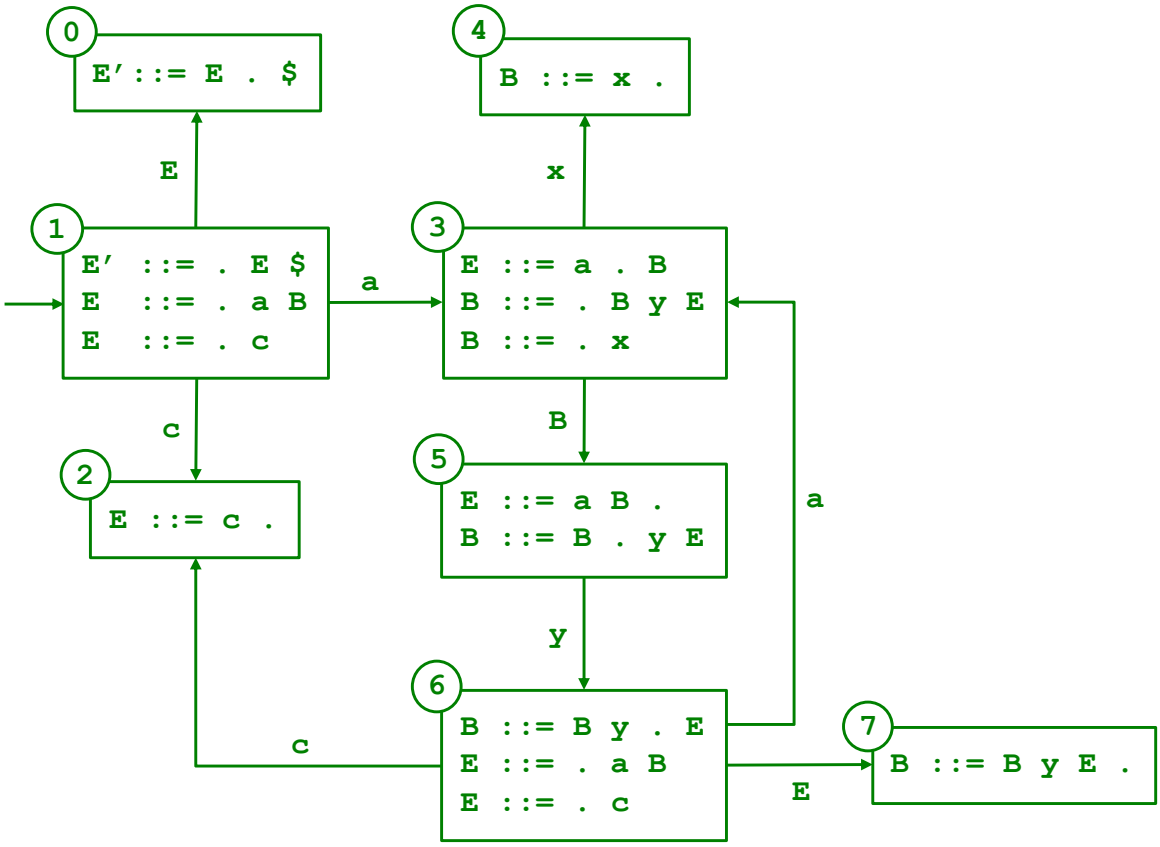
INTEGER(0) ID(x2) RBRACK CLASS LBRACK ERROR($) ID(bash)

RBRACK

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					**acc**		
1	**s3**	**s2**				**g0**	
2	**r2**	**r2**	**r2**	**r2**	**r2**		
3			**s4**				**g5**
4	**r4**	**r4**	**r4**	**r4**	**r4**		
5	**r1**	**r1**	**r1**	**r1, s6**	**r1**		
6	**s3**	**s2**				**g7**	
7	**r3**	**r3**	**r3**	**r3**	**r3**		
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).

No. There is a shift-reduce conflict in state 5 on input `y`.

