INFO-F-403 : Language theory and compiling
Rapport projet partie 2 - Grammaire

Simon Picard
Arnaud Rosette

18 décembre 2014

1 Transformation de la grammaire donnée

1.1 Opérateurs binaires et unaires

La première étape pour rendre la grammaire LL(1) fut la distinction des deux types d'opérateurs : les opérateurs binaires qui agissent sur deux expressions l'une située à gauche et l'autre à droite de ces opérateurs et les opérateurs unaires qui agissent sur une seule expression située à droite de ces opérateurs. Il faut différencier les deux types d'opérateurs afin de ne pas avoir des expressions du style : »4, *2, 6|,... .Les opérateurs unaires sont au nombre de quatre :! (negation), ~ (not bit à bit), + et -. Les opérateurs + et - sont également des opérateurs binaires.

1.2 Priorité et associativité des opérateurs

Lors de cette étape, nous avons modifié la grammaire afin de fixer la priorité et l'associativité des différents opérateurs, ceci afin de rendre la grammaire moins ambigue. Nous avons remarqué que les opérateurs unaires étaient plus prioritaires que les opérateurs binaires et que les opérateurs unaires étaient tous associatifs à droite, tandis que les opérateurs binaires l'étaient tous à gauche. Pour fixer la priorité, nous avons mis les opérateurs les moins prioritaires le plus "haut" possible dans la grammaire et les plus prioritaires le plus "bas" possible.

1.3 Suppression des récursion à gauche

L'étape suivante fut l'étape de suppression des récursions à gauche qui sont incompatibles avec les "top-down parser" car cela fait entrer ce genre de parser dans une boucle ou récursion infinie.

1.4 Left factoring

Cette étape rassemble les règles de production d'une variable qui ont un préfixe commun en une seule règle de production qui contient ce préfixe commun et une nouvelle variable. Cette nouvelle variable possède des règles de production vers les différents suffixes qui étaient présents à l'origine dans les différentes règles de production qui ont été mises en commun. Ceci est nécessaire car le parser qu'on va créer est LL(1), il faut donc qu'il puisse choisir la bonne règle de production en regardant seulement le prochain token qui est en entrée.

1.5 Suppression de la variable <Instruction>

Nous avons décidé de supprimer la variable <Instruction> et de ne laisser que <InstructionList>. <InstructionList> pouvant être une liste d'instruction, une instruction vide ou rien (epsilon), ceci afin de rendre la grammaire moins ambigue, plus allégée et sans grande perte de fonctionnalité du langage.

