Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Кафедра вычислительной техники

Отчет по лабораторной работе № 4

по дисциплине «Теория формальных языков и компиляторов»

на тему «Синтаксис языков программирования.  
Нисходящий синтаксический анализ»

Студент: Резниченко М. К.

Группа: АВТ-709

Преподаватель: Малявко А.А.

Вариант: 33143423

Новосибирск, 2020

# Цель работы

Изучение основных идей и понятий нисходящих методов синтаксического анализа. Выявление свойств формальных грамматик, необходимых для реализации нисходящего восстановления дерева грамматического разбора. Приобретение навыков построения процедурной и различных автоматных реализаций нисходящего анализа. Исследование поведения нисходящих синтаксических акцепторов.

# Постановка задачи

1. Используя пакет ВебТрансЛаб:

- расширить грамматику заданного на курсовую работу языка до полной;

- изучить и освоить проверку принадлежности грамматики к классу LL1, используя в качестве проверяемых грамматики, полученные при выполнении работы №4;

- освоить технологию удаления символов из множеств выбора правил с использованием тега <exclude> для приведения грамматики к классу LL1 и примеров 6IfElseWithConflict и 7IfElseNoConflict;

- построить конечный автомат со стековой памятью и несколькими состояниями (шаблон …SyntAsMultiFSM…), разобраться в структуре управляющей таблицы автомата, уяснить способы формирования и использования всех полей;

- построить конечный автомат со стековой памятью и одним состоянием, управляемый входным символом и символом, снятым с верхушки стека (шаблон …SyntAsSingleFSM…), разобраться в структуре управляющей таблицы автомата, уяснить способы формирования и использования клеток таблицы;

- построить процедурную реализацию рекурсивного спуска (шаблон …SyntAsRD…), уяснить способы формирования функций этого акцептора.

2. Выполнить трассировку процессов нисходящего синтаксического акцепта, изучить поведение всех построенных синтаксических акцепторов при разборе как правильных предложений, так и предложений с намеренно внесенными синтаксическими ошибками.

3. Проанализировать и сравнить между собой все полученные тексты программ и результаты выполнения пункта 3.2. Оценить степень пригодности изученных вариантов реализации нисходящих синтаксических акцепторов для выполнения курсовой работы.

**Лексика языка программирования**, заданного вариантом 33143423:

1. Идентификаторы:  
   <б><пЦ><пБ> (a13KL, w8U, …)
2. Константы:  
   целые по основаниям 4, 8 и 10, вещественные, строковые.
3. Объявления примитивных типов:  
   – целое – long, long\_u  
   – вещественное – number  
   – строковые – char
4. Оператор присваивания:  
   put <В> to <И>;
5. Условный оператор:  
   when <ЛВ> then <ОБ> [other <ОБ>];
6. Оператор цикла:  
   exec <ОБ> with <И> from <К> to <К> [step <К>];
7. Оператор переключателя:  
   case <В> { when <К> then <ОБ> [exit;] … [other <ОБ> ]}

# Доработка грамматики заданного языка

В ходе данной работе была доработана грамматика языка, заданного вариантом. В дополнение к правилам, реализованным в 3 л/р, были добавлено следующее:

– Полноценная программа должна состоять из опционального объявления пакета, любого количества объявлений импортов (0 и более) и как минимум одного объявления функции или переменной верхнего уровня. Описание объявления пакета и импортов было приведено во 2 л/р.

– Функции должны состоять из объявления типа возвращаемого значения (в случае отсутствия такового – тип *void*), названия функции, перечисления любого количества параметров (0 и более) в круглых скобках и блока кода в фигурных скобках. Блок кода может состоять из любого количества (0 и более) объявлений функций и переменных и вызовов операторов, описанных в 3 л/р.

Правила синтаксиса разработанного языка представлены в приложении A.

