CSE 401 Midterm Exam Sample Solution 11/4/11

Question 1. (12 points, 2 each) The front end of a compiler consists of three parts: scanner, parser, and (static) semantics. Collectively these need to analyze the input program and decide if it is correctly formed. For each of the following properties or errors, indicate which stage of the front-end of the compiler would normally handle it. If it helps to explain your answer you can give a brief reason why, but that is not required.

(a) In a variable declaration, the variable has not previously been declared in the same scope.

semantics

(b) += is not a legal assignment operator in MiniJava.

parser (+ and = are both legal tokens, so the scanner won't detect the error)

(c) A comment beginning with `/*` is not terminated before reaching the end of the input file (i.e., no matching `*/` found).

scanner

(e) In the method call `x.m(e1,e2,…,en)`, the type of x includes a suitable method `m`.

semantics

(e) Class declarations cannot be nested in MiniJava.

parser (the grammar does not include nested class declarations)

(f) The € character cannot be used in an identifier in MiniJava.

scanner

Question 2. Regular expressions (12 points, 6 each) Give regular expressions for the following sets of strings. You may only use the basic operations of concatenation, choice (|), and repetition (*), plus the simple extensions ? and +, and character sets like [a-z] and [^a-z]. You may also give names to subexpressions like *vowels* = [aeiou].

(a) All strings of 0's and 1's that have at least one character and start and end with the same character. (Note that the strings 0 and 1 by themselves are in this set, as well as longer strings of 0's and 1's that have the same character at the beginning and end.)

Here are two possibilities; there are many others:

0 | 1 | 0(0|1)*0 | 1(0|1)*1 **0((0|1)*0)? | 1((0|1)*1)?**

(b) All strings of a's, b's and c's that (i) have at least one c, and (ii) all of the a's (if any) are to the left of all of the b's (if any).

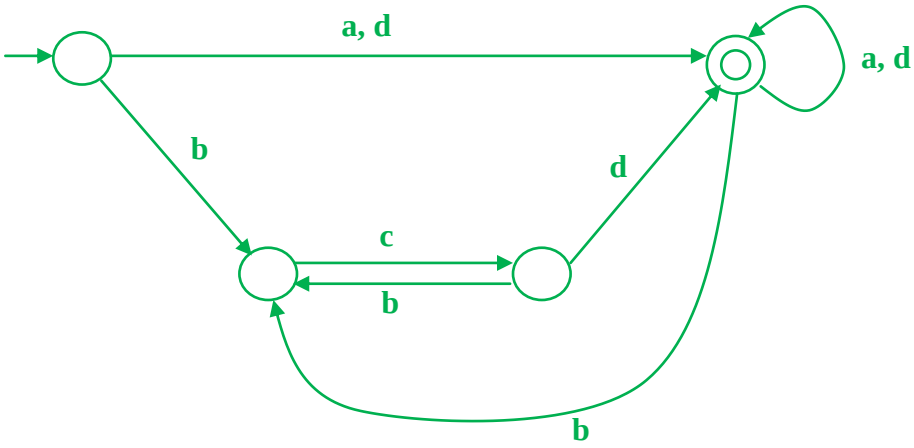
Two example solutions. For both, let acs = (a|c)* and bcs = (b|c)*.

Solution 1: acs bcs c | acs c bcs | c acs bcs

Solution 2: acs c acs bcs | acs bcs c bcs

Question 3. DFAs. (8 points) Draw a DFA that accepts strings generated by the regular expression (a | (bc)* d)+

Note that (bc)* means empty or bc or bcbc or bcbcbc…, but not an arbitrary sequence of b's and c's. That would be (b|c)*.

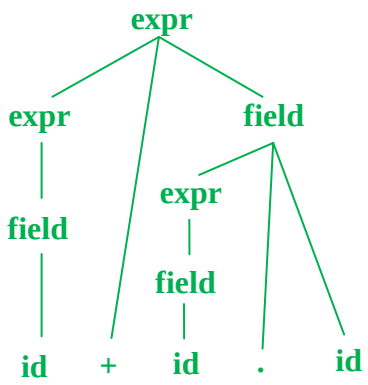


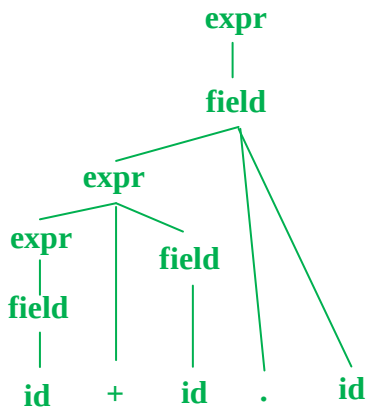
Question 4. Context-free grammars (14 points) Consider the following syntax for expressions involving addition and field selection:

expr ::= *expr* + *field*
expr ::= *field*
field ::= *expr* `.` `id`
field ::= `id`

(a) (8 points) Show that this grammar is ambiguous.

Here are two derivations of id+id.id:





(b) (6 points) Give an unambiguous context-free grammar that fixes the problem(s) with the grammar in part (a) and generates expressions with `id`, field selection and addition. As in Java, field selection should have higher precedence than addition and both field selection and addition should be left-associative (i.e., a+b+c means (a+b)+c).

