

Министерство науки и высшего образования РФ  
Федеральное государственное автономное  
образовательное учреждение высшего образования  
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий  
институт  
Программная инженерия  
кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №5  
Синтаксический анализ контекстно-свободных языков  
тема

Преподаватель

подпись, дата

А. С. Кузнецов

инициалы, фамилия

Студент КИ23-17/1Б, 032320072

номер группы, зачетной книжки

подпись, дата

М. А. Мальцев

инициалы, фамилия

Красноярск 2025

## 1 Цель

Исследование контекстно-свободных грамматик и алгоритмов синтаксического анализа контекстно-свободных языков.

## 2 Задания

### Часть 1.

Необходимо с использованием системы JFLAP, построить LL(1)-грамматику, описывающую заданный язык, или формально доказать невозможность этого. Полученная грамматика не должна повторять SLR(1)-грамматику, конструируемую в части 3. Ассоциативность операций на усмотрение разработчика.

### Вариант 2.

Язык оператора присваивания, в правой части которого задано логическое выражение. Элементами выражений являются целочисленные константы в шестнадцатеричной системе счисления, имена переменных из одного символа (от g до k), знаки операций и скобки для изменения порядка вычисления подвыражений. Операции (в сторону уменьшения приоритета): отрицание, мультипликативные, аддитивные, присваивание.

### Часть 2.

Предложить программную реализацию метода рекурсивного спуска для распознавания строк заданного языка. Представить формальное доказательство принадлежности к классу LL(1)-грамматики, лежащей в основе синтаксического анализа заданного языка. Во всех случаях язык должен состоять из последовательностей выражений. В качестве разделителя может выступать символ новой строки, точка с запятой или любой другой символ, не задействованный в других лексемах. Ассоциативность операций на усмотрение разработчика. Результатом работы синтаксического анализатора является выдача сообщения «Accepted» или «Rejected».

### Вариант 2.

Язык логических выражений, элементами которых являются целочисленные константы в шестнадцатеричной, двоичной или десятичной системах счисления, имена переменных из 1-2 символов, знаки операций и скобки для изменения порядка вычисления подвыражений. Операции (в сторону уменьшения приоритета): отрицание, мультипликативные, аддитивные, присваивание

### Часть 3.

Необходимо с использованием системы JFLAP, построить SLR(1)-грамматику, описывающую заданный язык, или формально доказать невозможность этого. Во всех случаях реализуется язык, состоящий из последовательностей операторов присваивания. В качестве разделителя может выступать символ новой строки, точка с запятой или любой другой символ, не задействованный в прочих лексемах. В качестве L-значения оператора присваивания выступает только имя переменной. В правой части оператора присваивания указывается выражение, элементы которых оговариваются в каждом варианте задания. Ассоциативность операций на усмотрение разработчика. Полученная грамматика не должна повторять LL(1)-грамматику, конструируемую в части 1.

### Вариант 2.

Элементами логического выражения являются целочисленные константы в 8- и 16-чной системах счисления, имена переменных из одного символа (от g до k), знаки операций и скобки для изменения порядка вычисления подвыражений. Операции (в сторону уменьшения приоритета): отрицание, мультипликативные, аддитивные, присваивание.

## 3 Ход выполнения

### 3.1 Создание LL(1)-грамматики

Сначала была сделана первая часть задания практической работы. За знаки операций были приняты следующие: «~» (отрицание), «|» (аддитивная, ИЛИ),

«&» (мультипликативная, И) и «=» (присваивание). Были использованы круглые скобки. Составленная LL(1)-грамматика в JFLAP показана на рисунке 1.

LHS		RHS
S	→	V=E
V	→	g
V	→	h
V	→	i
V	→	j
V	→	k
E	→	A
A	→	MB
B	→	MB
B	→	ε
M	→	UN
N	→	&UN
N	→	ε
U	→	~U
U	→	P
P	→	V
P	→	C
C	→	HD
D	→	HD
D	→	ε
H	→	0
H	→	1
H	→	2
H	→	3
H	→	4
H	→	5
H	→	6
H	→	7
H	→	8
H	→	9
H	→	a
H	→	b
H	→	c
H	→	d
H	→	e
H	→	f
P	→	(E)

Рисунок 1 – Составленная LL(1)-грамматика

Затем был выполнен «Build LL(1) Parse» и получено множество первых порождаемых символов и символов последователей и таблица синтаксического анализа. Они показаны на рисунке 2 и 3.