2 Grammaire

[1]	<Program>	→	<InstructionList>
[2]	<Instruction>	→	<IdentifierInstruction>

[3]		$\rightarrow$	<ConstDefinition>
[4]		$\rightarrow$	<Block>
[5]		$\rightarrow$	<Loop>
[6]		$\rightarrow$	<BuiltInFunctionCall>
[7]		$\rightarrow$	<FunctionDefinition>
[8]	<InstructionList>	$\rightarrow$	<Instruction> <InstructionListTail>
[9]		$\rightarrow$	<InstructionListTail>
[10]	<InstructionListTail>	$\rightarrow$	END_OF_INSTRUCTION <InstructionList>
[11]		$\rightarrow$	ϵ
[12]	<IdentifierInstruction>	$\rightarrow$	IDENTIFIER <IdentifierInstructionTail>
[13]	<IdentifierInstructionTail>	$\rightarrow$	<AssignationTail>
[14]		$\rightarrow$	TYPE_DEFINITION <Type>
[15]		$\rightarrow$	<FunctionCallTail>
[16]	<AssignationTail>	$\rightarrow$	ASSIGNATION <Expression>
[17]		$\rightarrow$	COMMA IDENTIFIER <AssignationTail> COMMA <Expression>
[18]	<ConstDefinition>	$\rightarrow$	CONST IDENTIFIER <AssignationTail>
[19]	<Block>	$\rightarrow$	LET IDENTIFIER <AssignationTail> END_OF_INSTRUCTION <InstructionList> END
[20]	<Loop>	$\rightarrow$	<If>
[21]		$\rightarrow$	WHILE <Expression> END_OF_INSTRUCTION <InstructionList> END
[22]		$\rightarrow$	FOR IDENTIFIER ASSIGNATION <Expression> TERNARY_ELSE <Expression> <ForTail>
[23]	<ForTail>	$\rightarrow$	END_OF_INSTRUCTION <InstructionList> END
[24]		$\rightarrow$	TERNARY_ELSE <Expression> END_OF_INSTRUCTION <InstructionList> END
[25]	<Type>	$\rightarrow$	BOOLEAN_TYPE
[26]		$\rightarrow$	REAL_TYPE
[27]		$\rightarrow$	INTEGER_TYPE
[28]	<Expression>	$\rightarrow$	<BinaryExpression> <TernaryIfExpression>
[29]	<TernaryIfExpression>	$\rightarrow$	TERNARY_IF <Expression> <TernaryElseExpression>
[30]		$\rightarrow$	ϵ
[31]	<TernaryElseExpression>	$\rightarrow$	TERNARY_ELSE <Expression>
[32]	<AtomicExpression>	$\rightarrow$	<AtomicIdentifierExpression>
[33]		$\rightarrow$	INTEGER
[34]		$\rightarrow$	REAL
[35]		$\rightarrow$	BOOLEAN
[36]		$\rightarrow$	<BuiltInFunctionCall>
[37]	<AtomicIdentifierExpression>	$\rightarrow$	IDENTIFIER <AtomicIdentifierExpressionTail>
[38]	<AtomicIdentifierExpressionTail>	$\rightarrow$	<FunctionCallTail>
[39]		$\rightarrow$	ϵ
[40]	<UnaryExpression>	$\rightarrow$	NEGATION <UnaryExpression>
[41]		$\rightarrow$	<UnaryBitwiseNotExpression>
[42]	<UnaryBitwiseNotExpression>	$\rightarrow$	BITWISE_NOT <UnaryBitwiseNotExpression>
[43]		$\rightarrow$	<UnaryMinusPlusExpression>
[44]	<UnaryMinusPlusExpression>	$\rightarrow$	MINUS <UnaryMinusPlusExpression>
[45]		$\rightarrow$	PLUS <UnaryMinusPlusExpression>
[46]		$\rightarrow$	<UnaryAtomicExpression>
[47]	<UnaryAtomicExpression>	$\rightarrow$	<AtomicExpression>
[48]		$\rightarrow$	LEFT_PARENTHESIS <Expression> RIGHT_PARENTHESIS
[49]	<BinaryExpression>	$\rightarrow$	<BinaryLazyOrExpression> <BinaryExpression'>
[50]	<BinaryExpression'>	$\rightarrow$	LAZY_OR <BinaryLazyOrExpression> <BinaryExpression'>

[51]		$\rightarrow$	ϵ
[52]	\<BinaryLazyOrExpression\>	$\rightarrow$	\<BinaryLazyAndExpression\> \<BinaryLazyOrExpression'\>
[53]	\<BinaryLazyOrExpression'\>	$\rightarrow$	LAZY_AND \<BinaryLazyAndExpression\> \<BinaryLazyOrExpression'\>
[54]		$\rightarrow$	ϵ
[55]	\<BinaryLazyAndExpression\>	$\rightarrow$	\<BinaryNumericExpression\> \<BinaryLazyAndExpression'\>
[56]	\<BinaryLazyAndExpression'\>	$\rightarrow$	GREATER_THAN \<BinaryNumericExpression\> \<BinaryLazyAndExpression'\>
[57]		$\rightarrow$	LESS_THAN \<BinaryNumericExpression\> \<BinaryLazyAndExpression'\>
[58]		$\rightarrow$	GREATER_OR_EQUALS_THAN \<BinaryNumericExpression\> \<BinaryLazyAndExpression'\>
[59]		$\rightarrow$	LESS_OR_EQUALS_THAN \<BinaryNumericExpression\> \<BinaryLazyAndExpression'\>
[60]		$\rightarrow$	EQUALITY \<BinaryNumericExpression\> \<BinaryLazyAndExpression'\>
[61]		$\rightarrow$	INEQUALITY \<BinaryNumericExpression\> \<BinaryLazyAndExpression'\>
[62]		$\rightarrow$	ϵ
[63]	\<BinaryNumericExpression\>	$\rightarrow$	\<BinaryTermExpression\> \<BinaryNumericExpression'\>
[64]	\<BinaryNumericExpression'\>	$\rightarrow$	PLUS \<BinaryTermExpression\> \<BinaryNumericExpression'\>
[65]		$\rightarrow$	MINUS \<BinaryTermExpression\> \<BinaryNumericExpression'\>
[66]		$\rightarrow$	BITWISE_OR \<BinaryTermExpression\> \<BinaryNumericExpression'\>
[67]		$\rightarrow$	BITWISE_XOR \<BinaryTermExpression\> \<BinaryNumericExpression'\>
[68]		$\rightarrow$	ϵ
[69]	\<BinaryTermExpression\>	$\rightarrow$	\<BinaryShiftedExpression\> \<BinaryTermExpression'\>
[70]	\<BinaryTermExpression'\>	$\rightarrow$	ARITHMETIC_SHIFT_LEFT \<BinaryShiftedExpression\> \<BinaryTermExpression'\>
[71]		$\rightarrow$	ARITHMETIC_SHIFT_RIGHT \<BinaryShiftedExpression\> \<BinaryTermExpression'\>
[72]		$\rightarrow$	ϵ
[73]	\<BinaryShiftedExpression\>	$\rightarrow$	\<BinaryFactorExpression\> \<BinaryShiftedExpression'\>
[74]	\<BinaryShiftedExpression'\>	$\rightarrow$	TIMES \<BinaryFactorExpression\> \<BinaryShiftedExpression'\>
[75]		$\rightarrow$	DIVIDE \<BinaryFactorExpression\> \<BinaryShiftedExpression'\>
[76]		$\rightarrow$	REMAINDER \<BinaryFactorExpression\> \<BinaryShiftedExpression'\>
[77]		$\rightarrow$	BITWISE_AND \<BinaryFactorExpression\> \<BinaryShiftedExpression'\>
[78]		$\rightarrow$	INVERSE_DIVIDE \<BinaryFactorExpression\> \<BinaryShiftedExpression'\>
[79]		$\rightarrow$	ϵ
[80]	\<BinaryFactorExpression\>	$\rightarrow$	\<UnaryExpression\> \<BinaryFactorExpression'\>