(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	**a, c**	**y, $**	**no**
B	**x**	**y, $**	**no**

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

No. Since `y` is in FOLLOW(*E*), we cannot eliminate the shift-reduce conflict in state 5. A reduce is still appropriate in that state because of the follow set information.

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.

No. Rule 3 has a direct left recursion on the non-terminal *B*.

We should be able fix this with the standard algorithm for handling direct left recursions. Replace rules 3 and 4 from the original grammar with the following two rules:

B* ::= `x` *bTail
***bTail* ::= `y` *E bTail* | ε**

Unfortunately, this grammar turned out to be trickier than we had expected. Because *bTail* is nullable and `y` is in its follow set, the grammar rules for *bTail* do not satisfy the LL(1) condition. There is no easy set of grammar transformations to fix the problem, although it's not too hard to come up with a suitable, but completely different, LL(1) grammar that generates the language.

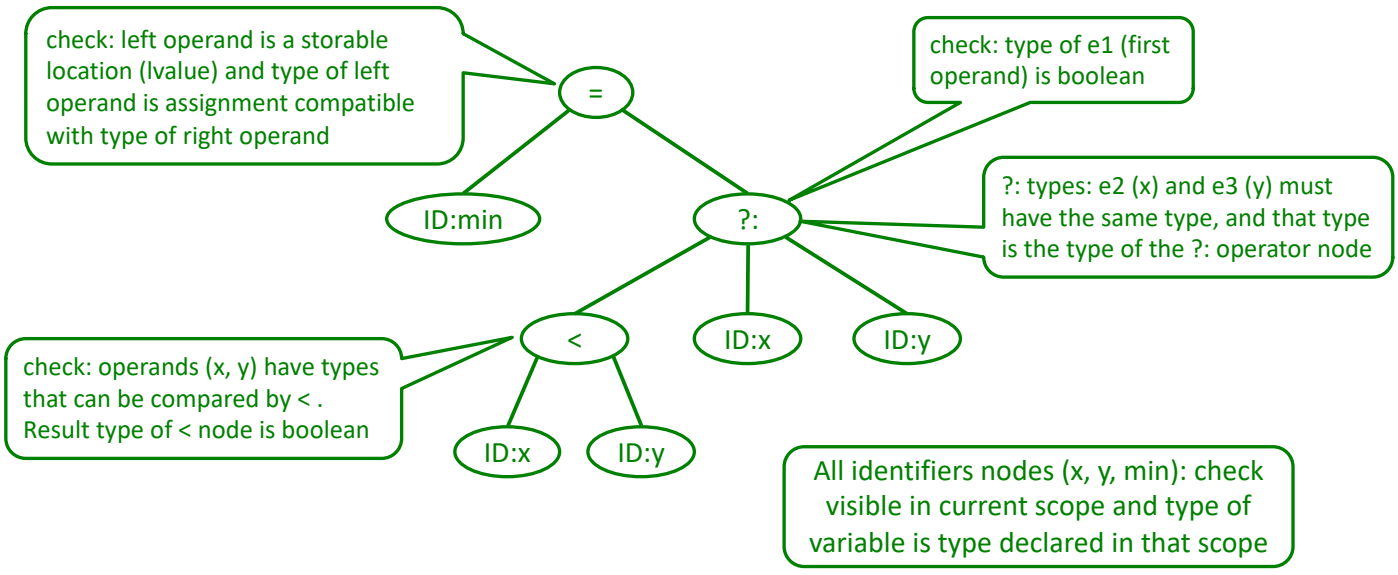
In the end, when grading the question, we gave full credit to answers that recognized the problem with the original grammar rule 3 and explained how the standard direct left recursion rule would be the correct approach to try to solve the problem.

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 `7oolean` and object (class) types.



Notes: Because the only MiniJava type that can be used with the < operator is `int`, we gave full credit for answers that restricted the identifier types to `int`. The above type-checking rules for `?:` are more general and handle situations where other types are possible.

Because Java requires all language constructs to have types that can be known at compile time, the `?:` operator requires that the two expressions *e2* and *e3* have the same type, and that type is the result type of `?:`.

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?

All answers received full credit

(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 _____.