	FIRST	FOLLOW
A	{ a, b, c, d, e, f, g, h, (, i, j, k, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ~ }	{ \$, ) }
B	{ ε,   }	{ \$, ) }
C	{ a, b, c, d, e, f, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }	{ \$, &, ),   }
D	{ ε, a, b, c, d, e, f, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }	{ \$, &, ),   }
E	{ a, b, c, d, e, f, g, h, (, i, j, k, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ~ }	{ \$, ) }
H	{ a, b, c, d, e, f, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }	{ a, b, c, \$, d, e, &, f, ), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,   }
M	{ a, b, c, d, e, f, g, h, (, i, j, k, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ~ }	{ \$, ),   }
N	{ ε, & }	{ \$, ),   }
P	{ a, b, c, d, e, f, g, h, (, i, j, k, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }	{ \$, &, ),   }
S	{ g, h, i, j, k }	{ \$ }
U	{ a, b, c, d, e, f, g, h, (, i, j, k, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ~ }	{ \$, &, ),   }
V	{ g, h, i, j, k }	{ \$, &, ),  , = }

Рисунок 2 – Множество первых порождаемых символов и символов последователей составленной грамматики

	&	(	)	0	1	2	3	4	5	6	7	8	9	=	a	b	c	d	e	f	g	h	i	j	k		~	\$	
A		MB		MB	MB	MB	MB	MB	MB	MB	MB	MB	MB		MB	MB	MB	MB	MB	MB	MB	MB	MB	MB	MB	MB			
B		ε																							IMB		ε		
C				HD	HD	HD	HD	HD	HD	HD	HD	HD	HD		HD	HD	HD	HD	HD	HD									
D	ε		ε	HD	HD	HD	HD	HD	HD	HD	HD	HD	HD		HD	HD	HD	HD	HD	HD						ε		ε	
E		A		A	A	A	A	A	A	A	A	A	A		A	A	A	A	A	A	A	A	A	A	A		A		
H				0	1	2	3	4	5	6	7	8	9		a	b	c	d	e	f									
M		UN		UN	UN	UN	UN	UN	UN	UN	UN	UN	UN		UN	UN	UN	UN	UN	UN	UN	UN	UN	UN	UN	UN		UN	
N &UN		ε																								ε		ε	
P		(E)		C	C	C	C	C	C	C	C	C	C		C	C	C	C	C	C	V	V	V	V	V				
S																					V=E	V=E	V=E	V=E	V=E				
U		P		P	P	P	P	P	P	P	P	P	P		P	P	P	P	P	P	P	P	P	P	P		~U		
V																					g	h	i	j	k				

Рисунок 3 – Таблица синтаксического анализа составленной грамматики

Как можно заметить, в каждой ячейке таблицы не более одной записи, что говорит о принадлежности грамматики LL(1)-грамматике.

Дальше было проведено распознавание тестовых цепочек, путём нажатия на кнопку «Parse» и ввода 6 тестовых цепочек для проверки. Результаты теста показаны на рисунках 4, 5, 6, 7, 8 и 9.

Start	Step	Derivation Table
Input	g= ~(h a2)&c2	
Input Remaining	\$	
Stack		
Input Field Text Size (For optimization, move one of the window size adjusters around this window after resizing the text fields)		
Table Text Size		
S→V=E		S
V→g		V=E
E→A		g=E
A→MB		g=A
M→UN		g=MB
U→~U		g=UNB
U→P		g=~UNB
P→(E)		g=~PNB
E→A		g=~(E)NB
A→MB		g=~(A)NB
M→UN		g=~(MB)NB
U→P		g=~(UNB)NB
P→V		g=~(PNB)NB
V→h		g=~(VNB)NB
N→ε		g=~(hNB)NB
B→ MB		g=~(hB)NB
M→UN		g=~(h MB)NB
U→P		g=~(h UNB)NB
P→C		g=~(h PNB)NB
C→HD		g=~(h CNB)NB
H→a		g=~(h HDNB)NB
D→HD		g=~(h aDNB)NB
H→2		g=~(h aHDNB)NB
D→ε		g=~(h a2DNB)NB
N→ε		g=~(h a2NB)NB
B→ε		g=~(h a2B)NB
N→&UN		g=~(h a2)NB
U→P		g=~(h a2)&UNB
P→C		g=~(h a2)&PNB
C→HD		g=~(h a2)&CNB
H→c		g=~(h a2)&HDNB
D→HD		g=~(h a2)&cDNB
H→2		g=~(h a2)&cHDNB
D→ε		g=~(h a2)&c2DNB
N→ε		g=~(h a2)&c2NB
B→ε		g=~(h a2)&c2B
		g=~(h a2)&c2