[81]	<BinaryFactorExpression'>	→	POWER <UnaryExpression> <BinaryFactorExpression'>
[82]		→	ε
[83]	<If>	→	IF <Expression> END_OF_INSTRUCTION <InstructionList> <IfEnd>
[84]	<IfEnd>	→	ELSE_IF <Expression> END_OF_INSTRUCTION <InstructionList> <IfEnd>
[85]		→	ELSE <InstructionList> END
[86]		→	END
[87]	<BuiltInFunctionCall>	→	READ_REAL LEFT_PARENTHESIS RIGHT_PARENTHESIS
[88]		→	READ_INTEGER LEFT_PARENTHESIS RIGHT_PARENTHESIS
[89]		→	INTEGER_CAST LEFT_PARENTHESIS <Expression> RIGHT_PARENTHESIS
[90]		→	REAL_CAST LEFT_PARENTHESIS <Expression> RIGHT_PARENTHESIS
[91]		→	BOOLEAN_CAST LEFT_PARENTHESIS <Expression> RIGHT_PARENTHESIS
[92]		→	PRINTLN LEFT_PARENTHESIS <Expression> RIGHT_PARENTHESIS
[93]	<FunctionCallTail>	→	LEFT_PARENTHESIS <Parameter> RIGHT_PARENTHESIS
[94]	<Parameter>	→	<Expression> <ParameterTail>
[95]		→	ε
[96]	<ParameterTail>	→	COMMA <Expression> <ParameterTail>
[97]		→	ε
[98]	<FunctionDefinition>	→	FUNCTION IDENTIFIER LEFT_PARENTHESIS <Argument> RIGHT_PARENTHESIS <InstructionList> <FunctionDefinitionEnd>
[99]	<FunctionDefinitionEnd>	→	RETURN <Expression> END
[100]		→	END
[101]	<Argument>	→	IDENTIFIER TYPE_DEFINITION <Type> <ArgumentTail>
[102]		→	ε
[103]	<ArgumentTail>	→	COMMA IDENTIFIER TYPE_DEFINITION <Type> <ArgumentTail>
[104]		→	ε

3 Ensembles First et Follow

Variable	First	Follow
<Program>	FUNCTION, WHILE, READ_REAL EPSILON_VALUE IDENTIFIER, CONST BOOLEAN_CAST, PRINTLN END_OF_INSTRUCTION READ_INTEGER, FOR INTEGER_CAST, LET, IF REAL_CAST	
<Instruction>	FUNCTION, READ_INTEGER FOR, WHILE, READ_REAL INTEGER_CAST BOOLEAN_CAST, CONST IDENTIFIER, PRINTLN, LET, IF REAL_CAST	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN

