Semantic Rules for AST Creation

Group 40
Siddharth Sharma (2019A7PS0064P)
Atharva Chandak (2019A7PS0062P)
Archit Bhatnagar (2019A7PS0133P)
Suchismita Tripathy (2019A7PS0554P)
Srujan Deolasee (2019A7PS1139P)

Semantic Rules for Abstract Syntax Tree creation

Functions to be used:

a) mknode(string **s**) : Creates a new node with value **s** and returns pointer to this node
b) mkleaf(string **s**, entry **e**) : Creates a leaf node with value **s** and related entry **e** and returns pointer to this leaf node
c) insert(pointer **a**, pointer **b**) : Inserts a new node for pointer **b** after pointer **a** in the linked list
d) addChild(pointer **a**, pointer **b**) : Inserts a new child to **a** using pointer **b**
e) free(pointer **a**): Assigns the node after freed node as the current child of parent

1. <program> ===> <otherFunctions> <mainFunction>

   Semantic Rule:
   <program>.ptr = mknode(“<program>”)
   addChild(<program>.ptr,<otherFunctions>.ptr)
   addChild(<program>.ptr,<mainFunction>.ptr)

2. <mainFunction> ===> TK_MAIN <stmts> TK_END

   Semantic Rule:
   <mainFunction>.ptr = <stmts>.ptr

3. <otherFunctions> ===> <function> <otherFunctions1>

   Semantic Rule:
   insert(<otherFunctions>.ptr,<function>.ptr)
   insert(<function>.ptr,<otherFunctions1>.ptr)

free(<otherFunctions>.ptr)

4. <otherFunctions> ===> ∈

   Semantic Rule:
   <otherFunctions>.ptr = NULL

5. <function> ===> TK_FUNID <input_par> <output_par> TK_SEM <stmts> TK_END

   Semantic Rule:
   <function>.ptr = mknode("<function>")
   addChild(<function>.ptr, mkleaf("TK_FUNID",entry.id)
   addChild(<function>.ptr, <input_par>.ptr)
   addChild(<function>.ptr, <output_par>.ptr)
   addChild(<function>.ptr, <stmts>.ptr)

6. <input_par> ===> TK_INPUT TK_PARAMETER TK_LIST TK_SQL <parameter_list> TK_SQR

   Semantic Rule:
   insert(<input_par>.ptr, mkleaf("TK_INPUT",entry.id))
   insert(<input_par>.ptr->next, <parameter_list>.ptr)
   free (<input_par>.ptr)

7. <output_par> ===> TK_OUTPUT TK_PARAMETER TK_LIST TK_SQL <parameter_list> TK_SQR

   Semantic Rule:
   insert(<input_par>.ptr, mkleaf("TK_OUTPUT",entry.id))
   insert(<output_par>.ptr, <parameter_list>.ptr)
   free (<output_par>.ptr)

8. <output_par>===>∈

   Semantic Rule:
   <output_par>.ptr = NULL;

9. <parameter_list> ===> <dataType> TK_ID <remaining_list>

Semantic Rule:
<parameter_list>.ptr = mknode(“<parameter_list”)
addChild(<parameter_list>.ptr,<dataType>.ptr)
addChild(<parameter_list>.ptr, mkleaf(“TK_ID”,entry.id))
addChild(<parameter_list>.ptr, <remaining_list>.ptr)

10. <dataType> ===> <primitiveDatatype>

    Semantic Rule:
    <dataType>.ptr = <primitiveDatatype>.ptr

11. <dataType> ===> <constructedDatatype>

    Semantic Rule:
    <dataType>.ptr = <constructedDatatype>.ptr

12. <primitiveDatatype> ===> TK_INT

    Semantic Rule:
    <primitiveDataType>.ptr = mkleaf(“TK_INT”, entry.id)

13. <primitiveDatatype> ===> TK_REAL

    Semantic Rule:
    < primitiveDataType>.ptr = mkleaf(“TK_REAL”,entry.id)