# Тестовая программа

package my.program;

import system.output.a0print;

long\_u f4ctorial(long\_u x0x) {

when x0x == 0

then

return 0;

other {

long\_u a0f;

put 1 to a0f;

exec put a0f \* i0i to a0f;

with i0i from 1 tox0x;

return a0f;

};

}

void s4yNumber(long a0a) {

char n1n;

case a0a {

when 0 then {

put "zero" to n1n;

exit;  
 }  
 when 1 then {

put "one " to n1n;

exit;  
 }

other put "unknown" to n1n;

}

}

long\_u a0a;

void m4in() {

put 0q10 to a0a;

a0print(f4ctorial(a0a));

s4yNumber(0c1);

}

Правила синтаксиса были протестированы на данной программе. Тестирования показало, что данные правила составлены корректно.

# LL(1)-грамматика

LL(1)-грамматикой называется такая контекстно-свободная грамматика, у которой множества выбора правил с одинаковой нетерминалом в левой части попарно не пересекаются. Любая такая грамматика может быть использована для организации нисходящего детерминированного восстановления дерева грамматического разбора предложений порождаемого ею языка.

Принято считать, что символы в названии класса LL(1)-грамматик обозначают следующее. Первая буква L (left - левый) - чтение слов анализируемого предложения производится слева направо. Вторая буква L (leftmost - самый левый) - на каждом шаге принимается решение для замены самого левого нетерминала из текущего уровня восстанавливаемого дерева. Цифра 1 в скобках обозначает количество символов из начала остатка предложения, необходимых для принятия решения о выборе правила на каждом шаге детерминированного нисходящего восстановления дерева грамматического разбора.

Рассмотрим множества выбора правил синтаксиса разработанного языка (приложение Б). Как можно видеть из таблицы, множества выбора правил с одинаковым нетерминалом в левой части попарно не пересекаются. Поэтому разработанная контекстно-свободная грамматика является LL(1)-грамматикой согласно определению, данному выше.

# Конечный автомат с несколькими состояниями

Функционирование конечного автомата со стековой памятью и несколькими состояниями определяется управляющей таблицей. Предполагается, что автомат при запуске оказывается в особом начальном состоянии, на каждом такте по входному символу и текущему состоянию определяет и выполняет операции над входным потоком символов, стековой памятью и собственным состоянием.

Каждому символу каждого правила грамматики должно быть поставлено в соответствие в точности одно состояние автомата. С каждым состоянием должно быть связано множество выбора и два адреса перехода (для положительного и отрицательного результата проверки). Под адресом перехода понимается номер состояния.

С каждый состоянием должны быть также связаны операции управления стековой памятью и операции управления чтением следующего входного символа. Все операции управления могут задаваться булевыми значениями (далее – флажками):

- флажок a управляет чтением следующего входного символа;

- флажок s управляет занесением адреса точки возврата в стек;

- флажок r обеспечивает переключение автомата в состояние, номер которого снимается с верхушки стека возвратов;

- флажок e запрещает останов по ошибке в случае, когда состояние соответствует нетерминалу из левой части и есть еще хотя бы одно правило для такого нетерминала.

# Конечный автомат с одним состоянием

LL(1)-грамматику легко можно преобразовать в конечный автомат с единственным состоянием и стековой памятью, управляемый:

- текущим входным символом;

- символом, находящимся на верхушке стека.

Поведение такого автомата определяется управляющей таблицей, столбцы которой соответствуют входным символам, строки – символам, которые могут находиться в стеке, а в клетках указана некоторая последовательность операций над стеком, входным потоком и состоянием автомата.

# Вывод

В ходе выполнения лабораторной работы была доработана грамматика языка, заданного вариантом. Также были изучены основные идеи и понятия нисходящих методов синтаксического анализа и рассмотрены принципы работы конечного автомата с несколькими состояниями и конченого автомата с одним состоянием для реализации рекурсивного спуска.

