Analysis of the Given Grammar

(February 18, 2016)

COLORS in this document
Black: for original rules in the grammar in Language Specification document
Red: specifies needs for modifications
Green: rules do not require modification

Grammar:

The nonterminal <program> is the start symbol of the given grammar.

1. **<program>** ===> <otherFunctions> <mainFunction>
   The LL(1) property for the nonterminal <program> is satisfied trivially due to single production for it. The following FOLLOW set will be required for verifying LL(1) compatibility for the nonterminal <otherFunctions> in (3).
   FOLLOW(<otherFunctions>) = FIRST(<mainFunction>) = {TK_MAIN}
   Note that all function definitions precede the main function definition and the language does not have constructs for function prototype declarations.
2. <mainFunction>===> TK_MAIN <stmts> TK_END
   The LL(1) property for the nonterminal <mainFunction> is satisfied trivially due to single production for it.
   FIRST(<mainFunction>) = FIRST(α) = {TK_MAIN}
   where α represents the right hand side of the production i.e. TK_MAIN <stmts> TK_END.
3. <otherFunctions>===> <function><otherFunctions> | eps
   The nonterminal <otherFunctions> need special care for verifying the LL(1) compatibility due to a nullable production. Let us first verify that the sets FIRST(<function><otherFunctions>) and FOLLOW<otherFunctions>) are disjoint.
   FIRST(<function><otherFunctions>) = FIRST(<function>) = {TK_FUNID}
   Note that <function> has no nullable production.

Also from 1, we have FOLLOW(<otherFunctions>) = {TK_MAIN}
i.e.
FIRST(<function><otherFunctions) $\cap$ FOLLOW(<otherFunctions>) = ϕ
The given productions for <otherFunctions> are LL(1) compatible.
NOTE: Other properties such as ambiguity, left recursiveness, left factoring needs etc. causing violation of LL(1) compatibility of the rules will be discussed only if one or more of them exist. Otherwise I am focussing on violations due to epsilon productions. Also I will highlight the introduction of new nonterminals to incorporate the precedence of arithmetic operators and handling operations on record variables.

4. <function>===>TK_FUNID <input_par> <output_par> TK_SEM <stmts> TK_END
   This has no issue of violations in LL(1) compatibility due to epsilon productions. <input_par> is an essential construct to be part of the function definition while a function may or may not return values. Hence the <output_par> can have a syntax of (6) .
   FIRST(<function>) = {TK_FUNID}
5. <input_par>===>TK_INPUT TK_PARAMETER TK_LIST TK_SQL <parameter_list> TK_SQR
   Single production having no conflict has its FIRST set as given follows:
   FIRST(<input_par>) = {TK_INPUT}
6. <output_par>===>TK_OUTPUT TK_PARAMETER TK_LIST TK_SQL <parameter_list> TK_SQR | eps
   Presence of epsilon production makes us verify whether FIRST(α) $\cap$ FOLLOW(<output_par>) = ϕ or not, where α represents the right hand side
   TK_OUTPUT TK_PARAMETER TK_LIST TK_SQL <parameter_list> TK_SQR
   FIRST(α) = {TK_OUTPUT}
   Refer rule 4 to compute the FOLLOW(<output_par>) as below
   FOLLOW(<output_par>) = {TK_SEM}
   This implies that FIRST(α) $\cap$ FOLLOW(<output_par>) = ϕ
   Hence the given rules for <output_par> conform to the LL(1) specifications.
7. <parameter_list>===><dataType> TK_ID <remaining_list>
   The single production for <parameter_list> is trivially LL(1) compatible.
   FIRST(<parameter_list>) = FIRST(<dataType>)

= FIRST(<primitiveDatatype>) $\cup$ FIRST(<constructedDatatype>)
= {TK_INT, TK_REAL, TK_RECORD}

8. <dataType>===> <primitiveDatatype> |<constructedDatatype>
   FIRST(<primitiveDatatype>) $\cap$ FIRST(<constructedDatatype>)
   = {TK_INT, TK_REAL} $\cap$ { TK_RECORD}
   = ϕ
   Since there is no nullable production for <dataType>, there is no need for computing FOLLOW(<dataType>).