14. <constructedDatatype> ===> TK_RECORD TK_RUID

    Semantic Rule:
    <constructedDatatype>.ptr = mkleaf(“TK_RECORD TK_RUID”, entry.id)

15. <constructedDatatype> ===> TK_UNION TK_RUID

    Semantic Rule:
    <constructedDatatype>.ptr = mkleaf(“TK_UNION TK_RUID”, entry.id)

16. <constructedDatatype> ===> TK_RUID

    Semantic Rule:
    <constructedDatatype>.ptr = mkleaf(“TK_RUID”, entry.id)

17. <remaining_list> ===> TK_COMMA <parameter_list>

    Semantic Rule:
    <remaining_list>.ptr = <parameter_list>.ptr

18. <remaining_list> ===> ∈

    Semantic Rule:
    <remaining_list>.ptr = NULL

19. <stmts> ===> <typeDefinitions> <declarations> <otherStmts> <returnStmt>

    Semantic Rule:
    insert(<stmts>.ptr, <typeDefinitions>.ptr)
    insert(<typeDefinitions>.ptr, <declarations>.ptr)
    insert(<declarations>.ptr, <otherStmt>.ptr)
    insert(<otherStmt>.ptr, <returnStmt>.ptr)
    free(<stmt>.ptr)

20. <typeDefinitions> ===> <actualOrRedefined> <typeDefinitions1>

    Semantic Rule:
    insert(<typeDefinitions>.ptr, <actualOrRedefined>.ptr)
    insert(<actualOrRedefined>.ptr, <typeDefinitions1>.ptr)
    free(<typeDefinitions>.ptr)

21. <typeDefinitions> ===> ∈

    Semantic Rule:
    <typeDefinitions>.ptr = NULL

22. <actualOrRedefined> ===> <typeDefinition>

    Semantic Rule:
    <actualOrRedefined>.ptr = <typeDefinition>.ptr

23. <actualOrRedefined> ===> <definetypestmt>

    Semantic Rule:

<actualOrRedefined>.ptr = <definetypestmt>.ptr

24. <typeDefinition> ===> TK_RECORD TK_RUID <fieldDefinitions> TK_ENDRECORD

    Semantic Rule:
    <typeDefinition>.ptr = mknode(“TK_RECORD”)
    addChild(<typeDefinition>.ptr , mkleaf(“TK_RUID”, entry.id))
    addChild(<typeDefinition>.ptr, <fieldDefinitions>.ptr)

25. <typeDefinition> ===> TK_UNION TK_RUID <fieldDefinitions> TK_ENDUNION

    Semantic Rule:
    <typeDefinition>.ptr = mknode(“TK_UNION”)
    addChild(<typeDefinition>.ptr , mkleaf(“TK_RUID”, entry.id))
    addChild(<typeDefinition>.ptr, <fieldDefinitions>.ptr)

26. <fieldDefinitions> ===> <fieldDefinition> <fieldDefinition1> <moreFields>

    Semantic Rule:
    insert(<fieldDefinitions>.ptr,<fieldDefinition>.ptr)
    insert(<fieldDefinition>.ptr,<fieldDefinition1>.ptr)
    insert(<fieldDefinition1>.ptr,<moreFields>.ptr)
    free(<fieldDefinitions>.ptr)

27. <fieldDefinition> ===> TK_TYPE <fieldType> TK_COLON TK_FIELDID TK_SEM

    Semantic Rule:
    <fieldDefinition>.ptr = mknode(“<fieldDefinition>”)
    addChild(<fieldDefinition>.ptr, <fieldType>.ptr)
    addChild(<fieldDefinition>.ptr, mkleaf(“TK_FIELDID”, entry.id))

28. <fieldType> ===> <primitiveDatatype>

    Semantic Rule:
    <fieldType>.ptr = <primitiveDatatype>.ptr

29. <fieldType> ===> TK_RUID

    Semantic Rule:

