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

Simon Picard Arnaud Rosette

18 décembre 2014

1 Introduction

Dans cette partie du projet, il nous est demandé de modifier la grammaire donnée dans l'énoncé afin de rendre celle-ci LL(1). Il faut donc appliquer toute une série de règles et d'algorithmes pour rendre cette grammaire moins ambigüe et LL(1), dans le but de construire "l'action table" à partir des ensembles first et follow. L'inclusion des fonctions dans la grammaire fait partie d'un bonus que nous avons décidé de réaliser.

2 Transformation de la grammaire donnée

2.1 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), \sim (not bit à bit), + et -. Les opérateurs + et - sont également des opérateurs binaires.

2.2 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 ambigüe. 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.

2.3 Suppression des récursions 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. Cette étape rend tout opérateur associatif à droite si l'on ne fait rien pour résoudre ce problème lors de l'analyse sémantique.

2.4 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.

2.5 Les variables <Instruction> et <InstructionList>

Dans la grammaire donnée, à chaque fois que la variable <Instruction> se trouvait dans la partie droite d'une règle de production excepté lorsque la partie gauche de la règle était <InstructionList>, il y avait une autre règle de production pour cette même partie gauche où «Instruction» était remplacé par <InstructionList>. Par exemple, <If> \rightarrow <Empty> <InstructionList> <IfEnd> et <If>→<Expression> <Empty> <Instruction> <IfEnd>. Ceci permettant de ne pas mettre de END OF INSTRUCTION à la fin d'une «Instruction» lorsque le corps d'un block (ici, le corps du "if') ne contenait qu'une seule <Instruction>. Le problème posé par le doublement systématique des règles de production contenant des instructions (une règle pour <Instruction> et une autre pour <Instruction-List>) est que ce n'est pas factorisé à gauche et que le first de <InstructionList> peut être <Instruction>. Afin de résoudre ce problème, nous avons décidé de ne plus utilisé que la variable < InstructionList> dans les autres règles de production. Ainsi, on évite le doublement des règles de production. Une <InstructionList> est une liste d'<Instruction> séparées par des END OF INSTRUCTION, avec la dernière <Instruction> qui n'est pas nécessairement suivie d'un END OF INSTRUCTION. Ceci permet d'avoir un programme du style : if(a>b);a=10 end; Donc seule une <InstructionList> permet de produire des < Instruction>. De plus, vu qu'une < InstructionList> peut contenir des instructions vide (une instruction contenant seulement END_OF_INSTRUCTION) la variable <Empty> n'est plus nécessaire car celle-ci servait uniquement à indiquer qu'il fallait au moins un terminal END OF INSTRUCTION.

2.6 Fonctions

Les fonctions amènent deux types d'instructions en plus : les définitions de fonction et les appels de fonction. Les appels sont considérés comme des expressions atomiques afin de pourvoir mettre des appels de fonction au sein d'une expression. Par exemple, a=foo()+5; De plus, il faut qu'une fonction puisse être appelée sans avoir besoin de faire une assignation ou un autre type d'instruction car une fonction peut ne rien retourner et simplement avoir un effet de bord. Par exemple, la fonction println est une fonction qui ne retourne rien mais produit un effet de bord qui est l'affichage dans le terminal. Il faut donc pouvoir faire, par exemple, println(a). C'est pour cela qu'il faut qu'un appel de fonction soit une instruction à part entière.

2.7 Instructions commencant par un identifier

Plusieurs instructions commencent par un identifier. Il s'agit des assignations, des déclarations de variables et des appels de fonctions. Afin que la grammaire soit LL(1) il faut factoriser ces instructions. C'est pour cela que la variable «IdentifierInstruction» a été créée. Celle-ci représente une instruction qui commence par un identifier. La variable «IdentifierInstructionTail» a été également créée afin de distinguer les différents types d'instructions qui commencent par un identifier. La distinction est après aisément faite car lorsque le symbole suivant l'identifier est "=", on sait que c'est une assignation, lorsque ce symbole est ":", on sait que c'est une déclaration de variable et lorsque que ce symbole est "(", on sait que c'est un appel de fonction.

3 Grammaire

Voici la grammaire donnée dans l'énoncé et transformée en grammaire LL(1). A chaque règle de production correspond un numéro qui identifie cette règle.

```
[1]
       <Program>
                                                <InstructionList>
[2]
       <Instruction>
                                                <IdentifierInstruction>
[3]
                                                <ConstDefinition>
[4]
                                                <Block>
[5]
                                                <Loop>
[6]
                                                <BuiltInFunctionCall>
[7]
                                                <FunctionDefinition>
[8]
                                                <Instruction> <InstructionListTail>
       <InstructionList>
[9]
                                                <InstructionListTail>
[10]
                                                END OF INSTRUCTION <InstructionList>
       <InstructionListTail>
[11]
[12]
       <IdentifierInstruction>
                                                IDENTIFIER < IdentifierInstructionTail>
```