# Приложение А. Правила синтаксиса

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Левая часть** | **Правая часть** | | | | | | | | | | | | | **Примечание** |
| **S** | [ | **Package** | ]? | [ | **Import** | ]\* | [ | **FunVarTop** | ]+ | | | | | Код программы |
| **Package** | *package* | packageId | *;* | | | | | | | | | | | Объявление пакета |
| **Import** | *import* | importId | *;* | | | | | | | | | | | Объявление импорта |
| **FunVarTop** | **VarType** | id | **FunVarTopEnd** | | | | | | | | | | | Объявление ф-ии/перем. верхнего уровня |
| **FunVarTop** | *void* | id | **FunTopEnd** | | | | | | | | | | |  |
| **FunVarTopEnd** | **FunTopEnd** | | | **VarTopEnd** | | | | | | | | | | |  |
| **VarTopEnd** | *;* | | | | | | | | | | | | |  |
| **FunTopEnd** | *(* | [ | **FunArgs** | ]? | *)* | *{* | [ | **Statement** | ]\* | *}* | | | |  |
| **FunArgs** | **VarType** | id | [ | *,* | **VarType** | id | ]\* | | | | | | | Аргументы функции |
| **Statement** | **Variable** | | | **Assignment** | | | **Condition** | | | **Cycle** | | | **Switch** | | | **Return** | | | **FunctionCall** | Строка кода |
| **Variable** | **VarType** | id | *;* | | | | | | | | | | | Объявление переменной |
| **VarType** | *long* | | | *long\_u* | | | *number* | | | *char* | | | *bool* | | | | | Тип переменной |
| **Return** | *return* | [ | **Expression** | ]? | *;* | | | | | | | | | Возврат значения |
| **Assignment** | *put* | **Expression** | *to* | id | *;* | | | | | | | | | Присваивание |
| **Condition** | *when* | **LogicExpr** | *then* | **LineBlock** | [ | *other* | **LineBlock** | ]? | *;* | | | | | Условный оператор |
| **Cycle** | *exec* | **LineBlock** | *with* | id | *from* | **IntConstVar** | *to* | **IntConstVar** | [ | *step* | **IntConstVar** | ]? | *;* | Оператор цикла |
| **Switch** | *case* | **Expression** | *{* | [ | **SwitchWhen** | ]+ | [ | *other* | **LineBlock** | ]? | *}* | | | Оператор переключателя |
| **SwitchWhen** | *when* | **Const** | *then* | **WhenLineBlock** | | | | | | | | | |  |
| **LineBlock** | **Statement** | | | | | | | | | | | | |  |
| **LineBlock** | *{* | [ | **Statement** | ]\* | *}* | | | | | | | | |  |
| **WhenLineBlock** | **Statement** | | | | | | | | | | | | |  |
| **WhenLineBlock** | *{* | [ | **Statement** | ]\* | [ | *exit* | *;* | ]? | *}* | | | | |  |
| **Expressions** | **Expression** | [ | *,* | **Expression** | ]\* | | | | | | | | | Набор выражений |
| **Expression** | stringConst | | | | | | | | | | | | | Выражение |
| **Expression** | **AConst** | [ | **ArythmExprACC** | ]? | | | | | | | | | |  |
| **Expression** | **LConst** | [ | **LogicExprACC** | ]? | | | | | | | | | |  |
| **Expression** | **FunVarCall** | [ | **ExprAfterFVC** | ]? | | | | | | | | | |  |
| **Expression** | **ArythmExprWCC** | | | **LogicExprWCC** | | | | | | | | | | |  |