9. <primitiveDatatype>===> TK_INT | TK_REAL
   FIRST(TK_INT) $\cap$ FIRST(TK_REAL) = {TK_INT} $\cap$ {TK_REAL} = ϕ
   Hence the rules for <primitiveDataType> are LL(1) compatible.
   FIRST(<primitiveDatatype>) = {TK_INT, TK_REAL}
10. <constructedDatatype>===>TK_RECORD TK_RECORDID
   FIRST(<constructedDatatype>) = {TK_RECORD}
11. <remaining_list>===>TK_COMMA <parameter_list> | eps
   There exist a nullable production for the non-terminal <remaining_list>. The RHS (right hand side) of the other production is such that the null string eps (epsilon) is not derivable from it.
   Also
   FIRST(TK_COMMA <parameter_list>) $\cap$ FOLLOW(<remaining_list>) should be empty.
   FIRST(TK_COMMA <parameter_list>) = {TK_COMMA}
   and
   FOLLOW(<remaining_list>) = FOLLOW(parameter_list) .....from (7)
   = {TK_SQR}
   Hence,

FIRST(TK_COMMA <parameter_list>) ∩ FOLLOW(<remaining_list>)
= {TK_COMMA}∩ {TK_SQR}
= ϕ
Therefore both the given rules for the non-terminal <remaining_list> conform to LL(1) specifications.

12. <stmts>===><typeDefinitions> <declarations> <otherStmts><returnStmt>
The nonterminal <stmts> specifies the grammar for the body of the function. The ordering of other nonterminals such as <typeDefinictions> , <declarations> and <returnStmt> have fixed positions within the body of the function code.
There are no epsilon productions and a single non nullable production for <stmts> is LL(1) compatible.
FIRST(<stmts>) = FIRST(<typeDefinitions>) ∪ FIRST(<declarations>) ∪ FIRST(<otherStatements>) ∪ FIRST(<returnStmt>)
. .....using 13, 18,21 and 42
= {TK_RECORD} ∪ {TK_TYPE} ∪ { TK_ID, TK_RECORD_ID, TK_WHILE, TK_IF, TK_READ, TK_WRITE, TK_SQL, TK_CALL}∪ {TK_RETURN}
={ TK_RECORD, TK_TYPE, TK_ID, TK_RECORD_ID, TK_WHILE, TK_IF, TK_READ, TK_WRITE, TK_SQL, TK_CALL,TK_RETURN}
Since the nonterminal <stmts> does not have a nullable production, we need not compute FOLLOW(<stmts>) for populating the parsing table.
Recall the construction of the parsing table. The columns corresponding to the tokens in FIRST(<stmts>) corresponding to the row<stmts> are populated with the rule number 12.

13. <typeDefinitions>===><typeDefinition><typeDefinitions> |eps
The first of the above productions is not nullable as FIRST(<typeDefinition><typeDefinitions>) = FIRST(<typeDefinition>) = {TK_RECORD}.
Also
FOLLOW(<typeDefinitions>) = FIRST(<declarations>) = FIRST(<declaration>) = {TK_TYPE}
Next,
FIRST(<typeDefinition><typeDefinitions>) ∩ FOLLOW(typeDefinitions>)
= {TK_RECORD} ∩ {TK_TYPE} = ϕ
Hence the give two productions for the non terminal <typeDefinitions> conform to the LL(1) specifications.