[13] [14] [15]	$<\! Identifier Instruction Tail >$	$\begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \end{array}$	<assignationtail> TYPE_DEFINITION <type> <functioncalltail></functioncalltail></type></assignationtail>
[16] [17]	<assignationtail></assignationtail>	$\stackrel{\prime}{\rightarrow}$	ASSIGNATION <expression> COMMA IDENTIFIER <assignationtail> COMMA</assignationtail></expression>
[18] [19]	<constdefinition> <block></block></constdefinition>	$\begin{array}{c} \rightarrow \\ \rightarrow \end{array}$	<pre><expression> CONST IDENTIFIER <assignationtail> LET IDENTIFIER <assignationtail></assignationtail></assignationtail></expression></pre>
[20] [21]	<loop></loop>	$\overset{\rightarrow}{\rightarrow}$	END_OF_INSTRUCTION <instructionlist> END <if> WHILE <expression> END_OF_INSTRUCTION</expression></if></instructionlist>
[22]		\rightarrow	<pre><instructionlist> END FOR IDENTIFIER ASSIGNATION <expression> TEDNARY FLSE <empression> <entil></entil></empression></expression></instructionlist></pre>
[23] [24]	<fortail></fortail>	$\overset{\rightarrow}{\rightarrow}$	TERNARY_ELSE <expression> <fortail> END_OF_INSTRUCTION <instructionlist> END TERNARY ELSE <expression></expression></instructionlist></fortail></expression>
[25]	<type></type>	\rightarrow	END_OF_INSTRUCTION <instructionlist> END BOOLEAN_TYPE REAL_TYPE</instructionlist>
[26] [27] [28]	<expression></expression>	$\begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \end{array}$	INTEGER_TYPE <binaryexpression> <ternaryifexpression></ternaryifexpression></binaryexpression>
[29]	<ternaryifexpression></ternaryifexpression>	\rightarrow	TERNARY_IF <expression> <ternaryelseexpression></ternaryelseexpression></expression>
[30] [31]	<ternaryelseexpression></ternaryelseexpression>	\rightarrow	ε TERNARY_ELSE <expression></expression>
[32] [33] [34]	<atomicexpression></atomicexpression>	$\begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \end{array}$	<atomicidentifierexpression> INTEGER REAL</atomicidentifierexpression>
[35] [36]		$\overset{\rightarrow}{\rightarrow}$	BOOLEAN <builtinfunctioncall></builtinfunctioncall>
[37] [38]	<atomicidentifierexpression></atomicidentifierexpression> AtomicIdentifierExpressionTail>	\rightarrow	IDENTIFIER <atomicidentifierexpressiontail> <functioncalltail></functioncalltail></atomicidentifierexpressiontail>
[39] [40]	< UnaryExpression>	$\stackrel{ ightarrow}{ ightarrow}$	ϵ NEGATION < Unary Expression >
[41] [42]	$<\!$	$\overset{\rightarrow}{\rightarrow}$	<unarybitwisenotexpression></unarybitwisenotexpression>BITWISE_NOT <unarybitwisenotexpression></unarybitwisenotexpression>
[43] [44]	$<\!\!\text{UnaryMinusPlusExpression}\!\!>$	$\stackrel{\rightarrow}{\rightarrow}$	<unaryminusplusexpression></unaryminusplusexpression>MINUS <unaryminusplusexpression></unaryminusplusexpression>
[45] [46] [47]	<Unary $AtomicExpression>$	$\begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \end{array}$	PLUS <unaryminusplusexpression> <unaryatomicexpression> <atomicexpression></atomicexpression></unaryatomicexpression></unaryminusplusexpression>
[48]		\rightarrow	LEFT_PARENTHESIS < Expression > RIGHT_PARENTHESIS
[49]	<binaryexpression></binaryexpression>	\rightarrow	<binarylazyorexpression> <binaryexpression'></binaryexpression'></binarylazyorexpression>
[50] [51]	<binaryexpression'></binaryexpression'>	\rightarrow	$ \begin{array}{l} {\rm LAZY_OR} < {\rm BinaryLazyOrExpression}> \\ < {\rm BinaryExpression'}> \\ \epsilon \end{array} $
[52]	$<\!$	$\stackrel{'}{\rightarrow}$	<pre><binarylazyandexpression> <binarylazyorexpression'></binarylazyorexpression'></binarylazyandexpression></pre>
[53]	$<\!$	\rightarrow	$\begin{split} & LAZY_AND < BinaryLazyAndExpression> \\ & < BinaryLazyOrExpression'> \end{split}$
[54] [55]	$<\!$	$\overset{\rightarrow}{\rightarrow}$	ϵ <binarynumericexpression> <binarylazyandexpression'></binarylazyandexpression'></binarynumericexpression>
[56]	$<\!$	\rightarrow	<pre><binarylazyandexpression> GREATER_THAN <binarynumericexpression> <binarylazyandexpression'></binarylazyandexpression'></binarynumericexpression></binarylazyandexpression></pre>