Рисунок 4 – Тест для цепочки « $g \sim (h|a2) \& c2$ »

Start	Step	Derivation Table
Input	g=h	
Input Remaining	\$	
Stack		
Input Field Text Size (For optimization, move one of the window size adjusters around this window after resizing the text fields)		
Table Text Size		
S→V=E		S
V→g		V=E
E→A		g=E
A→MB		g=A
M→UN		g=MB
U→P		g=UNB
P→V		g=PNB
V→h		g=VNB
N→ε		g=hNB
B→ε		g=hB
		g=h

Рисунок 5 – Тест для цепочки « $g=h$ »

Start	Step	Derivation Table
Input	k~a23	
Input Remaining	\$	
Stack		
Input Field Text Size (For optimization, move one of the window size adjusters around this window after resizing the text fields)		
Table Text Size		
		S
S→V=E		V=E
V→k		k=E
E→A		k=A
A→MB		k=MB
M→UN		k=UNB
U→~U		k=~UNB
U→P		k=~PNB
P→C		k=~CNB
C→HD		k=~HDNB
H→a		k=~aDNB
D→HD		k=~aHDNB
H→2		k=~a2DNB
D→HD		k=~a2HDNB
H→3		k=~a23DNB
D→ε		k=~a23NB
N→ε		k=~a23B
B→ε		k=~a23

Рисунок 6 – Тест для цепочки « $k \sim a23$ »

Start	Step	Derivation Table
Input	j=(k 12))	
Input Remaining	)\$	
Stack		
Input Field Text Size (For optimization, move one of the window size adjusters around this window after resizing the text fields)		
Table Text Size		
		S
S→V=E		V=E
V→j		j=E
E→A		j=A
A→MB		j=MB
M→UN		j=UNB
U→P		j=PNB
P→(E)		j=(E)NB
E→A		j=(A)NB
A→MB		j=(MB)NB
M→UN		j=(UNB)NB
U→P		j=(PNB)NB
P→V		j=(VNB)NB
V→k		j=(kNB)NB
N→ε		j=(kB)NB
B→ MB		j=(k MB)NB
M→UN		j=(k UNB)NB
U→P		j=(k PNB)NB
P→C		j=(k CNB)NB
C→HD		j=(k HDNB)NB
H→1		j=(k 1DNB)NB
D→HD		j=(k 1HDNB)NB
H→2		j=(k 12DNB)NB
D→ε		j=(k 12NB)NB
N→ε		j=(k 12B)NB
B→ε		j=(k 12)NB
N→ε		j=(k 12)B
B→ε		j=(k 12)

Рисунок 7 – Тест для цепочки « $j=(k|12))$ »

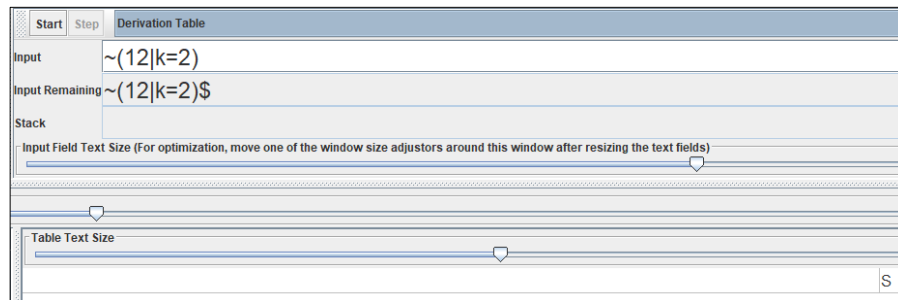


Рисунок 8 – Тест для цепочки « $\sim(12|k=2)$ »

Start	Step	Derivation Table
Input		$h=\sim j\&$
Input Remaining		$\$$
Stack		NB
Input Field Text Size (For optimization, move one of the window size adjusters around this window after resizing the text fields)		
Table Text Size		
S $\rightarrow$ V=E		S
V $\rightarrow$ h		V=E
E $\rightarrow$ A		h=E
A $\rightarrow$ MB		h=A
M $\rightarrow$ UN		h=MB
U $\rightarrow$ ~U		h=UNB
U $\rightarrow$ P		h=~UNB
P $\rightarrow$ V		h=~PNB
V $\rightarrow$ j		h=~VNB
N $\rightarrow$ &UN		h=~jNB
		h=~j&UNB

Рисунок 9 – Тест для цепочки « $h=\sim j\&$ »

В итоге все тесты были успешно пройдены, и полученная грамматика правильно определяет исходный язык.

