Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Кафедра вычислительной техники

Отчет по лабораторной работе № 3

по дисциплине «Теория формальных языков и компиляторов»

на тему «Синтаксис языков программирования.  
Формальные грамматики»

Студент: Резниченко М. К.

Группа: АВТ-709

Преподаватель: Малявко А.А.

Вариант: 33143423

Новосибирск, 2020

# Цель работы

Изучение основных понятий метаязыка формальных грамматик, свойств грамматик и нетерминальных символов, рекурсивности и однозначности грамматик, недостижимости, бесплодности, аннулируемости и рекурсивности нетерминальных символов, отношений предшествования и последования между символами, приобретение навыков эквивалентных преобразований формальных грамматик, освоение технологий разработки формальных грамматик.

# Постановка задачи

1. Изучить интерфейс пакета ВебТрансЛаб, предназначенный для работы с синтаксическими правилами.

2. На примерах 3LL1 и 4SLR1GrammarArithmExpr (грамматики Ga1 и Ga2 из учебника) изучить и освоить:

- ввод и редактирование правил грамматики;

- построение синтаксических акцепторов с целью вычисления свойств грамматики и ее символов – пункт меню «Построить» с выбором нужного шаблона;

- понятия множеств предшественников и множеств последователей для нетерминальных символов;

- понятие множеств выбора правил грамматики и их пересечений для правил с одинаковыми нетерминалами в левой части;

- просмотр свойств символов грамматики (следует достичь понимания того, почему те или иные символы грамматики имеют свой конкретный набор свойств – пункт меню «Показать» и подпункты «Правила грамматики», «Отношение предшествования», «Отношение последования» и «Множества выбора правил».

3. Изучить способы эквивалентных преобразований грамматики из примера 4SLR1GrammarArithmExpr для устранения пересечений множеств выбора правил с одинаковыми нетерминалами в левой части, т.е. превращения этой грамматики в LL(1)-грамматику.

4. Ориентируясь на свой вариант задания на курсовую работу, модифицировать (расширить) грамматики Ga1 и Ga2 до грамматик операторов присваивания, условного оператора и оператора цикла, добавляя новые нетерминалы и правила по технологии разработки «снизу вверх». Изучить свойства полученных грамматик и их символов.

5. Используя полученные навыки работы с грамматиками и программным обеспечением, начать поэтапную разработку грамматики по технологии «сверху вниз» для заданного варианта курсовой работы.