[57]		\rightarrow	$LESS_THAN < Binary Numeric Expression>$
[58]		\rightarrow	<binarylazyandexpression'> GREATER OR EQUALS THAN</binarylazyandexpression'>
[90]		/	<pre><binarynumericexpression></binarynumericexpression></pre>
			<binarylazyandexpression'></binarylazyandexpression'>
[59]		\rightarrow	LESS_OR_EQUALS_THAN
			<binarynumericexpression></binarynumericexpression>
			$<\!$
[60]		\rightarrow	${\bf EQUALITY} < \!$
			$<\!$
[61]		\rightarrow	INEQUALITY <binarynumericexpression></binarynumericexpression>
[60]			<binarylazyandexpression'></binarylazyandexpression'>
[62]	D' M ' D ' '	\rightarrow	€
[63]	<BinaryNumericExpression $>$	\rightarrow	<binarytermexpression></binarytermexpression>
[64]	<pin ?<="" a="" ani="" f.="" m.="" magai="" n.="" td=""><td>,</td><td><pre><binarynumericexpression'> PLUS</binarynumericexpression'></pre></td></pin>	,	<pre><binarynumericexpression'> PLUS</binarynumericexpression'></pre>
[64]	<BinaryNumericExpression' $>$	\rightarrow	PLUS <binarytermexpression> <binarynumericexpression'></binarynumericexpression'></binarytermexpression>
[65]		\rightarrow	MINUS <binarytermexpression></binarytermexpression>
[00]			
[66]		\rightarrow	BITWISE OR <binarytermexpression></binarytermexpression>
[00]		,	<pre><binarynumericexpression'></binarynumericexpression'></pre>
[67]		\rightarrow	BITWISE XOR <binarytermexpression></binarytermexpression>
. ,			<binarynumericexpression'></binarynumericexpression'>
[68]		\rightarrow	ϵ
[69]	<BinaryTermExpression $>$	\rightarrow	<BinaryShiftedExpression $>$
			<binarytermexpression'></binarytermexpression'>
[70]	$<\!$	\rightarrow	ARITHMETIC_SHIFT_LEFT
			<binaryshiftedexpression></binaryshiftedexpression>
F 3			<binarytermexpression'></binarytermexpression'>
[71]		\rightarrow	ARITHMETIC_SHIFT_RIGHT
			<binaryshiftedexpression></binaryshiftedexpression>
[79]		,	<binarytermexpression'></binarytermexpression'>
[72] [73]	<binaryshiftedexpression></binaryshiftedexpression>	\rightarrow \rightarrow	ϵ <binaryfactorexpression></binaryfactorexpression>
[19]	\Dinary5miredExpression/		<pre><binarypactorexpression'></binarypactorexpression'></pre>
[74]	<binaryshiftedexpression'></binaryshiftedexpression'>	\rightarrow	TIMES <binaryfactorexpression></binaryfactorexpression>
[• +]	(Binary Simred Empression)	,	<binaryshiftedexpression'></binaryshiftedexpression'>
[75]		\rightarrow	DIVIDE <binaryfactorexpression></binaryfactorexpression>
			<binaryshiftedexpression'></binaryshiftedexpression'>
[76]		\rightarrow	REMAINDER < BinaryFactorExpression>
			<BinaryShiftedExpression' $>$
[77]		\rightarrow	BITWISE_AND <binaryfactorexpression></binaryfactorexpression>
[- 0]			<binaryshiftedexpression'></binaryshiftedexpression'>
[78]		\rightarrow	INVERSE_DIVIDE <binaryfactorexpression></binaryfactorexpression>
[70]			<binaryshiftedexpression'></binaryshiftedexpression'>
[79]	< Din a my Ea at an Even neggion >	\rightarrow	€ «UnampEumpagian»
[80]	<binaryfactorexpression></binaryfactorexpression>	\rightarrow	<unaryexpression> <binaryfactorexpression'></binaryfactorexpression'></unaryexpression>
[81]	<binaryfactorexpression'></binaryfactorexpression'>	\rightarrow	POWER <unaryexpression></unaryexpression>
[01]	/Dinaryr actor Dxpression /	\rightarrow	SinaryFactorExpression'>
[82]		\rightarrow	€
[83]	<if></if>	$\stackrel{'}{ ightarrow}$	IF <expression> END OF INSTRUCTION</expression>
[-2]			<instructionlist> <ifend></ifend></instructionlist>
[84]	<IfEnd $>$	\rightarrow	ELSE_IF < Expression > END_OF_INSTRUCTION
			<instructionlist> <ifend></ifend></instructionlist>
[85]		\rightarrow	ELSE < InstructionList > END
[86]		\rightarrow	END