### 3.2 Создание SLR(1)-грамматики

Затем была выполнена 3 часть задания практической работы. В качестве знаков операций были приняты те же знаки, что и в прошлой части задания. В качестве разделителя использовался «;». Составленная SLR(1)-грамматика в JFLAP показана на рисунке 10.



LHS	RHS	LHS	RHS
Z	→ L	O	→ 1
L	→ S	O	→ 2
L	→ L;S	O	→ 3
S	→ I=E	O	→ 4
E	→ A	O	→ 5
E	→ I=E	O	→ 6
A	→ M	O	→ 7
A	→ A M	J	→ H
M	→ U	J	→ JH
M	→ M&U	H	→ 0
U	→ P	H	→ 1
U	→ ~U	H	→ 2
P	→ I	H	→ 3
P	→ C	H	→ 4
P	→ (A)	H	→ 5
I	→ g	H	→ 6
I	→ h	H	→ 7
I	→ i	H	→ 8
I	→ j	H	→ 9
I	→ k	H	→ a
C	→ 0Q	H	→ b
C	→ 0xJ	H	→ c
Q	→ O	H	→ d
Q	→ QO	H	→ e
O	→ 0	H	→ f

Рисунок 10 – Составленная RLS(1)-грамматика

Затем был выполнен «Build SLR(1) Parse» и получено множество первых порождаемых символов и символов последователей, канонический набор LR(0)-ситуаций и таблица синтаксического анализа. Они показаны на рисунке 11, 12 и 13.

	FIRST	FOLLOW
A	{0, g, (, h, i, j, k, ~}	{\$, ), ,,  }
C	{0}	{\$, & ), ,,  }
E	{0, g, h, (, i, j, k, ~}	{\$, ;}
H	{a, b, c, d, e, f, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}	{a, b, c, \$, d, e, & f, ), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ,,  }
I	{g, h, i, j, k}	{\$, & ), ,,  , =}
J	{a, b, c, d, e, f, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}	{a, b, c, \$, d, e, & f, ), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ,,  }
L	{g, h, i, j, k}	{\$, ;}
M	{0, g, (, h, i, j, k, ~}	{\$, & ), ,,  }
O	{0, 1, 2, 3, 4, 5, 6, 7}	{\$, & ), 0, 1, 2, 3, 4, 5, 6, 7, ,,  }
P	{0, g, (, h, i, j, k}	{\$, & ), ,,  }
Q	{0, 1, 2, 3, 4, 5, 6, 7}	{\$, & ), 0, 1, 2, 3, 4, 5, 6, 7, ,,  }
S	{g, h, i, j, k}	{\$, ;}
U	{0, g, (, h, i, j, k, ~}	{\$, & ), ,,  }
Z	{g, h, i, j, k}	{\$}

Рисунок 11 – Множество первых порождаемых символов и символов последователей составленной грамматики