FIRST(<typeDefinitions>) = FIRST(<typeDefinition>) = {TK_RECORD}

Recall the construction of the Predictive Parsing table.
The rule <typeDefinitions>===>eps is populated as the table entry T[<typeDefinitions>, TK_TYPE] and
the rule <typeDefinitions>===><typeDefinition><typeDefinitions> is populated as the table entry T[<typeDefinition>,TK_RECORD]

14. <typeDefinition>===>TK_RECORD TK_RECORDID <fieldDefinitions> TK_ENDRECORD TK_SEM
Here on, I am ignoring the check for LL(1) if the nonterminal has a single non nullable production.
FIRST(<typeDefinition>) = {TK_RECORD}

15. <fieldDefinitions>===> <fieldDefinition><fieldDefinition><moreFields>
As two consecutive occurrences of <fieldDefinition> are there at the RHS of the above production, this imposes a requirement of atleast two fields in a record. There can be one or more fields afterwrds. As the above rule is single and non nullable, it is LL(1) compatible.
FIRST(<fieldDefinitions>) = FIRST(<fieldDefinition>) = {TK_TYPE}

16. <fieldDefinition>===> TK_TYPE <primitiveDatatype> TK_COLON TK_FIELDID TK_SEM
FIRST(<fieldDefinition>) = {TK_TYPE}

17. <moreFields>===><fieldDefinition><moreFields> | eps
FOLLOW(<moreFields>) = ϕ ∪ FOLLOW(<fieldDefinitions>) = {TK_ENDRECORD} .....using 15 & 14

FIRST(<fieldDefinition><moreFields>) ∩ FOLLOW(<moreFields>)
= {TK_TYPE} ∩ {TK_ENDRECORD}
= ϕ
Hence, the above two rules for the non terminal <moreFields> are LL(1) compatible.

18. <declarations> ===> <declaration><declarations>|eps

FOLLOW(<declarations>) = FIRST(<otherStmts>) ∪ FOLLOW(<otherStmts>)

= { TK_ID, TK_RECORD_ID, TK_WHILE, TK_IF, TK_READ, TK_WRITE, TK_SQL, TK_CALL}

FIRST(<declaration><declarations>) = FIRST(<declaration>) = {TK_TYPE}

We observe that both of the above sets are disjoint. Therefore both the above productions for the non terminal <declarations> conform to LL(1) specifications.

19. <declaration>===> TK_TYPE <dataType> TK_COLON TK_ID TK_COLON <global_or_not> TK_SEM

This rule has a correction as follows

<declaration>===> TK_TYPE <dataType> TK_COLON TK_ID <global_or_not> TK_SEM

Also to incorporate the optional existence of global keyword prefixed with a colon is reflected as modification in rule 20.

FIRST(<declaration>) = {TK_TYPE}

Rule conforms to LL(1) specifications.

20. <global_or_not>===>TK_GLOBAL| eps

This rule is also modified as follows

<global_or_not>===>TK_COLON TK_GLOBAL| eps

FOLLOW(<global_or_not>) = {TK_SEM} ......using 19

and

FIRST(TK_COLON TK_GLOBAL) = {TK_COLON}

As

FOLLOW(<global_or_not>) ∩ FIRST(TK_COLON TK_GLOBAL) = {TK_SEM} ∩ {TK_COLON} = ϕ

Therefore the new rule for <global_or_not> is LL(1) compatible.

21. <otherStmts>===> <stmt><otherStmts> | eps

FOLLOW(<otherStmts>) = FIRST(<returnStmts>)

FIRST(<otherStmts>) = FIRST(<stmt>) [Note: <stmt> does not derive an empty string]