[87]	<BuiltInFunctionCall $>$	\rightarrow	READ REAL LEFT PARENTHESIS
			RIGHT PARENTHESIS
[88]		\rightarrow	READ_INTEGER LEFT_PARENTHESIS
			RIGHT PARENTHESIS
[89]		\rightarrow	INTEGER CAST LEFT PARENTHESIS < Expression >
			RIGHT PARENTHESIS
[90]		\rightarrow	REAL_CAST LEFT_PARENTHESIS < Expression>
. ,			RIGHT PARENTHESIS
[91]		\rightarrow	BOOLEAN CAST LEFT PARENTHESIS < Expression >
. ,			RIGHT PARENTHESIS
[92]		\rightarrow	PRINTLN LEFT PARENTHESIS < Expression>
. ,			RIGHT PARENTHESIS
[93]	<FunctionCallTail $>$	\rightarrow	LEFT PARENTHESIS < Parameter>
			RIGHT PARENTHESIS
[94]	<parameter></parameter>	\rightarrow	<expression> <parametertail></parametertail></expression>
[95]		\rightarrow	ϵ
[96]	<ParameterTail $>$	\rightarrow	${ m COMMA}$ < ${ m Expression} > { m < Parameter Tail} >$
[97]		\rightarrow	ϵ
[98]	<FunctionDefinition $>$	\rightarrow	FUNCTION IDENTIFIER LEFT_PARENTHESIS
			$<$ Argument $>$ RIGHT_PARENTHESIS
			<InstructionList $>$ $<$ FunctionDefinitionEnd $>$
[99]	<FunctionDefinitionEnd $>$	\rightarrow	RETURN < Expression > END
[100]		\rightarrow	END
[101]	<Argument $>$	\rightarrow	IDENTIFIER TYPE_DEFINITION < Type>
			<argumenttail></argumenttail>
[102]		\rightarrow	ϵ
[103]	<ArgumentTail $>$	\rightarrow	${\tt COMMA~IDENTIFIER~TYPE_DEFINITION < Type>}$
			<argumenttail></argumenttail>
[104]		\rightarrow	ϵ

4 Ensembles First et Follow

Variable	First	Follow
<program></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></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></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></assignationtail>	ASSIGNATION, COMMA	END, COMMA END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<constdefinition></constdefinition>	CONST	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<block></block>	LET	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<loop></loop>	FOR, WHILE, IF	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<fortail></fortail>	END_OF_INSTRUCTION TERNARY_ELSE	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<type></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
$<\!$	TERNARY_IF EPSILON_VALUE	RIGHT_PARENTHESIS, END COMMA END_OF_INSTRUCTION ELSE_IF, ELSE TERNARY ELSE, RETURN
$<\!$	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

ARITHMETIC_SHIFT_RIGHT EQUALITY, BITWISE_AND ARITHMETIC_SHIFT_LEFT TERNARY_IF, GREATER_THAN LESS OR EQUALS THAN
ARITHMETIC_SHIFT_LEFT TERNARY_IF, GREATER_THAN
TERNARY_IF, GREATER_THAN
ELSE, TERNARY_ELSE, POWER
INEQUALITY, BITWISE_OR
<pre><atomicidentifierexpression></atomicidentifierexpression></pre> <pre>IDENTIFIER</pre> MINUS, RIGHT_PARENTHESIS PITULES VOD DEMAINDED Output Output
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
ARITHMETIC_SHIFT_RIGHT
EQUALITY, BITWISE_AND
ARITHMETIC_SHIFT_LEFT TERNARY IF, GREATER THAN
LESS OR EQUALS THAN
ELSS_OR_EQUALS_THAN ELSE, TERNARY ELSE, POWER
INEQUALITY, BITWISE_OR MINUS, RIGHT PARENTHESIS
<atomicidentifierexpressiontail> EPSILON_VALUE MINUS, RIGHT_PARENTHESIS BITWISE XOR, REMAINDER</atomicidentifierexpressiontail>
COMMA COMMA
END OF INSTRUCTION
RETURN, LESS THAN
LAZY AND, PLUS, LAZY OR
INVERSE DIVIDE, END, TIMES
ELSE IF, DIVIDE
GREATER OR EQUALS THAN
ARITHMETIC SHIFT RIGHT
EQUALITY, BITWISE AND
ARITHMETIC SHIFT LEFT
TERNARY IF, GREATER THAN
INTEGER, REAL, BOOLEAN LESS OR EQUALS THAN
NEGATION, BITWISE NOT ELSE, TERNARY ELSE, POWER
READ REAL INEQUALITY, BITWISE OR
LEFT PARENTHESIS MINUS RIGHT PARENTHESIS
 UnaryExpression> IDENTIFIER, BOOLEAN CAST BITWISE XOR, REMAINDER
PRINTLN, READ INTEGER COMMA
PLUS, MINUS, INTEGER CAST END OF INSTRUCTION
REAL CAST RETURN, LESS THAN
LAZY_AND, PLUS, LAZY_OR
INVERSE_DIVIDE, END, TIMES
ELSE_IF, DIVIDE
GREATER_OR_EQUALS_THAN

$<\! {\tt UnaryBitwiseNotExpression}\!>$	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></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
<Unary A tomic E xpression $>$	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></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

	I	TERRAL PAR TE
<binaryexpression'></binaryexpression'>	LAZY_OR, EPSILON_VALUE	TERNARY_IF RIGHT_PARENTHESIS, END COMMA END_OF_INSTRUCTION ELSE_IF, ELSE TERNARY_ELSE, RETURN
<binarylazyorexpression></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'></binarylazyorexpression'>	LAZY_AND, EPSILON_VALUE	TERNARY_IF, LAZY_OR RIGHT_PARENTHESIS, END COMMA END_OF_INSTRUCTION ELSE_IF, ELSE TERNARY ELSE, RETURN
<binarylazyandexpression></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
$<\!$	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></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'></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></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'></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
$<\!$	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
$<\!$	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