| **ExprAfterFVC** | **ArythmExprACC** | | | **LogicExprACC** | | | | | | | | | | | Выражение после вызова ф-ии/перем. |
| **FunctionCall** | id | **FunCall** | *;* | | | | | | | | | | | Вызов функции как строка кода |
| **FunVarCall** | id | [ | **FunCall** | ]? | | | | | | | | | | Вызов функции или переменной |
| **FunCall** | *(* | [ | **Expressions** | ]? | *)* | | | | | | | | | Вызов функции |
| **ArythmExpr** | **AX** | [ | **AY** | ]? | | | | | | | | | | Арифметическое выражение |
| **ArythmExprACC** | **AZ** | [ | **AY** | ]? | | | | | | | | | | Арифм. выраж. после ф-ии/перем. /конст. |
| **ArythmExprACC** | **AY** | | | | | | | | | | | | |  |
| **ArythmExprWCC** | *(* | **ArythmExpr** | *)* | [ | **AZ** | ]? | [ | **AY** | ]? | | | | | Арифм. выраж. без ф-ии/перем./конст. |
| **AY** | *+* | **ArythmExpr** | | | | | | | | | | | |  |
| **AY** | *-* | **ArythmExpr** | | | | | | | | | | | |  |
| **AX** | **AR** | [ | **AZ** | ]? | | | | | | | | | |  |
| **AR** | *(* | **ArythmExpr** | *)* | | | | | | | | | | |  |
| **AR** | **FunVarCall** | | | | | | | | | | | | |  |
| **AR** | **AConst** | | | | | | | | | | | | |  |
| **AZ** | *\** | **AX** | | | | | | | | | | | |  |
| **AZ** | */* | **AX** | | | | | | | | | | | |  |
| **AConst** | **IntConst** | | | floatConst | | | | | | | | | | | Числовая константа |
| **IntConst** | quaterneryConst | | | octalConst | | | decimalConst | | | | | | | | | Целочисленная константа |
| **IntConstVar** | id | | | **IntConst** | | | | | | | | | | | Целочисленная переменная или константа |
| **LogicExpr** | **LX** | [ | **LY** | ]? | | | | | | | | | | Логическое выражение |
| **LogicExprACC** | [ | **CompPart** | ]? | **LogicExprACC1** | | | | | | | | | | Лог. выраж. После ф-ии/перем. /конст. |
| **LogicExprACC1** | **LZ** | [ | **LY** | ]? | | | | | | | | | |  |
| **LogicExprACC1** | **LY** | | | | | | | | | | | | |  |
| **LogicExprWCC** | *!* | **LRnot** | [ | **LZ** | ]? | [ | **LY** | ]? | | | | | | Лог. выраж. без ф-ии/перем./конст. |
| **LY** | *||* | **LogicExpr** | | | | | | | | | | | |  |
| **LX** | **LR** | [ | **LZ** | ]? | | | | | | | | | |  |
| **LZ** | *&&* | **LR** | [ | **LZ** | ]? | | | | | | | | |  |
| **LR** | *!* | **LRnot** | | | | | | | | | | | |  |
| **LR** | **FunVarCall** | [ | **CompPart** | ]? | | | | | | | | | |  |
| **LR** | **LConst** | [ | **CompPart** | ]? | | | | | | | | | |  |
| **LRnot** | **FunVarCall** | | | **LConst** | | | | | | | | | | |  |
| **LConst** | *true* | | | *false* | | | | | | | | | | | Логическая константа |
| **CompPart** | **CompSign** | **CompValue** | | | | | | | | | | | | Сравнение |
| **CompSign** | *==* | | | *!=* | | | *>* | | | *>=* | | | *<* | | | *<=* | | | Знак сравнения |
| **CompValue** | **FunVarCall** | | | **Const** | | | | | | | | | | | Логическое значение |
| **Const** | **AConst** | | | **LConst** | | | stringConst | | | | | | | | | Константа |