= { TK_ID, TK_RECORDID, TK_WHILE, TK_IF, TK_READ, TK_WRITE, TK_SQL, TK_CALL}
.....using 22

22. <stmt>===> <assignmentStmt> | <iterativeStmt>|<conditionalStmt>|<ioStmt>| <funCallStmt>
Refering rules 23, 28, 29, 31 and 25, we conclude that the five productions given above for the nonterminal <stmt> are not nullable.
We must check the LL(1) property that the FIRST sets of the RHSs of all productions above are disjoint.
FIRST(<assignmentStmt>) ∩ FIRST(<iterativeStmt>) ∩ FIRST(<conditionalStmt>) ∩ FIRST(<ioStmt>) ∩ FIRST(<funCallStmt>)
= {TK_ID, TK_RECORDID} ∩ {TK_WHILE} ∩ {TK_IF} ∩ {TK_READ, TK_WRITE} ∩ {TK_SQL, TK_CALL}
= ϕ
Hence, the above five productions for the nonterminal <stmt> conform to LL(1) specifications.
and ,
FIRST(<stmt>) = {TK_ID, TK_RECORDID, TK_WHILE, TK_IF, TK_READ, TK_WRITE, TK_SQL, TK_CALL}

23. <assignmentStmt>===><SingleOrRecId> TK_ASSIGNOP <arithmeticExpression> TK_SEM
FIRST(<assignmentStmt>) = FIRST(<singleOrRecId>) = {TK_ID, TK_RECORDID} ....using 24

24. <singleOrRecId>===>TK_ID | TK_RECORDID TK_DOT TK_FIELDID
The token TK_RECORDID is replaced with TK_ID as the identifier of record type expands with dot followed by the name of the field and not the record identifier name. You should left factor the new grammar rule to make it LL(1) compatible.
<singleOrRecId>===>TK_ID | TK_ID TK_DOT TK_FIELDID

Both of the above productions for the nonterminal <singleOrRecId> are non nullable and their first sets are not disjoint. This requires left factoring
The above two rules are left factored using new non terminal <new_24>
<singleOrRecId>===>TK_ID <new_24>
<new_24> ===> eps | TK_DOT TK_FIELDID

Therefore, the above productions are LL(1) compatible.
Also FIRST(<singleOrRecId>) = {TK_ID}

25. <funCallStmt>===><outputParameters> TK_CALL TK_FUNID TK_WITH TK_PARAMETERS <inputParameters>
The rule must have at its end TK_SEM to represent semicolon
FIRST(<funCallStmt>) = FIRST(<outputParameters>) ∪ FOLLOW(<outputParameters>)
= {TK_SQL} ∪ {TK_CALL} = {TK_SQL, TK_CALL}
The above rule conforms to LL(1) specifications.

26. <outputParameters> ==> TK_SQL <idList> TK_SQR TK_ASSIGNOP | eps
FOLLOW(<outputParameters>) = {TK_CALL}
Due to the presence of the nullable production for the non terminal <outputParameters>, we check the following LL(1) property
FIRST(TK_SQL <idList> TK_SQR TK_ASSIGNOP) ∩ FOLLOW(<outputParameters>)
= {TK_SQL} ∩ {TK_CALL} = ϕ
Hence, the productions in 26 are LL(1) compatible.
FIRST(<outputParameters>) = {TK_SQL}

27. <inputParameters>===> TK_SQL <idList> TK_SQR
LL(1) compatible with FIRST(<inputParameters>) = {TK_SQL}

28. <iterativeStmt>===> TK_WHILE TK_OP <booleanExpression> TK_CL <stmt><otherStmts> TK_ENDWHILE
LL(1) compatible with FIRST(<iterativeStmt>) = {TK_WHILE}

29. <conditionalStmt>===> TK_IF <booleanExpression> TK_THEN <stmt><otherStmts> TK_ELSE <otherStmts> TK_ENDIF

30. <conditionalStmt>===> TK_IF <booleanExpression> TK_THEN <stmt><otherStmts> TK_ENDIF

The two rules (29) and (30) for the nonterminal <conditionalStmt> need modifications as they originally form ambigous grammar. We need to perform the left factoring as follows

<conditionalStmt>===> TK_IF TK_OP <booleanExpression> TK_CL TK_THEN <stmt><otherStmts> <elsePart> .....number rule as 29