<fieldType>.ptr = mkleaf(“TK_RUID”, entry.id)

30. <moreFields> ===> <fieldDefinition> <moreFields1>

    Semantic Rule:
    insert(<moreFields>.ptr, <fieldDefinition>.ptr)
    insert(<fieldDefinition>.ptr, <moreFields1>.ptr)
    free(<moreFields>.ptr)

31. <moreFields> ===> ∈

    Semantic Rule:
    <moreFields>.ptr = NULL

32. <definetypestmt> ===> TK_DEFINETYPE <A> TK_RUID TK_AS TK_RUID

    Semantic Rule:
    addChild(<definetypestmt>.ptr,mkleaf(“TK_DEFINETYPE”,entry.id))
    addChild(<definetypestmt>.ptr, <A>.ptr)
    addChild(<definetypestmt>.ptr, mkleaf(“TK_RUID”, entry.id)
    addChild(<definetypestmt>.ptr, mkleaf(“TK_AS”, entry.id)
    addChild(<definetypestmt>.ptr, mkleaf(“TK_RUID”, entry.id)

33. <declarations> ===> <declaration> <declarations1>

    Semantic Rule:
    insert(<declarations>.ptr, <declaration>.ptr)
    insert(<declaration>.ptr, <declarations1>.ptr)
    free(<declarations>.ptr)

34. <declarations> ===> ∈

    Semantic Rule:
    <declarations>.ptr = NULL

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

    Semantic Rule:
    <declaration>.ptr = mknode(“<declaration>”)

addChild(<declaration>.ptr, <dataType>.ptr)
addChild(<declaration>.ptr, mkleaf(“TK_ID”, entry.id)
addChild(<declaration>.ptr, <global_or_not>.ptr)

36. <global_or_not> ===> TK_COLON TK_GLOBAL

    Semantic Rule:
    <global_or_not>.ptr = mkleaf(“TK_GLOBAL”, entry.id)

37. <global_or_not> ===> ∈

    Semantic Rule:
    <global_or_not>.ptr = NULL

38. <otherStmts> ===> <stmt> <otherStmts1>

    Semantic Rule:
    insert(<otherStmts>.ptr, <stmt>.ptr)
    insert(<stmt>.ptr, <otherStmts1>.ptr)
    free(<otherStmts>.ptr)

39. <otherStmts> ===> ∈

    Semantic Rule:
    <otherStmts>.ptr = NULL

40. <stmt> ===> <assignmentStmt>

    Semantic Rule:
    <stmt>.ptr = <assigmentStmt>.ptr

41. <stmt> ===> <iterativeStmt>

    Semantic Rule:
    <stmt>.ptr = <iterativeStmt>.ptr

42. <stmt> ===> <conditionalStmt>

    Semantic Rule:
    <stmt>.ptr = <conditionalStmt>.ptr

43. <stmt> ===> <ioStmt>

    Semantic Rule:
    <stmt>.ptr = <ioStmt>.ptr

44. <stmt> ===> <funCallStmt>

    Semantic Rule:
    <stmt>.ptr = <funCallStmt>.ptr

45. <assignmentStmt> ===> <singleOrRecId> TK_ASSIGNOP
    <arithmeticExpression> TK_SEM

    Semantic Rule:
    <assignmentStmt>.ptr = mknode(“<assignmentStmt>”)
    addChild(<assignmentStmt>.ptr, <singleOrRecId>.ptr)
    addChild(<assignmentStmt>.ptr, mkleaf(“TK_ASSIGNOP”, entry.id))
    addChild(<assignmentStmt>.ptr, <arithmeticExpression>.ptr)

46. <singleOrRecId> ===> TK_ID <option_single_constructed>

    Semantic Rule:
    <singleOrRecId>.ptr = mknode(“<singleOrRecId>”)
    addChild(<singleOrRecId>.ptr, mkleaf(“TK_ID”, entry.id))
    addChild(<singleOrRecId>.ptr, <option_single_constructed>.ptr)