<InstructionList>	FUNCTION, WHILE, READ_REAL EPSILON_VALUE BOOLEAN_CAST, IDENTIFIER CONST, PRINTLN END_OF_INSTRUCTION READ_INTEGER, FOR INTEGER_CAST, LET, IF REAL_CAST	END, ELSE_IF, ELSE, RETURN
<InstructionListTail>	EPSILON_VALUE END_OF_INSTRUCTION	END, ELSE_IF, ELSE, RETURN
<IdentifierInstruction>	IDENTIFIER	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<IdentifierInstructionTail>	ASSIGNATION TYPE_DEFINITION, COMMA LEFT_PARENTHESIS	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<AssignationTail>	ASSIGNATION, COMMA	END, COMMA END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<ConstDefinition>	CONST	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<Block>	LET	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<Loop>	FOR, WHILE, IF	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<ForTail>	END_OF_INSTRUCTION TERNARY_ELSE	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<Type>	REAL_TYPE, BOOLEAN_TYPE INTEGER_TYPE	RIGHT_PARENTHESIS, END COMMA END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<Expression>	READ_INTEGER, INTEGER REAL, PLUS, BOOLEAN NEGATION, MINUS BITWISE_NOT, READ_REAL INTEGER_CAST LEFT_PARENTHESIS BOOLEAN_CAST, IDENTIFIER PRINTLN, REAL_CAST	RIGHT_PARENTHESIS, END COMMA END_OF_INSTRUCTION ELSE_IF, ELSE TERNARY_ELSE, RETURN
<TernaryIfExpression>	TERNARY_IF EPSILON_VALUE	RIGHT_PARENTHESIS, END COMMA END_OF_INSTRUCTION ELSE_IF, ELSE TERNARY_ELSE, RETURN
<TernaryElseExpression>	TERNARY_ELSE	RIGHT_PARENTHESIS, END COMMA END_OF_INSTRUCTION ELSE_IF, ELSE TERNARY_ELSE, RETURN

<AtomicExpression>	READ_INTEGER, INTEGER REAL, BOOLEAN, READ_REAL INTEGER_CAST BOOLEAN_CAST, IDENTIFIER PRINTLN, REAL_CAST	ARITHMETIC_SHIFT_RIGHT EQUALITY, BITWISE_AND ARITHMETIC_SHIFT_LEFT TERNARY_IF, GREATER_THAN LESS_OR_EQUALS_THAN ELSE, TERNARY_ELSE, POWER INEQUALITY, BITWISE_OR MINUS, RIGHT_PARENTHESIS BITWISE_XOR, REMAINDER COMMA END_OF_INSTRUCTION RETURN, LESS_THAN LAZY_AND, PLUS, LAZY_OR INVERSE_DIVIDE, END, TIMES ELSE_IF, DIVIDE GREATER_OR_EQUALS_THAN
<AtomicIdentifierExpression>	IDENTIFIER	ARITHMETIC_SHIFT_RIGHT EQUALITY, BITWISE_AND ARITHMETIC_SHIFT_LEFT TERNARY_IF, GREATER_THAN LESS_OR_EQUALS_THAN ELSE, TERNARY_ELSE, POWER INEQUALITY, BITWISE_OR MINUS, RIGHT_PARENTHESIS BITWISE_XOR, REMAINDER COMMA END_OF_INSTRUCTION RETURN, LESS_THAN LAZY_AND, PLUS, LAZY_OR INVERSE_DIVIDE, END, TIMES ELSE_IF, DIVIDE GREATER_OR_EQUALS_THAN
<AtomicIdentifierExpressionTail>	EPSILON_VALUE LEFT_PARENTHESIS	ARITHMETIC_SHIFT_RIGHT EQUALITY, BITWISE_AND ARITHMETIC_SHIFT_LEFT TERNARY_IF, GREATER_THAN LESS_OR_EQUALS_THAN ELSE, TERNARY_ELSE, POWER INEQUALITY, BITWISE_OR MINUS, RIGHT_PARENTHESIS BITWISE_XOR, REMAINDER COMMA END_OF_INSTRUCTION RETURN, LESS_THAN LAZY_AND, PLUS, LAZY_OR INVERSE_DIVIDE, END, TIMES ELSE_IF, DIVIDE GREATER_OR_EQUALS_THAN