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
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	END, END_OF_INSTRUCTION ELSE IF, ELSE, RETURN
END, ELSE_IF, ELSE	END, END_OF_INSTRUCTION ELSE IF, ELSE, RETURN
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
	REAL, PLUS, BOOLEAN NEGATION, MINUS BITWISE_NOT, READ_REAL INTEGER_CAST BOOLEAN_CAST LEFT_PARENTHESIS IDENTIFIER, PRINTLN REAL_CAST IF END, ELSE_IF, ELSE READ_INTEGER, READ_REAL INTEGER_CAST BOOLEAN_CAST, PRINTLN

<functioncalltail></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></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></parametertail>	COMMA, EPSILON_VALUE	RIGHT_PARENTHESIS
<functiondefinition></functiondefinition>	FUNCTION	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<functiondefinitionend></functiondefinitionend>	END, RETURN	END, END_OF_INSTRUCTION ELSE_IF, ELSE, RETURN
<argument></argument>	EPSILON_VALUE IDENTIFIER	RIGHT_PARENTHESIS
<argumenttail></argumenttail>	COMMA, EPSILON_VALUE	RIGHT_PARENTHESIS

5 Action Table

Les lignes de "l'action table" représentent les variables et les colonnes représentent les terminaux. Une cellule de cette table contient le numéro d'une règle de production correspondant au numéro repris dans la section grammaire.