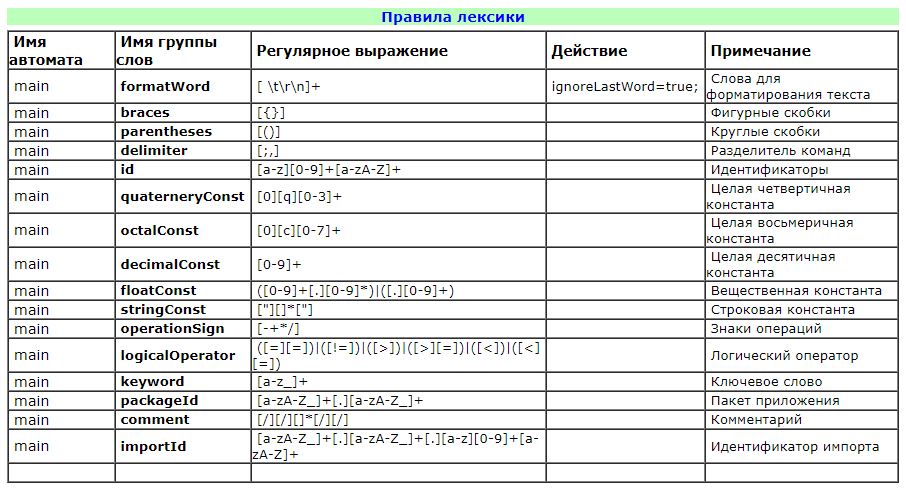
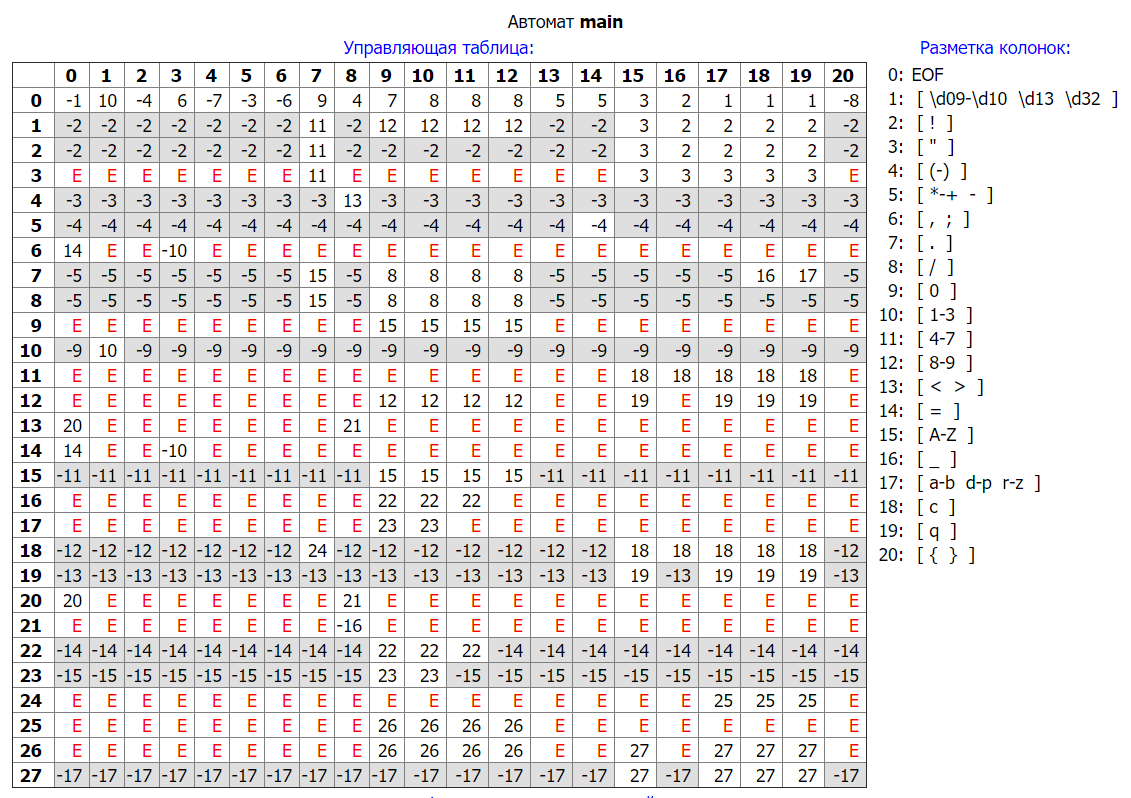
6. Оценить свои субъективные проблемы и трудности разработки формальных грамматик при использовании разных технологий («снизу вверх» и «сверху вниз»). Выбрать технологию для выполнения задания на курсовую работу (рекомендуется технология «сверху вниз»).

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

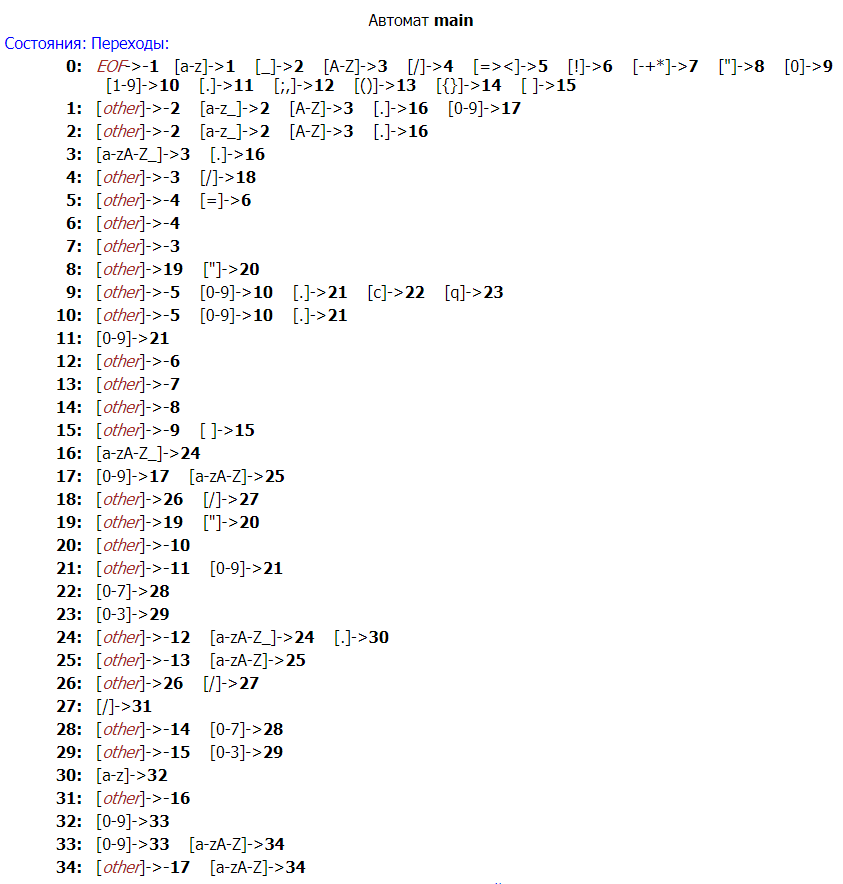
1. Идентификаторы:  
   <б><пЦ><пБ> (a13KL, w8U, …)
2. Константы:  
   целые по основаниям 4, 8 и 10, вещественные, строковые.
3. Объявления примитивных типов:  
   – целое – long, long\_u  
   – вещественное – number  
   – строковые – char
4. Оператор присваивания:  
   put <В> to <И>;
5. Условный 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> ]}