\<UnaryExpression\>	INTEGER, REAL, BOOLEAN NEGATION, BITWISE_NOT READ_REAL LEFT_PARENTHESIS IDENTIFIER, BOOLEAN_CAST PRINTLN, READ_INTEGER PLUS, MINUS, INTEGER_CAST REAL_CAST	ARITHMETIC_SHIFT_RIGHT EQUALITY, BITWISE_AND ARITHMETIC_SHIFT_LEFT TERNARY_IF, GREATER_THAN LESS_OR_EQUALS_THAN ELSE, TERNARY_ELSE, POWER INEQUALITY, BITWISE_OR MINUS, RIGHT_PARENTHESIS BITWISE_XOR, REMAINDER COMMA END_OF_INSTRUCTION RETURN, LESS_THAN LAZY_AND, PLUS, LAZY_OR INVERSE_DIVIDE, END, TIMES ELSE_IF, DIVIDE GREATER_OR_EQUALS_THAN
\<UnaryBitwiseNotExpression\>	READ_INTEGER, INTEGER REAL, PLUS, BOOLEAN, MINUS BITWISE_NOT, READ_REAL INTEGER_CAST LEFT_PARENTHESIS BOOLEAN_CAST, IDENTIFIER PRINTLN, REAL_CAST	ARITHMETIC_SHIFT_RIGHT EQUALITY, BITWISE_AND ARITHMETIC_SHIFT_LEFT TERNARY_IF, GREATER_THAN LESS_OR_EQUALS_THAN ELSE, TERNARY_ELSE, POWER INEQUALITY, BITWISE_OR MINUS, RIGHT_PARENTHESIS BITWISE_XOR, REMAINDER COMMA END_OF_INSTRUCTION RETURN, LESS_THAN LAZY_AND, PLUS, LAZY_OR INVERSE_DIVIDE, END, TIMES ELSE_IF, DIVIDE GREATER_OR_EQUALS_THAN
\<UnaryMinusPlusExpression\>	READ_INTEGER, INTEGER REAL, PLUS, BOOLEAN, MINUS READ_REAL, INTEGER_CAST LEFT_PARENTHESIS BOOLEAN_CAST, IDENTIFIER PRINTLN, REAL_CAST	ARITHMETIC_SHIFT_RIGHT EQUALITY, BITWISE_AND ARITHMETIC_SHIFT_LEFT TERNARY_IF, GREATER_THAN LESS_OR_EQUALS_THAN ELSE, TERNARY_ELSE, POWER INEQUALITY, BITWISE_OR MINUS, RIGHT_PARENTHESIS BITWISE_XOR, REMAINDER COMMA END_OF_INSTRUCTION RETURN, LESS_THAN LAZY_AND, PLUS, LAZY_OR INVERSE_DIVIDE, END, TIMES ELSE_IF, DIVIDE GREATER_OR_EQUALS_THAN

<UnaryAtomicExpression>	READ_INTEGER, INTEGER REAL, BOOLEAN, READ_REAL INTEGER_CAST LEFT_PARENTHESIS BOOLEAN_CAST, IDENTIFIER PRINTLN, REAL_CAST	ARITHMETIC_SHIFT_RIGHT EQUALITY, BITWISE_AND ARITHMETIC_SHIFT_LEFT TERNARY_IF, GREATER_THAN LESS_OR_EQUALS_THAN ELSE, TERNARY_ELSE, POWER INEQUALITY, BITWISE_OR MINUS, RIGHT_PARENTHESIS BITWISE_XOR, REMAINDER COMMA END_OF_INSTRUCTION RETURN, LESS_THAN LAZY_AND, PLUS, LAZY_OR INVERSE_DIVIDE, END, TIMES ELSE_IF, DIVIDE GREATER_OR_EQUALS_THAN
<BinaryExpression>	READ_INTEGER, INTEGER REAL, PLUS, BOOLEAN NEGATION, MINUS BITWISE_NOT, READ_REAL INTEGER_CAST BOOLEAN_CAST LEFT_PARENTHESIS IDENTIFIER, PRINTLN REAL_CAST	TERNARY_IF RIGHT_PARENTHESIS, END COMMA END_OF_INSTRUCTION ELSE_IF, ELSE TERNARY_ELSE, RETURN
<BinaryExpression'>	LAZY_OR, EPSILON_VALUE	TERNARY_IF RIGHT_PARENTHESIS, END COMMA END_OF_INSTRUCTION ELSE_IF, ELSE TERNARY_ELSE, RETURN
<BinaryLazyOrExpression>	READ_INTEGER, INTEGER REAL, PLUS, BOOLEAN NEGATION, MINUS BITWISE_NOT, READ_REAL INTEGER_CAST LEFT_PARENTHESIS BOOLEAN_CAST, IDENTIFIER PRINTLN, REAL_CAST	TERNARY_IF, LAZY_OR RIGHT_PARENTHESIS, END COMMA END_OF_INSTRUCTION ELSE_IF, ELSE TERNARY_ELSE, RETURN
<BinaryLazyOrExpression'>	LAZY_AND, EPSILON_VALUE	TERNARY_IF, LAZY_OR RIGHT_PARENTHESIS, END COMMA END_OF_INSTRUCTION ELSE_IF, ELSE TERNARY_ELSE, RETURN
<BinaryLazyAndExpression>	READ_INTEGER, INTEGER REAL, PLUS, BOOLEAN NEGATION, MINUS BITWISE_NOT, READ_REAL INTEGER_CAST BOOLEAN_CAST LEFT_PARENTHESIS IDENTIFIER, PRINTLN REAL_CAST	LAZY_AND, TERNARY_IF LAZY_OR RIGHT_PARENTHESIS, END COMMA END_OF_INSTRUCTION ELSE_IF, ELSE TERNARY_ELSE, RETURN