<elsePart>===>TK_ELSE <stmt><otherStmts> TK_ENDIF | TK_ENDIF .....number rules as 30

Notice that I also added TK_OP and TK_CL in new rule 29 to enclose boolean expression . These two tokens were missing earlier.
Also I am putting a constraint of atleast one statement within else part.
Now there is a single rule for <conditionalStmt> which is non nullable and its first set is {TK_IF}
But the newly introduced nonterminal <elsePart> has two non nullable productions and their corrsponding FIRST sets are disjoint.
FIRST(TK_ELSE <otherStmts> TK_ENDIF) ∩ FIRST(TK_ENDIF) = {TK_ELSE} ∩ {TK_ENDIF} = ϕ
Therefore the new set of rules for <elsePart> is also LL(1) compatible.

And,
FIRST(<conditionalStmt>) = {TK_IF}
FIRST(<elsePart>) = {TK_ELSE, TK_ENDIF}

31. <ioStmt>===>TK_READ TK_OP <allVar> TK_CL TK_SEM | TK_WRITE TK_OP <allVar> TK_CL TK_SEM

This rule needs a minor modification. The identifier and record identifier are the only arguments of the read operation.
<ioStmt>===> TK_READ TK_OP <singleOrRecId> TK_CL TK_SEM
Also the write statement is not expected to take as argument a record id. Any number, real number, identifier or constructed record identifier can be the arguments of write statement.
<ioStmt> ===> TK_WRITE TK_OP <allVar> TK_CL TK_SEM

Both the above productions are non nullable.
and,

FIRST(TK_READ TK_OP <singleOrRecId > TK_CL TK_SEM) ∩ FIRST(TK_WRITE TK_OP <allVar> TK_CL TK_SEM)
= {TK_READ} ∩ {TK_WRITE} = ϕ
Therefore, the above productions are LL(1) compatible.
and,
FIRST(<ioStmt>) = {TK_READ, TK_WRITE}
Notice that the arguments for a write may be a number, real number, id, or a constructed record id variable. A record identifier is not the argument of WRITE directly. This is relaxed here to simplify the code generation equivalent of printing all field values of the record type variable.

32. <allVar>===><var>| TK_RECORDID TK_DOT TK_FIELDID

Trivially, the two non nullable productions above conform to LL(1) and
FIRST(<allVar>) = FIRST(<var>) ∪ {TK_RECORDID}
= {TK_NUM, TK_RNUM, TK_ID, TK_RECORDID}

33. <arithmeticExpression>===><arithmeticExpression> <operator> <arithmeticExpression>
I had left a major correction for you to do in the arithmetic expression grammar. You were required to impose precedence of operators and the operations on record variables.

34. <arithmeticExpression> ====>TK_OP <arithmeticExpression> TK_CL | <var>
A replacement of <var> with a new nonterminal <all> is required to incorporate the record addition and subtraction
<arithmeticExpression> ====>TK_OP <arithmeticExpression> TK_CL | <all>
where <all>===> TK_ID | TK_NUM| TK_RNUM | TK_RECORDID <temp>
and <temp>===> eps | TK_DOT TK_FIELD

An arithmetic expression is expected to take as argument an identifier, statically available number, real number, a record identifier and a record id (constructed by expanding the name using dot with a field name)

Now we reformulate the rules 33, 34(a) and 35 to impose precedence.

<arithmeticExpression> ===> <arithmeticExpression> <lowPrecedenceOperators> <term> | <term> .....(A1)
<term> ===> <term> <highPrecedenceOperators> <factor> | <factor> .....(A2)
<factor> ===> TK_OP <arithmeticExpression> TK_CL | <all> .....(A3)
<highPrecedenceOperator>===> TK_MUL | TK_DIV .....(A4)
<lowPrecedenceOperators> ===> TK_PLUS | TK_MINUS ......(A5)
Also we have
<all>===> TK_ID | TK_NUM| TK_RNUM | TK_RECORDID <temp> ....(A6)
and <temp>===> eps | TK_DOT TK_FIELD ....(A7)