# Приложение Б. Множества выбора правил

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Правило № | Для нетерминала | Множество выбора | | | | | | | | | | | | | | | | | |
| 0 | **S** | *package* | *import* | *bool* | *char* | *long* | *long\_u* | *number* | *void* | | | | | | | | | | |
| 1 | **Package** | *package* | | | | | | | | | | | | | | | | | |
| 2 | **Import** | *import* | | | | | | | | | | | | | | | | | |
| 3 | **FunVarTop** | *bool* | *char* | *long* | *long\_u* | *number* | | | | | | | | | | | | | |
| 4 | **FunVarTop** | *void* | | | | | | | | | | | | | | | | | |
| 5 | **FunVarTopEnd** | *(* | | | | | | | | | | | | | | | | | |
| 6 | **FunVarTopEnd** | *;* | | | | | | | | | | | | | | | | | |
| 7 | **FunTopEnd** | *(* | | | | | | | | | | | | | | | | | |
| 8 | **FunArgs** | *bool* | *char* | *long* | *long\_u* | *number* | | | | | | | | | | | | | |
| 9 | **Statement** | *bool* | *char* | *long* | *long\_u* | *number* | | | | | | | | | | | | | |
| 10 | **Statement** | *put* | | | | | | | | | | | | | | | | | |
| 11 | **Statement** | *when* | | | | | | | | | | | | | | | | | |
| 12 | **Statement** | *exec* | | | | | | | | | | | | | | | | | |
| 13 | **Statement** | *case* | | | | | | | | | | | | | | | | | |
| 14 | **Statement** | *return* | | | | | | | | | | | | | | | | | |
| 15 | **Statement** | id | | | | | | | | | | | | | | | | | |
| 16 | **VarType** | *long* | | | | | | | | | | | | | | | | | |
| 17 | **VarType** | *long\_u* | | | | | | | | | | | | | | | | | |
| 18 | **VarType** | *number* | | | | | | | | | | | | | | | | | |
| 19 | **VarType** | *char* | | | | | | | | | | | | | | | | | |
| 20 | **VarType** | *bool* | | | | | | | | | | | | | | | | | |
| 21 | **SwitchWhen** | *when* | | | | | | | | | | | | | | | | | |
| 22 | **LineBlock** | id | *bool* | *case* | *char* | *exec* | *long* | *long\_u* | *number* | *put* | *return* | *when* | | | | | | | |
| 23 | **LineBlock** | *{* | | | | | | | | | | | | | | | | | |
| 24 | **WhenLineBlock** | id | *bool* | *case* | *char* | *exec* | *long* | *long\_u* | *number* | *put* | *return* | *when* | | | | | | | |
| 25 | **WhenLineBlock** | *{* | | | | | | | | | | | | | | | | | |
| 26 | **Expressions** | id | stringConst | floatConst | quaterneryConst | octalConst | decimalConst | *!* | *(* | *false* | *true* | | | | | | | | |
| 27 | **Expression** | stringConst | | | | | | | | | | | | | | | | | |
| 28 | **Expression** | floatConst | quaterneryConst | octalConst | decimalConst | | | | | | | | | | | | | | |
| 29 | **Expression** | *false* | *true* | | | | | | | | | | | | | | | | |
| 30 | **Expression** | id | | | | | | | | | | | | | | | | | |
| 31 | **Expression** | *(* | | | | | | | | | | | | | | | | | |
| 32 | **Expression** | *!* | | | | | | | | | | | | | | | | | |
| 33 | **ExprAfterFVC** | *\** | */* | | | | | | | | | | | | | | | | |
| 34 | **ExprAfterFVC** | *+* | *-* | | | | | | | | | | | | | | | | |
| 35 | **ExprAfterFVC** | *!=* | *<* | *<=* | *==* | *>* | *>=* | *&&* | *||* | | | | | | | | | | |
| 36 | **FunVarCall** | id | | | | | | | | | | | | | | | | | |
| 37 | **FunCall** | *(* | | | | | | | | | | | | | | | | | |