47. <option_single_constructed> ===> <oneExpansion> <moreExpansions>

    Semantic Rule:
    insert(<option_single_constructed>.ptr, <oneExpansion>.ptr)
    insert(<oneExpansion>.ptr, <moreExpansions>.ptr)
    free(<oneExpansion>.ptr)

48. <option_single_constructed> ===> ∈

    Semantic Rule:
    <option_single_constructed>.ptr = NULL

49. <oneExpansion> ===> TK_DOT TK_FIELDID

Semantic Rule:
<oneExpansion>.ptr = mkleaf(“TK_FIELDID”, entry.id)

50. <moreExpansions> ===> <oneExpansion> <moreExpansions1>

    Semantic Rule:
    insert(<moreExpansions>.ptr, <oneExpansion>.ptr)
    insert(<oneExpansion>.ptr, <moreExpansions1>.ptr)
    free(<moreExpansions>.ptr)

51. <moreExpansions> ===> ∈

    Semantic Rule:
    <moreExpansions>.ptr = NULL

52. <funCallStmt> ===> <outputParameters> TK_CALL TK_FUNID TK_WITH TK_PARAMETERS <inputParameters> TK_SEM

    Semantic Rule:
    <funCallStmt>.ptr = mknode(“<funCallStmt>”)
    addChild(<funCallStmt>.ptr, <outputParameters>.ptr)
    addChild(<funCallStmt>.ptr, mkleaf(“TK_CALL”, entry.id)
    addChild(<funCallStmt>.ptr, mkleaf(“TK_FUNID”, entry.id)
    addChild(<funCallStmt>.ptr, mkleaf(“TK_PARAMETERS”, entry.id))
    addChild(<funCallStmt>.ptr, <inputParameters>.ptr)

53. <outputParameters> ===> TK_SQL <idList> TK_SQR TK_ASSIGNOP

    Semantic Rule:
    <outputParameters>.ptr = mknode(“<outputParameters>”)
    addChild(<outputParameters>.ptr, <idList>.ptr)
    addChild(<outputParameters>.ptr,mkleaf(“TK_ASSIGNOP”,entry.id))

54. <outputParameters> ===> ∈

    Semantic Rule:
    <outputParameters>.ptr = NULL

55. <inputParameters> ===> TK_SQL <idList> TK_SQR

Semantic Rule:
<inputParameters>.ptr = <idList>.ptr

56. <iterativeStmt> ===> TK_WHILE TK_OP <booleanExpression> TK_CL <stmt> <otherStmts> TK_ENDWHILE

    Semantic Rule:
    <iterativeStmt>.ptr = mknode(“<iterativeStmt>”)
    addChild(<iterativeStmt>.ptr, <booleanExpression>.ptr)
    addChild(<iterativeStmt>.ptr, <stmt>.ptr)
    addChild(<iterativeStmt>.ptr,<otherStmts>.ptr)

57. <conditionalStmt> ===> TK_IF TK_OP <booleanExpression> TK_CL TK_THEN <stmt> <otherStmts> <elsePart>

    Semantic Rule:
    <conditionalStmt>.ptr = mknode(“<condi tionalStmt>”)
    addChild(<conditionalStmt>.ptr, mkleaf(“TK_IF”, entry.id))
    addChild(<conditionalStmt>.ptr, <booleanExpression>.ptr)
    addChild(<conditionalStmt>.ptr, mkleaf(“TK_THEN”, entry.id))
    addChild(<conditionalStmt>.ptr, <stmt>.ptr)
    addChild(<conditionalStmt>.ptr <otherStmts>.ptr)
    addChild(<conditionalStmt>.ptr, <elsePart>.ptr)