[Note : I have given new numbers for referring to the rules, but you can have your own numbering scheme for the rules]
Now let us verify the rules A1-7 for the nonterminals <arithmeticExpression>, <term>, <factor>, <lowPrecedenceOperators>, <highPrecedenceOperators>, <all> and <temp>
The rule A1 needs left recursion elimination and the new set of productions are obtained by introducing new nonterminal, say <expPrime>
<arithmeticExpression> ===> <term> <expPrime>
<expPrime> ===> <lowPrecedenceOperators> <term> <expPrime> | eps
Similarly rule A2 also needs left recursion elimination. The new set of productions are obtained by introducing new nonterminal, say, <termPrime>
<term>===> <factor> <termPrime>
<termPrime> ===> <highPrecedenceOperators><factor> <termPrime> | eps

Now we have two more nonterminals <expPrime> and <termPrime>. The complete set of non left recursive, unambigous arithmetic expression grammar then becomes with new numbers A1-A9.

A1) <arithmeticExpression> ===> <term> <expPrime>
A2) <expPrime> ===> <lowPrecedenceOperators> <term> <expPrime> | eps
A3) <term>===> <factor> <termPrime>
A4) <termPrime> ===> <highPrecedenceOperators><factor> <termPrime> | eps

A5) <factor> ===> TK_OP <arithmeticExpression> TK_CL | <all>

A6) <highPrecedenceOperator>===> TK_MUL | TK_DIV

A7) <lowPrecedenceOperators> ===> TK_PLUS | TK_MINUS

A8) <all>===> TK_ID | TK_NUM| TK_RNUM | TK_RECORDID <temp>

A9) <temp>===> eps | TK_DOT TK_FIELDID

Analysis of A1: As there is only single rule available for the nonterminal <arithmeticExpression>, we must see that is not nullable

FIRST(<arithmeticExpression>) = FIRST(<term>) = FIRST(<factor>) = {TK_ID, TK_NUM, TK_RNUM, TK_RECORDIID, TK_OP}

There is no need to go further to check LL(1) compatibility. The grammar for <arithmeticExpression> is LL(1) compatible.

Analysis of A2: One of the productions for the nonterminal <expPrime> is nullable.

Let us verify FIRST(<lowPrecedenceOperators> <term> <expPrime>) and FOLLOW(<expPrime>) are disjoint.

FIRST(<lowPrecedenceOperators> <term> <expPrime>) = FIRST(<lowPrecedenceOperators>)
= {TK_PLUS, TK_MINUS} ......using analysis of A7

and,

FOLLOW(<expPrime>) = FOLLOW(<arithmeticExpression>) .....using A1
= {TK_SEM} .....using 23

This implies that

FIRST(<lowPrecedenceOperators> <term> <expPrime>) ∩ FOLLOW(<expPrime>)

= {TK_PLUS, TK_MINUS}∩{TK_SEM}

= ϕ

Therefore, A2 is also LL(1) compatible.

Analysis of A3:

As there is only single rule available for the nonterminal <term>, we must see that is not nullable

FIRST(<term>)

= FIRST(<factor>)
= {TK_ID, TK_NUM, TK_RNUM, TK_RECORDIID, TK_OP} ....using analysis of A5
There is no need to go further to check LL(1) compatibility. The grammar for <term> is LL(1) compatible.