RETURN			6	11								C	9			39				7	10	54	6.0	70	89		7.5	79		82							66		
FUNCTION		7	∞		Ţ	T			\Box	Ţ	I	Ţ		_	П			_		\Box			Ţ							Ţ	Ţ					86	T	Ţ	_
PRINTLN	_	9	-			_					6	78	4	36	Ш	-	3 43 43 43 43 43	46	47	49	52525252		55 55 55 55 55	63 63 63 63 63 63	3	69 69 69 69 69	79 79 79 79 79	2	8080808080	_		92		94	Ш		_	_	_
BOOLEAN_CAST		9			_	4		\perp	Ш	_	- 6	3	4	386	Ш		4 3	346	747	45	52		555	63		69	12	3	080	4	_	91	_	194	Ш	_	\perp	\perp	_
REAL_CAST INTEGER CAST	_	9	_		_	_		_	Ш	_	Č	87.87.87.87.87.87.87.87.87.87.87.87.87.8	\perp	3636363	Н	-	4 5	346	747	345	25	1	55	99		396	1)8(4		88 89 90 91	1	94 94 94	Н	_	_	+	_
DEAD INTEGER	Г.		×		\dashv	+		\perp		\dashv	Č	$\frac{8}{2}$	+	33(Н		345	346	747	948	255		55	99	_	396	1	=)8(\dashv		8	1	192	Н		+	+	_
READ_INTEGER READ_REAL		9			_	_		-			è	<u>~</u>		36	Н	-	4 5	346	747	345	55		55	<u>ن</u>	5	396	1	-		_				192	Н		_	+	_
INTEGER TYPE	_	9	∞		_	_	_	\vdash	Н	_		<u>~</u>	+	36	Н	-	41	4	4.	34	52	-	22	<u></u>	5	39	72	4	80	4	_	87	1	94	Н	_	+	+	_
REAL TYPE	_		-		_	_	_	\vdash	Н	- 1	327	_	+		Н	_	_	-		_	_	\perp	-	+			_	+		\dashv	_	_	+		Н	_	+	+	_
BOOLEAN TYPE	_		⊢		\dashv	+	_	+	Н	- 1	526	+	+		Н		_	-			_	\vdash	_	+	+		_	+	\vdash	\dashv	_	_	+		Н	_	+	+	_
	_		ļ.,		-	+		-		-	25	_	+		Н		-	-				-		+	-		-	+	\blacksquare	\dashv			+		Н		+	+	_
LET	_	4	×		_	_		19		_		4	\perp		Н		-					1	-	N .	~		~1	_		21		+	-		Н		4	+	
ELSE			6		_	4		\perp	Ш			- 6	95		Ш	39	_			ы	10.	54	69	0	898		2 72	62		82).	ŏ	\perp		Ш		4	+	_
ELSE_IF			6		_	4		_		_		- 6	8		Ш	39				ы	10	54	ç	٥_	89		7.5	79		87	~	o T	+		Ш	_	_	+	_
IF	_	ည	×		4	4		\perp	20	_		4	+		Ш			_			_	\perp		+	_		_	\perp			83	_	\perp		Ш		\perp	\perp	_
POWER			_		4	4		\perp	Ш	_			4		Ш	39 39 39		_						4	_			-		81			\perp		Ш		\perp	\perp	_
INVERSE_DIVIDE															Ш	33												78		85								\perp	_
BITWISE_AND															Ш	33	\perp											77		82			_		Ш		_	4	_
REMAINDER																33												92		82								_	
DIVIDE															Ш	33								\perp				75		82			\perp		Ш				_
TIMES															Ш	33												7		82828			\perp		Ш			\perp	
ARITHMETIC_SHIFT_RIGHT			L						Ш						Ш	39 39 39 39 39										Ш	7.1	79	Ш	82					Ш				
ARITHMETIC_SHIFT_LEFT	L	L	L	\Box				\perp	Ш			\Box			Ш	39			Ш								70	79	Ш	82 82			\perp		Ш				
BITWISE_XOR	L	L	L	\Box	T		\perp	\perp	ĹĴ	T	[$oldsymbol{ol{ol{ol}}}}}}}}}}}}}}}} $	╝		\Box	39		\perp	ĹĴ	$\Box \Gamma$	\bot	\perp	$\Box \Gamma$	╝	29		72	79	\Box	$\overline{8}$	_[\perp	L	\Box	T	_[_
BITWISE_OR						T		Γ				T	J	Ι		39	T			I				J	99		72	79		82			Ι					T	_
INEQUALITY				П	\Box		T		П			T			П	39			\Box	\top	\top		-	01	89	П	72	79		82		\top	Τ		П		\top	\top	
EQUALITY					\neg					\neg		\top			\Box	39			\Box						89		7.5	79		828282			Т		\Box		\neg	\top	_
LESS OR EQUALS THAN				П	\neg	\top			П	7	T	\top	\top		П	39 39 39 39 39			П			П	EO 60 61	60	68 68 68 68 68 68 68 68 68 67	П	72 72 72 72 72	79 79 79 79 79 79 79 79 79 79 74 4 T	П	231	T				П		\neg	十	_
GREATER OR EQUALS THAN															П	33			П				0	ġ l	38		2	- 62		22					П		\top	\top	_
LESS THAN			\vdash		\dashv	\top		\top		\neg		\top	\top		Н	39 39 39 39			П				1	=	88		2	6		2			\top		Н		\top	\top	_
GREATER THAN					_	1		T							Н	66			Н				9	<u>ĕ</u> –	88		22	- 6		828282	_		+		Н	_	\top	+	_
LAZY AND			\vdash		_	+		+		_		+			Н	6						53	5	7	- 8		7.5	-6.		27	-		+		Н	_	+	+	-
LAZY OR	\vdash		\vdash		\dashv	+		+		\dashv		+	+		Н	393	-		\vdash	<u>C</u>	<u>-</u>	545	00 00	2	98		727272	-6.	\Box	878		+	+		Н		+	+	_
			\vdash		\dashv	-		+	\vdash	-		+	\vdash		Н	3																	+	,_				.7	4
RIGHT_PARENTHESIS												- 1	જ			39				E	10	54	G	ŏ	89		7.5	79		82				95			5	102	100
LEFT_PARENTHESIS					,	CI					0	22				38	1 40 41 41 41 41 3 42 43 43 43	46	48	49	52		22	63	3	69	7.5	9 79 79 79 79 79 79 79 79 79 79 79 79 79	80				93	94					
PLUS											6	χ _γ				39	43	45		49	52 52 52 52		55 55 55 55	63	64	69	2 2	2 6	80	82				94					
MINUS											8	\approx				39	43	44		49	52		22	63	65	69	72	2 62	80	82				94				П	
BITWISE NOT											0	22.28			П	-	12			49	52		55	65		69	7.5	2	80					94	П			T	
NEGATION											0	χ 7				9	-			49	52		55	65		69	7.5	2	80					94				T	
BOOLEAN											9	χχ.		35	П	-	41 41 41 43 43	46	47	49	52 52		22	633	3	69	2	2	80					94	П			T	
REAL											,	, , , ,		34	П	-	<u> </u>	16	17	61	22		55	33	3	39	2	2	200					94	П			\top	_
INTEGER			T					1			-	200		33	H	-	43,4	16	17	61	52		22	- 65	3	39	2	2	200	T			1	94	H		1	\top	_