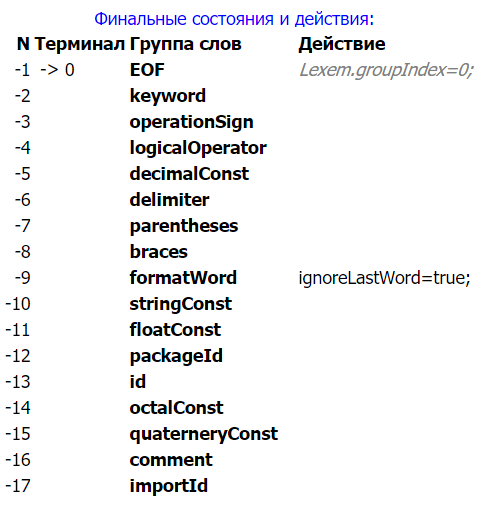
# Правила лексики и лексические КА

Правила лексики языка, написанные во 2 л/р, показаны на рис. 1. Табличный и графовый конечные автоматы показаны на рис. 2 и 3, соответственно. Финальные состояния показаны на рис. 4.

*Рис. 1 – Правила лексики*

*Рис. 2 – Табличный конечный автомат*

*Рис. 3 – Графовый конечный автомат*



*Рис. 4 – Финальные состояния*

# Разбор фрагмента программы

Рассмотрим работу конечных автоматов по разбору фрагмента программы:

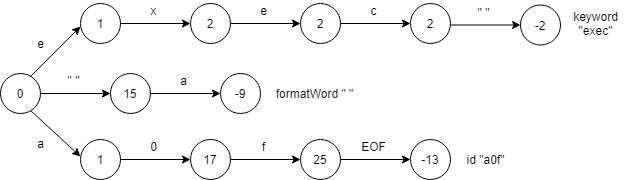
*exec a0f*

**Работа табличного конечного автомата:**

(0, 17) → (1, 17) → (2, 17) → (2, 18) → (2, 1) → -2 (keyword «exec»)

(0, 1) → (10, 17) → -9 (formatWord « »)

(0, 17) → (1, 9) → (12, 17) → (19, 0) → -13 (id «a0f»)

**Работа графового конечного автомата:**

*Рис. 5 – Работа графового КА по разбору заданного фрагмента*

# Описание метаязыка формальных грамматик

*Формальной грамматикой G* называется совокупность , состоящая из:

– алфавита терминальных символов ;

– алфавита нетерминальных символов ;

– начального терминального символа ;

– системы правил подстановки .

*Алфавит терминальных символов*  – конечное множество всех слов языка, порождаемого данной грамматикой. Понятие «терминальный» в данном случае обозначает неразложимость таких символов с точки зрения синтаксических правил.

*Алфавит нетерминальных символов*  – конечное множество названий синтаксических конструкций, например: <предложение>, <выражение>, <список аргументов>, <условный оператор>, <тело функции>. Нетерминальные символы используются только в метаязыке, на котором описывается язык программирования.

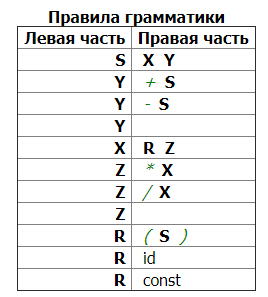
*Начальный нетерминальный символ S* – один из нетерминальных символов. Этим символом обычно обозначается наиболее общая синтаксическая конструкция, например: <правильная программа>.

*Система правил подстановки P* (система порождающих правил или продукций) – конечное множество пар цепочек вида α : β, причем цепочка α (левая часть правила) должна содержать хотя бы один нетерминальный символ.

# Свойства грамматик из примера

Рассмотрим в качестве примера язык, заданные в файле *3LL1.xml*. На рис. 6 показаны правила грамматики этого языка. Грамматика данного языка является контекстно-свободной (в левой части каждого порождающего правила только один нетерминал) и рекурсивной (нетерминальный символ S является косвеннорекурсивным).