58. <elsePart> ===> TK_ELSE <stmt> <otherStmts> TK_ENDIF

    Semantic Rule:
    <elsePart>.ptr = mknode(“<elsePart>”)
    addChild(<elsePart>.ptr, <stmt>.ptr)
    addChild(<elsePart>.ptr,<otherStmts>.ptr)

59. <elsePart> ===> TK_ENDIF

    Semantic Rule:
    <elsePar>.ptr = mkleaf(“TK_ENDIF”, entry.id)

60. <ioStmt> ===> TK_READ TK_OP <var> TK_CL TK_SEM

    Semantic Rule:

<ioStmt>.ptr = mknode(“<ioStmt>”)
addChild(<ioStmt>.ptr, mkleaf(“TK_READ”, entry.id))
addChild(<ioStmt>.ptr, <var>.ptr)

61. <ioStmt> ===> TK_WRITE TK_OP <var> TK_CL TK_SEM

    Semantic Rule:
    <ioStmt>.ptr = mknode(“<ioStmt>”)
    addChild(<ioStmt>.ptr, mkleaf(“TK_WRITE”, entry.id))
    addChild(<ioStmt>.ptr, <var>.ptr)

62. <arithmeticExpression> ===> <term> <expPrime>

    Semantic Rule:
    insert(<arithmeticExpression>.ptr, <term>.ptr)
    insert(<term>.ptr, <expPrime>.ptr)
    free(<arithmeticExpression>.ptr)

63. <expPrime> ===> <lowPrecedenceOperators> <term> <expPrime>

    Semantic Rule:
    insert(<expPrime>.ptr, <lowPrecedenceOperators>.ptr)
    insert(<lowPrecedenceOperators>.ptr,<term>.ptr)
    insert(<term>.ptr,<expPrime>.ptr)
    free(<expPrime>.ptr)

64. <expPrime> ===> ∈
    Semantic Rule:
    <expPrime>.ptr = NULL

65. <term> ===> <factor> <termPrime>

    Semantic Rule:
    insert(<term>.ptr, <factor>.ptr)
    insert(<factor>.ptr, <termPrime>.ptr)
    free(<term>.ptr)

66. <termPrime> ===> <highPrecedenceOperators> <factor> <termPrime>

    Semantic Rule:

insert(<termPrime>.ptr, <highPrecedenceOperators>.ptr)
insert(<highPrecedenceOperators>.ptr,<factor>.ptr)
insert(<factor>.ptr,<termPrime>.ptr)
free(<termPrime>.ptr)

67. <termPrime> ===> ∈

    Semantic Rule:
    <termPrime>.ptr = NULL

68. <factor> ===> TK_OP <arithmeticExpression> TK_CL

    Semantic Rule:
    <factor>.ptr = <arithmeticEpression>.ptr

69. <factor> ===> <var>

    Semantic Rule:
    <factor>.ptr =<var>.ptr

70. <highPrecedenceOperators> ===> TK_MUL

    Semantic Rule:
    <highPrecedenceOperators>.ptr = mkleaf(“TK_MUL”, entry.id)

71. <highPrecedenceOperators> ===> TK_DIV

    Semantic Rule:
    <highPrecedenceOperators>.ptr = mkleaf(“TK_DIV”, entry.id)

72. <lowPrecedenceOperators> ===> TK_PLUS

    Semantic Rule:
    <highPrecedenceOperators>.ptr = mkleaf(“TK_PLUS”, entry.id)

73. <highPrecedenceOperators> ===> TK_MINUS

    Semantic Rule:
    <highPrecedenceOperators>.ptr = mkleaf(“TK_MINUS”, entry.id)

74. <booleanExpression> ===> TK_OP <booleanExpression1> TK_CL <logicalOp> TK_OP <booleanExpression2> TK_CL