\<BinaryLazyAndExpression'\>	EQUALITY, INEQUALITY GREATER_THAN EPSILON_VALUE LESS_OR_EQUALS_THAN GREATER_OR_EQUALS_THAN LESS_THAN	LAZY_AND, TERNARY_IF LAZY_OR RIGHT_PARENTHESIS, END COMMA END_OF_INSTRUCTION ELSE_IF, ELSE TERNARY_ELSE, RETURN
\<BinaryNumericExpression\>	READ_INTEGER, INTEGER REAL, PLUS, BOOLEAN NEGATION, MINUS BITWISE_NOT, READ_REAL INTEGER_CAST LEFT_PARENTHESIS BOOLEAN_CAST, IDENTIFIER PRINTLN, REAL_CAST	EQUALITY, TERNARY_IF GREATER_THAN RIGHT_PARENTHESIS, COMMA END_OF_INSTRUCTION LESS_OR_EQUALS_THAN ELSE, TERNARY_ELSE, RETURN LESS_THAN, LAZY_AND INEQUALITY, LAZY_OR, END ELSE_IF GREATER_OR_EQUALS_THAN
\<BinaryNumericExpression'\>	BITWISE_OR, PLUS, MINUS BITWISE_XOR EPSILON_VALUE	EQUALITY, TERNARY_IF GREATER_THAN RIGHT_PARENTHESIS, COMMA END_OF_INSTRUCTION LESS_OR_EQUALS_THAN ELSE, TERNARY_ELSE, RETURN LESS_THAN, LAZY_AND INEQUALITY, LAZY_OR, END ELSE_IF GREATER_OR_EQUALS_THAN
\<BinaryTermExpression\>	READ_INTEGER, INTEGER REAL, PLUS, BOOLEAN NEGATION, MINUS BITWISE_NOT, READ_REAL INTEGER_CAST BOOLEAN_CAST LEFT_PARENTHESIS IDENTIFIER, PRINTLN REAL_CAST	EQUALITY, TERNARY_IF RIGHT_PARENTHESIS GREATER_THAN BITWISE_XOR, COMMA END_OF_INSTRUCTION LESS_OR_EQUALS_THAN ELSE, TERNARY_ELSE, RETURN LESS_THAN, LAZY_AND INEQUALITY, BITWISE_OR PLUS, LAZY_OR, MINUS, END ELSE_IF GREATER_OR_EQUALS_THAN
\<BinaryTermExpression'\>	ARITHMETIC_SHIFT_RIGHT ARITHMETIC_SHIFT_LEFT EPSILON_VALUE	EQUALITY, TERNARY_IF RIGHT_PARENTHESIS GREATER_THAN BITWISE_XOR, COMMA END_OF_INSTRUCTION LESS_OR_EQUALS_THAN ELSE, TERNARY_ELSE, RETURN LESS_THAN, LAZY_AND INEQUALITY, BITWISE_OR PLUS, LAZY_OR, MINUS, END ELSE_IF GREATER_OR_EQUALS_THAN

<BinaryShiftedExpression>	READ_INTEGER, INTEGER REAL, PLUS, BOOLEAN NEGATION, MINUS BITWISE_NOT, READ_REAL INTEGER_CAST LEFT_PARENTHESIS BOOLEAN_CAST, IDENTIFIER PRINTLN, REAL_CAST	ARITHMETIC_SHIFT_RIGHT EQUALITY, TERNARY_IF ARITHMETIC_SHIFT_LEFT RIGHT_PARENTHESIS GREATER_THAN BITWISE_XOR, COMMA END_OF_INSTRUCTION LESS_OR_EQUALS_THAN ELSE, TERNARY_ELSE, RETURN LESS_THAN, LAZY_AND INEQUALITY, PLUS BITWISE_OR, LAZY_OR, MINUS END, ELSE_IF GREATER_OR_EQUALS_THAN
<BinaryShiftedExpression'>	BITWISE_AND INVERSE_DIVIDE REMAINDER, TIMES EPSILON_VALUE, DIVIDE	ARITHMETIC_SHIFT_RIGHT EQUALITY, TERNARY_IF ARITHMETIC_SHIFT_LEFT RIGHT_PARENTHESIS GREATER_THAN BITWISE_XOR, COMMA END_OF_INSTRUCTION LESS_OR_EQUALS_THAN ELSE, TERNARY_ELSE, RETURN LESS_THAN, LAZY_AND INEQUALITY, PLUS BITWISE_OR, LAZY_OR, MINUS END, ELSE_IF GREATER_OR_EQUALS_THAN
<BinaryFactorExpression>	READ_INTEGER, INTEGER REAL, PLUS, BOOLEAN NEGATION, MINUS BITWISE_NOT, READ_REAL INTEGER_CAST BOOLEAN_CAST LEFT_PARENTHESIS IDENTIFIER, PRINTLN REAL_CAST	ARITHMETIC_SHIFT_RIGHT EQUALITY, BITWISE_AND ARITHMETIC_SHIFT_LEFT TERNARY_IF, GREATER_THAN LESS_OR_EQUALS_THAN ELSE, TERNARY_ELSE INEQUALITY, BITWISE_OR MINUS, RIGHT_PARENTHESIS BITWISE_XOR, REMAINDER COMMA END_OF_INSTRUCTION RETURN, LESS_THAN LAZY_AND, PLUS, LAZY_OR INVERSE_DIVIDE, END, TIMES ELSE_IF, DIVIDE GREATER_OR_EQUALS_THAN