<u>Analysis of A4:</u>

One of the productions for the nonterminal <termPrime> is nullable.
Let us verify FIRST(<highPrecedenceOperators><factor> <termPrime>) and FOLLOW(<termPrime>) are disjoint.
FIRST(<highPrecedenceOperators><factor> <termPrime>) = FIRST(<highPrecedenceOperators>)
={ TK_MUL, TK_DIV} .....using analysis of A6

And,
FOLLOW(<termPrime>)
= FOLLOW(<term>) .....using A3
= FIRST(<expPrime>) ∪ FOLLOW(<expPrime>) .....using A2
= FIRST(<lowPrecedenceOperators>) ∪ FOLLOW(<expPrime>) .....using A2
= {TK_PLUS, TK_MINUS} ∪ FOLLOW(<expPrime>) .....using analysis of A7
= {TK_PLUS, TK_MINUS} ∪ FOLLOW(<arithmeticExpression>) ....using A1
= {TK_PLUS, TK_MINUS} ∪ {TK_SEM} ....using 23
= {TK_PLUS, TK_MINUS, TK_SEM}

Therefore,
FIRST(<highPrecedenceOperators><factor> <termPrime>) ∩ FOLLOW(<termPrime>)
= { TK_MUL, TK_DIV} ∩ {TK_PLUS, TK_MINUS, TK_SEM}
= ϕ
Hence the rules defined by A4 conform to LL(1) specifications.

Analysis of A5: There are two productions for the non terminal <factor>.
FIRST(TK_OP <arithmeticExpression> TK_CL) = {TK_OP}
and,
FIRST(<all>) = {TK_ID, TK_NUM, TK_RNUM, TK_RECORDIID} ..... using analysis of A8
It is now obvious that no production for <factor> is a nullable production and,
FIRST(TK_OP <arithmeticExpression> TK_CL) ∩ FIRST(<all>) = ϕ
Therefore the grammar for <factor> is LL(1) compatible.
and,
FIRST(<factor>) = {TK_ID, TK_NUM, TK_RNUM, TK_RECORDIID, TK_OP}

Analysis of A6: The two productions of the nonterminal <highPrecedenceOperators> are trivially LL(1) compatible
FIRST(<highPrecedenceOperators>) = { TK_MUL, TK_DIV}
Analysis of A7: The two productions of the nonterminal <lowPrecedenceOperators> are also trivially LL(1) compatible
FIRST(<lowPrecedenceOperators>) = { TK_PLUS, TK_MINUS}

Analysis of A8: The two productions of the nonterminal <all> are also trivially LL(1) compatible.
FIRST(<all>) = { TK_NUM, TK_RNUM, TK_ID, TK_RECORDID}
Analysis of A9: There is an epsilon production for the nonterminal <temp> and it is essential to check that
FIRST(TK_DOT TK_FIELDID) ∩ FOLLOW(<temp>) is empty.
Now computing FOLLOW(<temp>),

FOLLOW(<temp>)
= FOLLOW(<all>) .....using A8
= FOLLOW(<factor>) .....using A5
= FIRST(<termPrime>)∪ FOLLOW(<termPrime>) .....using A4
= FIRST(<highPrecedenceOperators>) ∪ FOLLOW(<termPrime>) .....using A4
= {TK_MUL, TK_DIV} ∪ FOLLOW(<termPrime>) .....using analysis of A6

= {TK_MUL, TK_DIV} ∪ FOLLOW(<term>) .....using A3
= {TK_MUL, TK_DIV} ∪ FIRST(<expPrime>) ∪ FOLLOW(<expPrime>) .....using A2
= {TK_MUL, TK_DIV} ∪ FIRST(<lowPrecedenceOperators>) ∪ FOLLOW(<expPrime>) .....using A2
= {TK_MUL, TK_DIV} ∪ {TK_PLUS, TK_MINUS} ∪ FOLLOW(<expPrime>) .....using analysis of A7
= {TK_MUL, TK_DIV} ∪ {TK_PLUS, TK_MINUS} ∪ FOLLOW(<arithmeticExpression>) ....using A1
= {TK_MUL, TK_DIV} ∪ {TK_PLUS, TK_MINUS} ∪ {TK_SEM} ....using 23
= {TK_MUL, TK_DIV, TK_PLUS, TK_MINUS, TK_SEM}

Therefore,

FIRST(TK_DOT TK_FIELDID) ∩ FOLLOW(<temp>)
= {TK_DOT} ∩ {TK_MUL, TK_DIV, TK_PLUS, TK_MINUS, TK_SEM}
= ϕ

This proves that the rules defined in A9 are LL(1) compatible. [I again insist that I am avoiding proving every rule to be unambiguous. This is left to be proved by you.]