    Semantic Rule:
    <booleanExpression>.ptr = mknode(“<booleanExpression>”)
    addChild(<booleanExpression>.ptr, <booleanExpression1>.ptr)
    addChild(<booleanExpression>.ptr, <logicalOp>.ptr)
    addChild(<booleanExpression>.ptr, <booleanExpression2>.ptr)

75. <booleanExpression> ===> <var> <relationalOp> <var1>

    Semantic Rule:
    <booleanExpression>.ptr = mknode(“<booleanExpression>”)
    addChild(<booleanExpression>.ptr, <var>.ptr)
    addChild(<booleanExpression>.ptr, <relationalOp>.ptr)
    addChild(<booleanExpression>.ptr,<var1>.ptr)

76. <booleanExpression> ===> TK_NOT TK_OP <booleanExpression1> TK_CL

    Semantic Rule:
    insert(<booleanExpression>.ptr, mkleaf(“TK_NOT”, entry.id))
    insert(<booleanExpression>.ptr->next, <booleanExpression1>.ptr)
    free(<booleanExpression>.ptr)

77. <var> ===> <singleOrRecId>

    Semantic Rule:
    <var>.ptr = <singleOrRecId>.ptr

78. <var> ===> TK_NUM

    Semantic Rule:
    <var>.ptr = mkleaf(“TK_NUM”, entry.value)

79. <var> ===> TK_RNUM

    Semantic Rule:
    <var>.ptr = mkleaf(“TK_RNUM”, entry.value)

80. <logicalOp> ===> TK_AND

Semantic Rule:
<logicalOp>.ptr = mkleaf(“TK_AND”, entry.id)

81. <logicalOp> ===> TK_OR

    Semantic Rule:
    <logicalOp>.ptr = mkleaf(“TK_OR”, entry.id)

82. <relationalOp> ===> TK_LT

    Semantic Rule:
    <relationalOp>.ptr = mkleaf(“TK_LT”, entry.id)

83. <relationalOp> ===> TK_LE

    Semantic Rule:
    <relationalOp>.ptr = mkleaf(“TK_LE”, entry.id)

84. <relationalOp> ===> TK_EQ

    Semantic Rule:
    <relationalOp>.ptr = mkleaf(“TK_EQ”, entry.id)

85. <relationalOp> ===> TK_GT

    Semantic Rule:
    <relationalOp>.ptr = mkleaf(“TK_GT”, entry.id)

86. <relationalOp> ===> TK_GE

    Semantic Rule:
    <relationalOp>.ptr = mkleaf(“TK_GE”, entry.id)

87. <relationalOp> ===> TK_NE

    Semantic Rule:
    <relationalOp>.ptr = mkleaf(“TK_NE”, entry.id)

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

Semantic Rule:
<returnStmt>.ptr = mknode(“<returnStmt>”)
addChild(<returnStmt>.ptr, mkleaf(“TK_RETURN”, entry.id)
addChild(<returnStmt>.ptr,<optionalReturn>.ptr)

89. <optionalReturn> ===> TK_SQL <idList> TK_SQR

    Semantic Rule:
    <optionalReturn>.ptr = <idList>.ptr

90. <optionalReturn> ===> ∈

    Semantic Rule:
    <optionalReturn>.ptr = NULL

91. <idList> ===> TK_ID <more_ids>

    Semantic Rule;
    insert(<idList>.ptr, mkleaf(“TK_ID”, entry.id))
    insert(<idList>.ptr->next, <more_ids>.ptr)
    free(<idList>.ptr)

92. <more_ids> ===> TK_COMMA <idList>

    Semantic Rule:
    <more_ids>.ptr = <idList>.ptr

93. <more_ids> ===> ∈
    Semantic Rule:
    <more_ids>.ptr = NULL

94. <A> ===> TK_RECORD

    Semantic Rule:
    <A>.ptr = mkleaf(“TK_RECORD”, entry.id)

95. <A> ===> TK_UNION

    Semantic Rule:

<A>.ptr = mkleaf(“TK_UNION”, entry.id)