TERNARY IF			\vdash		\dashv	\top		\top		\neg		``	67		Н	39	7 7	7			70	54	2.5	7	89		72	79	1	82			\top	<u> </u>	Н		\top	\top	_
COMMA			\vdash			1 2	_	+		\dashv			300		Н	39		\vdash	\Box		70	54	00	4	89		727	79		82			+		96		\dashv	- 9	103
			_			_ -	_	\perp		_		٥	٦		Ш	33						1 2	9	9	9		-	-1		∞		_	\perp		6		\perp	-	\equiv
TERNARY_ELSE	_		_		_			_		24		- 6	31		Ш	39				7	7 C	54	C	٥_	89		72	79		82			_		Ш		_	$^{\perp}$	_
FOR	П	2	∞						22						Ш															_			_		Ш		_	4	_
WHILE	-	2	∞						21						Ш																				Ш			\perp	
INTEGER_TYPE										- 1	22																											_	
REAL_TYPE										- 1	25 26 27													\perp									\perp						
BOOLEAN_TYPE	L		L	\square				L	Ш		25		\perp		Ш			\perp	\Box			$oxed{\Box}$		\perp		Ш			Ш	\Box			\perp		Ш				_
END_OF_INSTRUCTION	П	Ĺ	6	10	\Box			\perp	Ш	23	\Box		30		Ш	39		\perp	ШĪ	7	T C	54	G O	70	89	Ш	72	79	\Box	82	\Box	\perp	\perp		Ш	\Box			_
ASSIGNATION					(13	01																		I														
END			6	11	\Box	\top	T	П	П	\Box	\Box	Ş	3	T	П	39	\top		П	7.	10	54	G	7	89	П	72	79		82	ç	2	Т		П	6	100	T	
CONST		3	∞	-	+	+	20	+	\vdash	\dashv	\dashv	+	+	+	\vdash		+	-	\vdash	-	+	1	+	+	+	\vdash	-	+-	\vdash	~	-	+	+	+	\vdash	-	+	+	-
TYPE DEFINITION	Ë		1~	\vdash		41	-	+	\vdash	+	\dashv	+	+	-	$\vdash \vdash$	+	+		\vdash	+	+	+	+	+	+	\vdash	+	+	\vdash	+	+	+	+		$\vdash \vdash$	+	+	+	_
-			-	\vdash	-	-	+	+	\vdash	\dashv	\dashv	+	+			+			\square	+	-	\vdash	+	+	+		+	+		+	+	+	+		\vdash	\dashv	-	_	_
IDENTIFIER	-	2	∞		12					_	6	22		32	37	=	41	46	47	49	52		22	63	3	69	75	-	80					94				102 101	
EPSILON_VALUE	П		6	11								2	95			39				5	10	54	G	7	89		72	79		82				95	22		2	7.0	104
			H		\dashv	+	-	+	H	-		-1	'-	-	-		-		Н	- 1		L		+		Н		1-	Н	\sim		+	+	0,	0,		-	4	$\overline{}$
															$ \wedge $	딒		Λ						۸٨			- 1.												
					, ,	≙l							, 1	:	on	Ę	<u>1</u> .0	lon				l',	e l	# 'E	Ę	$ \Delta $	<u>^</u> <u>^</u>	16		싀							Δ		
			١,			<u>a</u>	۸ I ۸					- 13	글.[글	Ž	SSi	.i.j	^ 88	SSi	Sio	△ ./	^ ·s	sio	SSi		Sic	ior	0 S		iol	jö.			ĺΛ		١, ١		ğΙ		
		Λ	st?	Tai	Ĭij,	ਰ -						۸ ŀ	SSIC	ior	pre	ess		pr	[es	ior		es	ore	ž š	res	ess	SSS	Š S	.es	ess		$ \mathcal{C} $	jĘ,	\wedge		ţį.	넴	, l:	4
	^	<instruction></instruction>	ij	ist	5	5 F	ΞĮΞ			△	$^{\wedge}$	<expression></expression>	ğ <u>ă</u>	ess	뙶	pr	SS X	EX.	[찬	ess	ZSS X	x V	찙	<u> </u>	ţ k	Di/	pre	취류	Ϋ́	P		√ e	II	er	Ta	E	원(<argument></argument>	Тa
	<pre><pre>rogram</pre></pre>	cti	ion	nF.	st	밁.	loi lui	<block></block>	ď	<fortail></fortail>	<type></type>	SS	쳁	pr	erl	ξ		ns	[왕]	pr	H H	ıΕ	퀴달	븴	E	떮	퓠		년	췿	۸ď	< IIEnd >	Ca	<parameter< td=""><td>ter</td><td>)ef</td><td></td><td>ne.</td><td>int</td></parameter<>	ter)ef		ne.	int
	0.0	īTŪ	lct.	tioi.	11	nst	Dellar	lë.	<tood></tood>	딦	[3	ig E	S E	EX	[i]	Jer L		Ϋ́	اقزا	씳		ŐŐ	Ā	A H	eri		<u>, [</u>]	red F	tol	ţç	# [l e	an	ne.	넯	je	E.	me
	$_{ m L}$	nst	tr	uct	ifie	닯.	181 181	14	abla	띳	ν̈́,	<u>\$</u>		nic	en	ıtii	Wis	nus	[to]	ury.	az Z	azy	rzy		H		Fer F	Pil H	ac	ac	,	텔	Ę;	ar	rai	ţį.	덽	¥]	gn
	\ V	$\overline{\lor}$	<instructionlist< p=""></instructionlist<>	str	jit:	اإ	< Assignation Lan > (ConstDefinition)			×		∀	E E	ton		der	3it	ΜË	λ	ing	친	yL,	Ϋ́	2 Z	ĮŽ	LT.	Ly S	$\frac{1}{2} \hat{S} $		됬		Œ	Ę	\triangle	<parametertail></parametertail>	inc	ξĘ	V -	<argumenttail< p=""></argumenttail<>
			∇	<instructionlisttail< p=""></instructionlisttail<>	<identifierinstruction< td=""><td><identification p="" tail<=""></identification></td><td>7 7</td><td></td><td></td><td></td><td></td><td></td><td>< TernaryIIExpression > < TernaryFlseExpression ></td><td><atomicexpression< td=""><td><atomicidentifierexpression></atomicidentifierexpression></td><td>AtomicIdentifier Expression Tail></td><td><unarybitwisenotexpression></unarybitwisenotexpression></td><td>ry.</td><td><unaryatomicexpression></unaryatomicexpression></td><td><binaryexpression< td=""><td>< BinaryExpression > < BinaryLazyOrExpression></td><td><binarylazyorexpression'></binarylazyorexpression'></td><td>ary</td><td></td><td>ar</td><td><binarytermexpression></binarytermexpression></td><td><pre><binarytermexpression'> <binarytermexpression'></binarytermexpression'></binarytermexpression'></pre></td><td>Sinary Shifted Expression.</td><td><binaryfactorexpression></binaryfactorexpression></td><td><binaryfactorexpression?