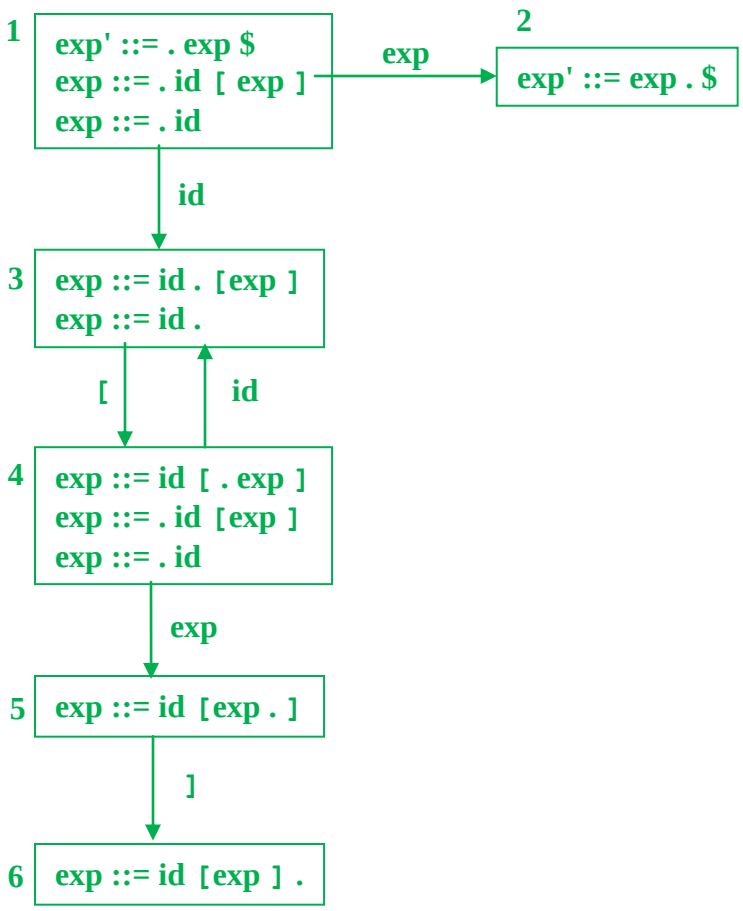
The problem is in the first rule for *field*, which creates an ambiguous precedence. Here is a reasonably simple fix.

expr* ::= *expr* + *field
expr* ::= *field
***field* ::= *field* `.` `id`**
***field* ::= `id`**

Question 5. (26 points) The oh-no-not-again LR-parsing question. Here is a tiny grammar.

0. *exp'* ::= *exp* $
1. *exp* ::= `id [` *exp* `]`
2. *exp* ::= `id`

(a) (11 points) Draw the LR(0) state machine for this grammar.



(continued on next page)

Question 5. (cont.) Grammar repeated from previous page for reference.

0. *exp′* ::= *exp* $
1. *exp* ::= `id [` *exp* `]`
2. *exp* ::= `id`

(b) (9 points) Construct the LR(0) parse table for this grammar based on the state machine in your answer to part (a).

State	id	[	]	$	exp
1	s3				g2
2				acc	
3	r2	r2, s4	r2	r2	
4	s3				g5
5			s6		
6	r1	r1	r1	r1	

(b) (3 points) Is this grammar LR(0)? Why or why not?

No. There is a shift/reduce conflict in state 3 on input `[`.

(c) (3 points) Is this grammar SLR? Why or why not?

Yes. `[` is not in FOLLOW(exp) so we can eliminate the conflict in state 3.

Question 6. Project (14 points) Suppose we want to add a do-while loop to MiniJava like the one found in Java, C, C++, and other languages. The syntax of do-while is

`do` *Statement* `while (` *Expression* `)`

The semantics of do-while is that the *Statement* is executed, then the *Expression* is evaluated, then, if the *Expression* is true, the process repeats again until the *Expression* eventually evaluates to false.

(a) (3 points) What changes need to be made to the MiniJava scanner if this statement is added? You do not need to give specific JFlex or CUP code, just describe the needed changes, if any.

- **Add DO as a new token**
- **Add a rule to JFlex to recognize "do" as a new keyword and return this new token.**

(b) (3 points) What changes or additions (if any) need to be made to the MiniJava AST classes to add the do-while statement? Again, you don't need to give specific code, but describe your changes specifically enough that anyone familiar with the project could make the changes mechanically.

- **Add a new DoStatement AST node class extending Statement with instance variables for the expression and statement that make up the loop.**

(c) (8 points) Give the code that needs to be added to the grammar part of the CUP specification for *Statement* to successfully parse the new do-while statement, including appropriate semantic action code to create appropriate part(s) of the AST.

```
Statement ::= DO Statement:s WHILE LPAREN Expression:e RPAREN
                        {: RESULT = new DoStatement(s,e,sleft); :}
```

No points were deducted if you did not include the line number.

Question 7. (14 points, 7 each) Suppose we encounter the following statement in a MiniJava program:

```
foo = x.m(a[i]);
```

(a) Draw a tree below showing the abstract syntax for this statement. Don't worry about whether you match the AST classes in the MiniJava project exactly (you're not expected to memorize that sort of thing), but be sure that you have the appropriate nodes and edges to include the information that belongs in the abstract syntax.

(b) After you've drawn the AST for this statement, annotate it by writing next to the appropriate nodes the checks or tests that need to be done in the static semantics/type-checking phase of the compiler to ensure that this statement does not contain any errors.