| 38 | **ArythmExpr** | id | floatConst | quaterneryConst | octalConst | decimalConst | *(* | | | | | | | | | | | | |
| 39 | **ArythmExprACC** | *\** | */* | | | | | | | | | | | | | | | | |
| 40 | **ArythmExprACC** | *+* | *-* | | | | | | | | | | | | | | | | |
| 41 | **ArythmExprWCC** | *(* | | | | | | | | | | | | | | | | | |
| 42 | **AY** | *+* | | | | | | | | | | | | | | | | | |
| 43 | **AY** | *-* | | | | | | | | | | | | | | | | | |
| 44 | **AX** | id | floatConst | quaterneryConst | octalConst | decimalConst | *(* | | | | | | | | | | | | |
| 45 | **AR** | *(* | | | | | | | | | | | | | | | | | |
| 46 | **AR** | id | | | | | | | | | | | | | | | | | |
| 47 | **AR** | floatConst | quaterneryConst | octalConst | decimalConst | | | | | | | | | | | | | | |
| 48 | **AZ** | *\** | | | | | | | | | | | | | | | | | |
| 49 | **AZ** | */* | | | | | | | | | | | | | | | | | |
| 50 | **AConst** | quaterneryConst | octalConst | decimalConst | | | | | | | | | | | | | | | |
| 51 | **AConst** | floatConst | | | | | | | | | | | | | | | | | |
| 52 | **IntConst** | quaterneryConst | | | | | | | | | | | | | | | | | |
| 53 | **IntConst** | octalConst | | | | | | | | | | | | | | | | | |
| 54 | **IntConst** | decimalConst | | | | | | | | | | | | | | | | | |
| 55 | **IntConstVar** | id | | | | | | | | | | | | | | | | | |
| 56 | **IntConstVar** | quaterneryConst | octalConst | decimalConst | | | | | | | | | | | | | | | |
| 57 | **LogicExpr** | id | *!* | *false* | *true* | | | | | | | | | | | | | | |
| 58 | **LogicExprACC** | *!=* | *<* | *<=* | *==* | *>* | *>=* | *&&* | *||* | | | | | | | | | | |
| 59 | **LogicExprACC1** | *&&* | | | | | | | | | | | | | | | | | |
| 60 | **LogicExprACC1** | *||* | | | | | | | | | | | | | | | | | |
| 61 | **LogicExprWCC** | *!* | | | | | | | | | | | | | | | | | |
| 62 | **LY** | *||* | | | | | | | | | | | | | | | | | |
| 63 | **LX** | id | *!* | *false* | *true* | | | | | | | | | | | | | | |
| 64 | **LZ** | *&&* | | | | | | | | | | | | | | | | | |
| 65 | **LR** | *!* | | | | | | | | | | | | | | | | | |
| 66 | **LR** | id | | | | | | | | | | | | | | | | | |
| 67 | **LR** | *false* | *true* | | | | | | | | | | | | | | | | |
| 68 | **LRnot** | id | | | | | | | | | | | | | | | | | |
| 69 | **LRnot** | *true* | | | | | | | | | | | | | | | | | |
| 70 | **LRnot** | *false* | | | | | | | | | | | | | | | | | |
| 71 | **LConst** | *true* | | | | | | | | | | | | | | | | | |
| 72 | **LConst** | *false* | | | | | | | | | | | | | | | | | |
| 73 | **CompPart** | *!=* | *<* | *<=* | *==* | *>* | *>=* | | | | | | | | | | | | |
| 74 | **CompSign** | *==* | | | | | | | | | | | | | | | | | |
| 75 | **CompSign** | *!=* | | | | | | | | | | | | | | | | | |
| 76 | **CompSign** | *>* | | | | | | | | | | | | | | | | | |
| 77 | **CompSign** | *>=* | | | | | | | | | | | | | | | | | |
| 78 | **CompSign** | *<* | | | | | | | | | | | | | | | | | |
| 79 | **CompSign** | *<=* | | | | | | | | | | | | | | | | | |
| 80 | **CompValue** | id | | | | | | | | | | | | | | | | | |
| 81 | **CompValue** | floatConst | quaterneryConst | octalConst | decimalConst | | | | | | | | | | | | | | |