< td=""><td></td><td><pre><iiend> <builtinfunctioncall></builtinfunctioncall></iiend></pre></td><td><functioncalltail< td=""><td></td><td>\vee</td><td><functiondefinition></functiondefinition></td><td><functiondefinitionend></functiondefinitionend></td><td>- [</td><td>V</td></functioncalltail<></td></binaryfactorexpression?<></td></binaryexpression<></td></atomicexpression<></td></identifierinstruction<>	<identification p="" tail<=""></identification>	7 7						< TernaryIIExpression > < TernaryFlseExpression >	<atomicexpression< td=""><td><atomicidentifierexpression></atomicidentifierexpression></td><td>AtomicIdentifier Expression Tail></td><td><unarybitwisenotexpression></unarybitwisenotexpression></td><td>ry.</td><td><unaryatomicexpression></unaryatomicexpression></td><td><binaryexpression< td=""><td>< BinaryExpression > < BinaryLazyOrExpression></td><td><binarylazyorexpression'></binarylazyorexpression'></td><td>ary</td><td></td><td>ar</td><td><binarytermexpression></binarytermexpression></td><td><pre><binarytermexpression'> <binarytermexpression'></binarytermexpression'></binarytermexpression'></pre></td><td>Sinary Shifted Expression.</td><td><binaryfactorexpression></binaryfactorexpression></td><td><binaryfactorexpression?< td=""><td></td><td><pre><iiend> <builtinfunctioncall></builtinfunctioncall></iiend></pre></td><td><functioncalltail< td=""><td></td><td>\vee</td><td><functiondefinition></functiondefinition></td><td><functiondefinitionend></functiondefinitionend></td><td>- [</td><td>V</td></functioncalltail<></td></binaryfactorexpression?<></td></binaryexpression<></td></atomicexpression<>	<atomicidentifierexpression></atomicidentifierexpression>	AtomicIdentifier Expression Tail>	<unarybitwisenotexpression></unarybitwisenotexpression>	ry.	<unaryatomicexpression></unaryatomicexpression>	<binaryexpression< td=""><td>< BinaryExpression > < BinaryLazyOrExpression></td><td><binarylazyorexpression'></binarylazyorexpression'></td><td>ary</td><td></td><td>ar</td><td><binarytermexpression></binarytermexpression></td><td><pre><binarytermexpression'> <binarytermexpression'></binarytermexpression'></binarytermexpression'></pre></td><td>Sinary Shifted Expression.</td><td><binaryfactorexpression></binaryfactorexpression></td><td><binaryfactorexpression?< td=""><td></td><td><pre><iiend> <builtinfunctioncall></builtinfunctioncall></iiend></pre></td><td><functioncalltail< td=""><td></td><td>\vee</td><td><functiondefinition></functiondefinition></td><td><functiondefinitionend></functiondefinitionend></td><td>- [</td><td>V</td></functioncalltail<></td></binaryfactorexpression?<></td></binaryexpression<>	< BinaryExpression > < BinaryLazyOrExpression>	<binarylazyorexpression'></binarylazyorexpression'>	ary		ar	<binarytermexpression></binarytermexpression>	<pre><binarytermexpression'> <binarytermexpression'></binarytermexpression'></binarytermexpression'></pre>	Sinary Shifted Expression.	<binaryfactorexpression></binaryfactorexpression>	<binaryfactorexpression?< td=""><td></td><td><pre><iiend> <builtinfunctioncall></builtinfunctioncall></iiend></pre></td><td><functioncalltail< td=""><td></td><td>\vee</td><td><functiondefinition></functiondefinition></td><td><functiondefinitionend></functiondefinitionend></td><td>- [</td><td>V</td></functioncalltail<></td></binaryfactorexpression?<>		<pre><iiend> <builtinfunctioncall></builtinfunctioncall></iiend></pre>	<functioncalltail< td=""><td></td><td>\vee</td><td><functiondefinition></functiondefinition></td><td><functiondefinitionend></functiondefinitionend></td><td>- [</td><td>V</td></functioncalltail<>		$ \vee $	<functiondefinition></functiondefinition>	<functiondefinitionend></functiondefinitionend>	- [V
	l			V	\vee	ğ						1	∨ IĘ		15	B	, lan	na	151	v \	× ابت	3ir	.E	ğ 1.5	1.1	m	찐	겙	Bil	<u> </u>		$-1\overline{\vee}$	T.	1		V [뙨		
																					- 144		~ 1.,	Z 10~	1 ~~				1 1		- 1				1 1				
						\vee							\		$ V_A $	Ato	15	<unaryminusplusexpression< td=""><td>∇</td><td></td><td> 7</td><td></td><td><binarylazyandexpression></binarylazyandexpression></td><td>< BinaryLazyAndExpression ></td><td>< Binary Numeric Expression</td><td>\vee</td><td>¥ 7</td><td>7 7</td><td>∇</td><td>∇</td><td></td><td></td><td></td><td></td><td></td><td></td><td>٧ J</td><td></td><td></td></unaryminusplusexpression<>	$ \nabla $		7		<binarylazyandexpression></binarylazyandexpression>	< BinaryLazyAndExpression >	< Binary Numeric Expression	$ \vee $	¥ 7	7 7	∇	∇							٧ J		