⟨BinaryFactorExpression'⟩	POWER, EPSILON_VALUE	ARITHMETIC_SHIFT_RIGHT EQUALITY, BITWISE_AND ARITHMETIC_SHIFT_LEFT TERNARY_IF, GREATER_THAN LESS_OR_EQUALS_THAN ELSE, TERNARY_ELSE INEQUALITY, BITWISE_OR MINUS, RIGHT_PARENTHESIS BITWISE_XOR, REMAINDER COMMA END_OF_INSTRUCTION RETURN, LESS_THAN LAZY_AND, PLUS, LAZY_OR INVERSE_DIVIDE, END, TIMES ELSE_IF, DIVIDE GREATER_OR_EQUALS_THAN
⟨If⟩	IF	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
⟨IfEnd⟩	END, ELSE_IF, ELSE	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
⟨BuiltInFunctionCall⟩	READ_INTEGER, READ_REAL INTEGER_CAST BOOLEAN_CAST, PRINTLN REAL_CAST	ARITHMETIC_SHIFT_RIGHT EQUALITY, BITWISE_AND TERNARY_IF ARITHMETIC_SHIFT_LEFT GREATER_THAN LESS_OR_EQUALS_THAN TERNARY_ELSE, ELSE, POWER INEQUALITY, BITWISE_OR MINUS, RIGHT_PARENTHESIS REMAINDER, BITWISE_XOR COMMA END_OF_INSTRUCTION RETURN, LESS_THAN LAZY_AND, PLUS, LAZY_OR INVERSE_DIVIDE, TIMES, END ELSE_IF, DIVIDE GREATER_OR_EQUALS_THAN
⟨FunctionCallTail⟩	LEFT_PARENTHESIS	ARITHMETIC_SHIFT_RIGHT EQUALITY, BITWISE_AND TERNARY_IF ARITHMETIC_SHIFT_LEFT GREATER_THAN LESS_OR_EQUALS_THAN TERNARY_ELSE, ELSE, POWER INEQUALITY, BITWISE_OR MINUS, RIGHT_PARENTHESIS REMAINDER, BITWISE_XOR COMMA END_OF_INSTRUCTION RETURN, LESS_THAN LAZY_AND, PLUS, LAZY_OR INVERSE_DIVIDE, TIMES, END ELSE_IF, DIVIDE GREATER_OR_EQUALS_THAN

<Parameter>	INTEGER, REAL, BOOLEAN NEGATION, BITWISE_NOT READ_REAL, EPSILON_VALUE BOOLEAN_CAST, IDENTIFIER LEFT_PARENTHESIS PRINTLN, READ_INTEGER PLUS, MINUS, INTEGER_CAST REAL_CAST	RIGHT_PARENTHESIS
<ParameterTail>	COMMA, EPSILON_VALUE	RIGHT_PARENTHESIS
<FunctionDefinition>	FUNCTION	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<FunctionDefinitionEnd>	END, RETURN	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<Argument>	EPSILON_VALUE IDENTIFIER	RIGHT_PARENTHESIS
<ArgumentTail>	COMMA, EPSILON_VALUE	RIGHT_PARENTHESIS

4 Action Table

LL(1) parsing table (terminals as rows, non-terminals as columns). Cell values are production rule numbers.