| 82 | **CompValue** | *false* | *true* | | | | | | | | | | | | | | | | |
| 83 | **CompValue** | stringConst | | | | | | | | | | | | | | | | | |
| 84 | **Const** | floatConst | quaterneryConst | octalConst | decimalConst | | | | | | | | | | | | | | |
| 85 | **Const** | *false* | *true* | | | | | | | | | | | | | | | | |
| 86 | **Const** | stringConst | | | | | | | | | | | | | | | | | |
| 87 | **S\_0** | *package* | | | | | | | | | | | | | | | | | |
| 88 | **S\_0** | *import* | *bool* | *char* | *long* | *long\_u* | *number* | *void* | | | | | | | | | | | |
| 89 | **S\_1** | *import* | | | | | | | | | | | | | | | | | |
| 90 | **S\_1** | *bool* | *char* | *long* | *long\_u* | *number* | *void* | | | | | | | | | | | | |
| 91 | **S\_2** | *bool* | *char* | *long* | *long\_u* | *number* | *void* | | | | | | | | | | | | |
| 92 | **S\_3** | *bool* | *char* | *long* | *long\_u* | *number* | *void* | | | | | | | | | | | | |
| 93 | **S\_3** | EOF | | | | | | | | | | | | | | | | | |
| 94 | **FunTopEnd\_0** | *bool* | *char* | *long* | *long\_u* | *number* | | | | | | | | | | | | | |
| 95 | **FunTopEnd\_0** | *)* | | | | | | | | | | | | | | | | | |
| 96 | **FunTopEnd\_1** | id | *bool* | *case* | *char* | *exec* | *long* | *long\_u* | *number* | *put* | *return* | *when* | | | | | | | |
| 97 | **FunTopEnd\_1** | *}* | | | | | | | | | | | | | | | | | |
| 98 | **FunArgs\_0** | *,* | | | | | | | | | | | | | | | | | |
| 99 | **FunArgs\_0** | *)* | | | | | | | | | | | | | | | | | |
| 100 | **Return\_0** | id | stringConst | floatConst | quaterneryConst | octalConst | decimalConst | *!* | *(* | *false* | *true* | | | | | | | | |
| 101 | **Return\_0** | *;* | | | | | | | | | | | | | | | | | |
| 102 | **Condition\_0** | *other* | | | | | | | | | | | | | | | | | |
| 103 | **Condition\_0** | *;* | | | | | | | | | | | | | | | | | |
| 104 | **Cycle\_0** | *step* | | | | | | | | | | | | | | | | | |
| 105 | **Cycle\_0** | *;* | | | | | | | | | | | | | | | | | |
| 106 | **Switch\_0** | *when* | | | | | | | | | | | | | | | | | |
| 107 | **Switch\_1** | *when* | | | | | | | | | | | | | | | | | |
| 108 | **Switch\_1** | *other* | *}* | | | | | | | | | | | | | | | | |
| 109 | **Switch\_2** | *other* | | | | | | | | | | | | | | | | | |
| 110 | **Switch\_2** | *}* | | | | | | | | | | | | | | | | | |
| 111 | **LineBlock\_0** | id | *bool* | *case* | *char* | *exec* | *long* | *long\_u* | *number* | *put* | *return* | *when* | | | | | | | |
| 112 | **LineBlock\_0** | *}* | | | | | | | | | | | | | | | | | |
| 113 | **WhenLineBlock\_0** | id | *bool* | *case* | *char* | *exec* | *long* | *long\_u* | *number* | *put* | *return* | *when* | | | | | | | |
| 114 | **WhenLineBlock\_0** | *exit* | *}* | | | | | | | | | | | | | | | | |
| 115 | **WhenLineBlock\_1** | *exit* | | | | | | | | | | | | | | | | | |
| 116 | **WhenLineBlock\_1** | *}* | | | | | | | | | | | | | | | | | |
| 117 | **Expressions\_0** | *,* | | | | | | | | | | | | | | | | | |
| 118 | **Expressions\_0** | *)* | | | | | | | | | | | | | | | | | |
| 119 | **Expression\_0** | *\** | *+* | *-* | */* | | | | | | | | | | | | | | |
| 120 | **Expression\_0** | *,* | *)* | *;* | *to* | *{* | | | | | | | | | | | | | |