Предшественник некоторого символа X – символ, с которого начинается цепочка, выводимая из X. Символ Y является последователем символа X, если хотя бы в одной цепочке ω, выводимой из начального нетерминала грамматики, символ Y непосредственно следует за X. На рис. 7 показана матрица отношения предшествования для данного языка, на рис. 8 – матрица отношения последования для данного языка.



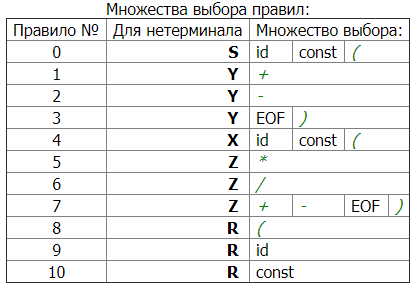
*Рис. 6 – Правила грамматики языка из примера*



*Рис. 7 – Отношение предшествования языка из примера*



*Рис. 8 – Отношение последования языка из примера*

**

*Рис. 8 – Множества выбора правил языка из примера*

Грамматика языка из примера является LL(1)-грамматикой, т.к. множества выбора правил с одинаковым нетерминалом в левой части попарно не пересекаются (рис. 8). Поэтому данная может быть использована для организации нисходящего детерминированного восстановления дерева грамматического разбора предложений языка из примера.

# Разработка грамматики заданного языка

В ходе данной работе также была частично разработана грамматика языка, заданного вариантом. На данный момент, программа может состоять из следующих элементов лексики, описанной в л/р 2:

– Объявление переменной целого, вещественного, строкового или логического типа.

– Присвоение значения переменной. В качестве значения может выступать любое корректное выражение (см. ниже).

– Условный оператор. Для корректной работы условного оператора в лексику языка был добавлен логический тип *bool* (может принимать значения *true* или *false*) и логическое выражение (см. ниже).

1– Оператор цикла. В качестве аргументов оператора (значения после *from*, *to*, *step*) может быть указана целочисленная константа или переменная.

– Оператор переключателя. В качестве аргумента *case* может быть указано любое корректное выражение (см. ниже), в качестве аргумента *when* – константа.

– Оператор возврата значения. Позволяет вернуть корректное выражение в качестве результата выполнения программы (после доработки грамматики – в качестве результата выполнения функции).

– Вызова функции. При вызове возможна передача любого количества аргументов, в качестве которых могут выступать любые корректные выражения (см. ниже).

В качестве выражения могут выступать следующие элементы:

– Строковая константа

– Арифметическая константа (целая или вещественная константа)

– Арифметическое выражение – может состоять из арифметических констант, обращений к переменным и вызовов функций, знаков «+» и «-» и круглых скобок, выражение разработано на основе грамматики Ga1, взятой из примера

– Логическая константа (*true* или *false*)

– Логическое выражение – может состоять из логических констант, обращений к переменным и вызовов функций, сравнений (с использованием операторов сравнения) и знаков «||», «&&» и «!» для составления комплексных логических выражений, выражение разработано на основе грамматики Ga1, взятой из примера

– Обращение к переменной

– Вызов функции

Правила синтаксиса разработанной части языка представлены в приложении A. Множества выбора правил попарно не пересекаются, поэтому разработанная грамматика является LL(1)-грамматикой. Она может быть использована для организации нисходящего детерминированного восстановления дерева грамматического разбора предложений заданного языка.

# Тестовая программа

long x0x;

put 1 to 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;

};

long\_u a0a;

put 0q2 to 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 a0b;

put 0q10 to a0b;

a0print(f4ctorial(a0b));

s4yNumber(0c1);

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

# Вывод

В ходе выполнения лабораторной работы был изучены основные понятия метаязыка формальных грамматик, рассмотрена грамматика языка из примера и разработана часть грамматики языка, заданного вариантом.

# Приложение А. Правила синтаксиса

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Левая часть** | **Правая часть** | | | | | | | | | | | | | **Примечание** |
| **S** | [ | **Statement** | ]\* | | | | | | | | | | | Код программы |
| **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** | id | [ | **FunCall** | ]? | [ | **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 функции/ перем./конст. |
| **ArythmExprACC** | **AY** | | | | | | | | | | | | |  |
| **ArythmExprWCC** | *(* | **ArythmExpr** | *)* | [ | **AZ** | ]? | [ | **AY** | ]? | | | | | Арифм. выраж. без функции/ 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 функции/ перем./конст. |
| **LogicExprACC1** | **LZ** | [ | **LY** | ]? | | | | | | | | | |  |
| **LogicExprACC1** | **LY** | | | | | | | | | | | | |  |
| **LogicExprWCC** | *!* | **LRnot** | [ | **LZ** | ]? | [ | **LY** | ]? | | | | | | Лог. выраж. без функции/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 |