35. <operator> ===> TK_PLUS | TK_MUL |TK_MINUS|TK_DIV
    This rule is discarded now.
36. <booleanExpression>===>TK_OP <booleanExpression> TK_CL <logicalOp> TK_OP <booleanExpression> TK_CL
    FIRST(TK_OP <booleanExpression> TK_CL <logicalOp> TK_OP <booleanExpression> TK_CL) = { TK_OP}
    and the production is not nullable.
37. <booleanExpression>===> <var> <relationalOp> <var>
    FIRST(<var> <relationalOp> <var>) = FIRST(<var>)
    = {TK_ID, TK_NUM, TK_RNUM}
    and the production is not nullable.

38. <booleanExpression>===> TK_NOT <booleanExpression>

    Considering the above rules 36-38 for analysis, we observe that none of the three productions for the nonterminal <booleanExpression> is nullable (using First set of <var> from 39)

    and, The first sets of the right hand sides of the three productions are disjoint.

    FIRST(TK_OP <booleanExpression> TK_CL <logicalOp> TK_OP <booleanExpression> TK_CL)

    ∩ FIRST(<var> <relationalOp> <var>)

    ∩ FIRST(TK_NOT <booleanExpression>)

    = {TK_OP} ∩ {TK_ID, TK_NUM, TK_RNUM}∩ {TK_NOT}

    = ϕ

    Hence, the three productions for the nonterminal <booleanExpression> are LL(1) compatible.

39. <var>===> TK_ID | TK_NUM | TK_RNUM

    Trivially the three non nullable productions above for the non terminal <var> are LL(1) compatible.

    and,

    FIRST(<var>) = {TK_ID, TK_NUM, TK_RNUM}

40. <logicalOp>===>TK_AND | TK_OR

    I have not defined the precedence of AND over OR or vice versa. We have the support to enclose the boolean expression in parentheses.

    Trivially the productions for <logicalOp> conform to LL(1) specifications

41. <relationalOp>===> TK_LT | TK_LE | TK_EQ |TK_GT | TK_GE | TK_NE

    All six productions above for the nonterminal <relationalOp> are trivially LL(1) compatible and

    FIRST(<relationalOp>) = { TK_LT, TK_LE, TK_EQ , TK_GT , TK_GE , TK_NE}

42. <returnStmt>===>TK_RETURN <optionalReturn> TK_SEM

    FIRST(<returnStmt>) = {TK_RETURN}

    and the production is not nullable, hence the rule is LL(1) compatible.

43. <optionalReturn>===>TK_SQL <idList> TK_SQR | eps

FIRST(TK_SQL <idList> TK_SQR) = { TK_SQL}
and,
FOLLOW(<optionalReturn>) = {TK_SEM} .....using 42
Since both of the above sets are disjoint, the grammar for the nonterminal <optionalReturn> is LL(1) compatible.

44. <idList>===> TK_ID <more_ids>
FIRST(<idList>) = {TK_ID} and the production is not nullable, hence the rule is LL(1) compatible.

45. <more_ids>===> TK_COMMA <idList> | eps
FIRST(TK_COMMA <idList>) = {TK_COMMA}
and,
FOLLOW(<more_ids>) = FOLLOW(<idList>) ....using 44
= {TK_SQR} ....using 43
This implies that
FIRST(TK_COMMA <idList>) ∩ FOLLOW(<more_ids>)
= {TK_COMMA} ∩ {TK_SQR}
= ϕ
Therefore, the rules for the nonterminal <more_ids> are LL(1) conformable.

Vandana