| 121 | **Expression\_1** | *!=* | *&&* | *<* | *<=* | *==* | *>* | *>=* | *||* | | | | | | | | | | |
| 122 | **Expression\_1** | *,* | *)* | *;* | *to* | *{* | | | | | | | | | | | | | |
| 123 | **Expression\_2** | *!=* | *&&* | *<* | *<=* | *\** | *+* | *-* | */* | *==* | *>* | *>=* | *||* | | | | | | |
| 124 | **Expression\_2** | *,* | *)* | *;* | *to* | *{* | | | | | | | | | | | | | |
| 125 | **FunVarCall\_0** | *(* | | | | | | | | | | | | | | | | | |
| 126 | **FunVarCall\_0** | *,* | *!=* | *&&* | *<* | *<=* | *\** | *+* | *-* | */* | *==* | *>* | *>=* | *||* | *)* | *;* | *then* | *to* | *{* |
| 127 | **FunCall\_0** | id | stringConst | floatConst | quaterneryConst | octalConst | decimalConst | *!* | *(* | *false* | *true* | | | | | | | | |
| 128 | **FunCall\_0** | *)* | | | | | | | | | | | | | | | | | |
| 129 | **ArythmExpr\_0** | *+* | *-* | | | | | | | | | | | | | | | | |
| 130 | **ArythmExpr\_0** | *,* | *)* | *;* | *to* | *{* | | | | | | | | | | | | | |
| 131 | **ArythmExprACC\_0** | *+* | *-* | | | | | | | | | | | | | | | | |
| 132 | **ArythmExprACC\_0** | *,* | *)* | *;* | *to* | *{* | | | | | | | | | | | | | |
| 133 | **ArythmExprWCC\_0** | *\** | */* | | | | | | | | | | | | | | | | |
| 134 | **ArythmExprWCC\_0** | *,* | *+* | *-* | *)* | *;* | *to* | *{* | | | | | | | | | | | |
| 135 | **ArythmExprWCC\_1** | *+* | *-* | | | | | | | | | | | | | | | | |
| 136 | **ArythmExprWCC\_1** | *,* | *)* | *;* | *to* | *{* | | | | | | | | | | | | | |
| 137 | **AX\_0** | *\** | */* | | | | | | | | | | | | | | | | |
| 138 | **AX\_0** | *,* | *+* | *-* | *)* | *;* | *to* | *{* | | | | | | | | | | | |
| 139 | **LogicExpr\_0** | *||* | | | | | | | | | | | | | | | | | |
| 140 | **LogicExpr\_0** | *,* | *)* | *;* | *then* | *to* | *{* | | | | | | | | | | | | |
| 141 | **LogicExprACC\_0** | *!=* | *<* | *<=* | *==* | *>* | *>=* | | | | | | | | | | | | |
| 142 | **LogicExprACC\_0** | *&&* | *||* | | | | | | | | | | | | | | | | |
| 143 | **LogicExprACC1\_0** | *||* | | | | | | | | | | | | | | | | | |
| 144 | **LogicExprACC1\_0** | *,* | *)* | *;* | *to* | *{* | | | | | | | | | | | | | |
| 145 | **LogicExprWCC\_0** | *&&* | | | | | | | | | | | | | | | | | |
| 146 | **LogicExprWCC\_0** | *,* | *||* | *)* | *;* | *to* | *{* | | | | | | | | | | | | |
| 147 | **LogicExprWCC\_1** | *||* | | | | | | | | | | | | | | | | | |
| 148 | **LogicExprWCC\_1** | *,* | *)* | *;* | *to* | *{* | | | | | | | | | | | | | |
| 149 | **LX\_0** | *&&* | | | | | | | | | | | | | | | | | |
| 150 | **LX\_0** | *,* | *||* | *)* | *;* | *then* | *to* | *{* | | | | | | | | | | | |
| 151 | **LZ\_0** | *&&* | | | | | | | | | | | | | | | | | |
| 152 | **LZ\_0** | *,* | *||* | *)* | *;* | *then* | *to* | *{* | | | | | | | | | | | |
| 153 | **LR\_0** | *!=* | *<* | *<=* | *==* | *>* | *>=* | | | | | | | | | | | | |
| 154 | **LR\_0** | *,* | *||* | *&&* | *)* | *;* | *then* | *to* | *{* | | | | | | | | | | |
| 155 | **LR\_1** | *!=* | *<* | *<=* | *==* | *>* | *>=* | | | | | | | | | | | | |
| 156 | **LR\_1** | *,* | *||* | *&&* | *)* | *;* | *then* | *to* | *{* | | | | | | | | | | |