Terminal	⟨Program⟩	⟨Instruction⟩	⟨InstructionList⟩	⟨InstructionListTail⟩	⟨IdentifierInstruction⟩	⟨IdentifierInstructionTail⟩	⟨AssignationTail⟩	⟨ConstDefinition⟩	⟨Block⟩	⟨Loop⟩	⟨ForTail⟩	⟨Type⟩	⟨Expression⟩	⟨TernaryIfExpression⟩	⟨TernaryElseExpression⟩	⟨AtomicExpression⟩	⟨AtomicIdentifierExpression⟩	⟨AtomicIdentifierExpressionTail⟩	⟨UnaryExpression⟩	⟨UnaryBitwiseNotExpression⟩	⟨UnaryMinusPlusExpression⟩	⟨UnaryAtomicExpression⟩	⟨BinaryExpression⟩	⟨BinaryExpression'⟩	⟨BinaryLazyOrExpression'⟩	⟨BinaryLazyAndExpression'⟩	⟨BinaryNumericExpression⟩	⟨BinaryNumericExpression'⟩	⟨BinaryTermExpression⟩	⟨BinaryTermExpression'⟩	⟨BinaryShiftedExpression⟩	⟨BinaryShiftedExpression'⟩	⟨BinaryFactorExpression⟩	⟨BinaryFactorExpression'⟩	⟨If⟩	⟨IfEnd⟩	⟨BuiltInFunctionCall⟩	⟨FunctionCallTail⟩	⟨Parameter⟩	⟨ParameterTail⟩	⟨FunctionDefinition⟩	⟨FunctionDefinitionEnd⟩	⟨Argument⟩	⟨ArgumentTail⟩
RETURN			9	11										30				39						51	54	62		68		72		79		82								99		
FUNCTION	1	7	8																																						98			
PRINTLN	1	6	8										28			36			41	43	46	49	52				63		69		73		80				92		94					
BOOLEAN_CAST	1	6	8										28			36			41	43	46	49	52				63		69		73		80				91		94					
REAL_CAST	1	6	8										28			36			41	43	46	49	52				63		69		73		80				90		94					
INTEGER_CAST	1	6	8										28			36			41	43	46	49	52				63		69		73		80				89		94					
READ_INTEGER	1	6	8										28			36			41	43	46	49	52				63		69		73		80				88		94					
READ_REAL	1	6	8										28			36			41	43	46	49	52				63		69		73		80				87		94					
INTEGER_TYPE												25																																
REAL_TYPE												26																																
BOOLEAN_TYPE												27																																
LET	1	4	8						19																																			
ELSE			9	11										30				39						51	54	62		68		72		79		82		84								
ELSE_IF			9	11										30				39						51	54	62		68		72		79		82		85								
IF	1	5	8							20																									83									
POWER																		39																81										
INVERSE_DIVIDE																		39														78		82										
BITWISE_AND																		39														77		82										
REMAINDER																		39														76		82										
DIVIDE																		39														75		82										
TIMES																		39														74		82										
ARITHMETIC_SHIFT_RIGHT																		39												71		79		82										
ARITHMETIC_SHIFT_LEFT																		39												70		79		82										
BITWISE_XOR																		39										67		72		79		82										
BITWISE_OR																		39										66		72		79		82										
INEQUALITY																		39								61		68		72		79		82										
EQUALITY																		39								60		68		72		79		82										
LESS_OR_EQUALS_THAN																		39								59		68		72		79		82										
GREATER_OR_EQUALS_THAN																		39								58		68		72		79		82										
LESS_THAN																		39								57		68		72		79		82										
GREATER_THAN																		39								56		68		72		79		82										
LAZY_AND																		39							53	62		68		72		79		82										
LAZY_OR																		39						50	54	62		68		72		79		82										
RIGHT_PARENTHESIS														30				39						51	54	62		68		72		79		82					95	97			102	104
LEFT_PARENTHESIS						15							28			38		38	41	43	46	49	52				63		69		73		80					93	94					
PLUS													28					39	41	43	44							64		72		79		82					94					
MINUS													28					39	41	43	45							65		72		79		82					94					
BITWISE_NOT													28						41	40																			94					
NEGATION													28						41	43	46	47																	94					
BOOLEAN													28			33			41	43	46	49	52				63		69		73		80						94					
REAL													28			34			41	43	46	49	52				63		69		73		80						94					
INTEGER													28			35			41	43	46	49	52				63		69		73		80						94					
TERNARY_IF														29																														
COMMA						13	17							30				39						51	54	62		68		72		79		82						96				103
TERNARY_ELSE											24			30	31			39						51	54	62		68		72		79		82										
FOR	1	5	8							22	24																																	
WHILE	1	5	8							21																																		
INTEGER_TYPE												25																																
REAL_TYPE												26																																
BOOLEAN_TYPE												27																																
END_OF_INSTRUCTION	1		9	10							23			30				39						51	54	62		68		72		79		82										
ASSIGNATION						13	16																																					
END			9	11										30				39						51	54	62		68		72		79		82		86						100		
CONST	1	3	8					18																																				
TYPE_DEFINITION						14																																						
IDENTIFIER	1	2	8		12								28			32	37		41	43	46	49	52				63		69		73		80						94				101	
EPSILON_VALUE	1		9	11										30				39						51	54	62		68		72		79		82					95	97			102	104