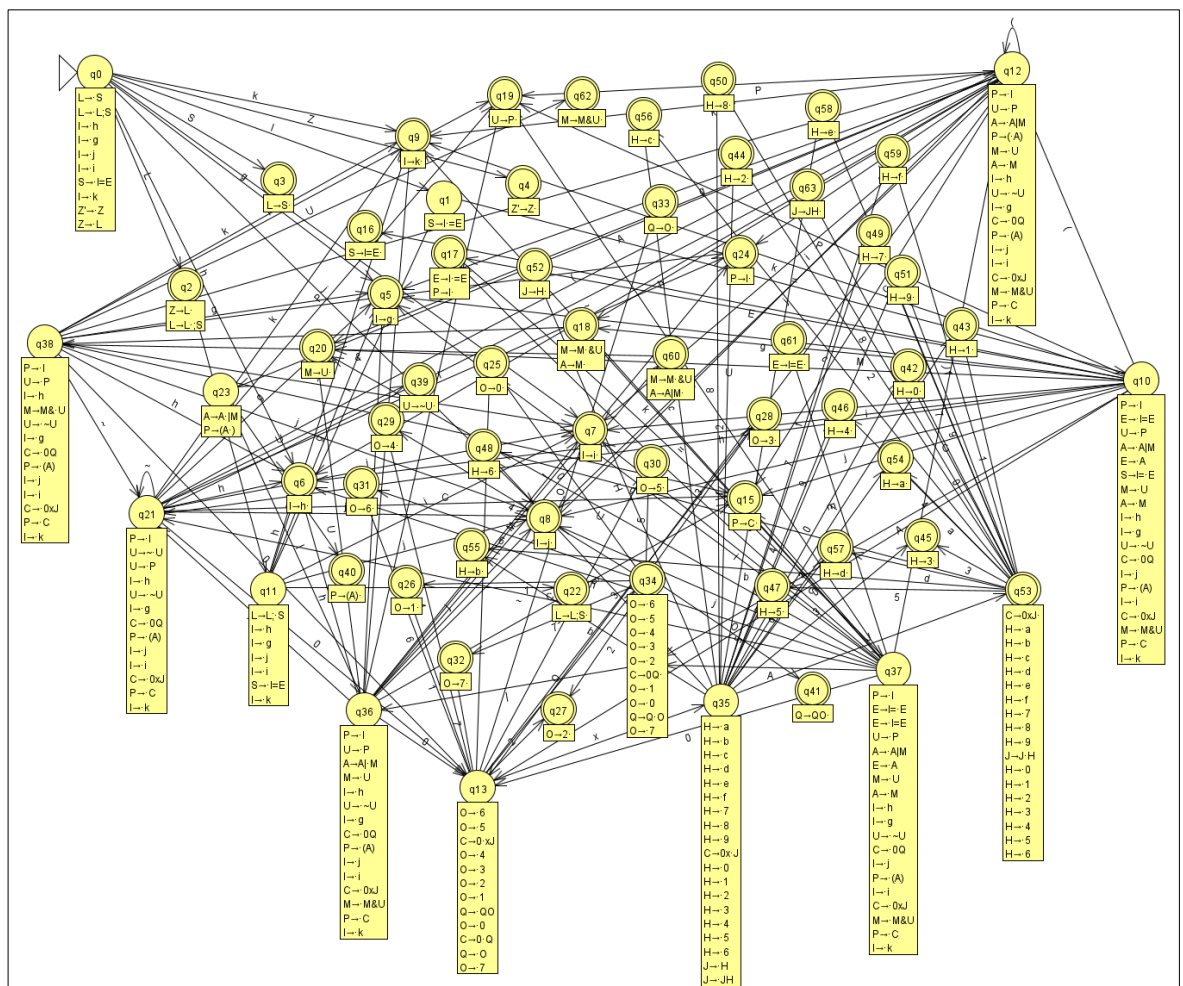


Рисунок 12 – Канонический набор LR(0)-ситуаций

[illegible]

Рисунок 13 – Таблица синтаксического анализа составленной грамматики

Дальше было проведено распознавание тестовых цепочек, путём нажатия на кнопку «Parse» и ввода 6 тестовых цепочек для проверки. Результаты теста показаны на рисунках 14, 15, 16, 17, 18 и 19.

Start Step Derivation Table	
Input	$g \sim (h 071) \& 0xa2f$
Input Remaining	\$
Stack	Z0
Input Field Text Size (For optimization, move one of the window size a	
Table Text Size	
	$g \sim (h 071) \& 0xa2f$
$I \rightarrow g$	$I \sim (h 071) \& 0xa2f$
$I \rightarrow h$	$I \sim (I 071) \& 0xa2f$
$P \rightarrow I$	$I \sim (P 071) \& 0xa2f$
$U \rightarrow P$	$I \sim (U 071) \& 0xa2f$
$M \rightarrow U$	$I \sim (M 071) \& 0xa2f$
$A \rightarrow M$	$I \sim (A 071) \& 0xa2f$
$O \rightarrow 7$	$I \sim (A 001) \& 0xa2f$
$Q \rightarrow O$	$I \sim (A 0Q1) \& 0xa2f$
$O \rightarrow 1$	$I \sim (A 0QO) \& 0xa2f$
$Q \rightarrow QO$	$I \sim (A 0Q) \& 0xa2f$
$C \rightarrow 0Q$	$I \sim (A C) \& 0xa2f$
$P \rightarrow C$	$I \sim (A P) \& 0xa2f$
$U \rightarrow P$	$I \sim (A U) \& 0xa2f$
$M \rightarrow U$	$I \sim (A M) \& 0xa2f$
$A \rightarrow A M$	$I \sim (A) \& 0xa2f$
$P \rightarrow (A)$	$I \sim P \& 0xa2f$
$U \rightarrow P$	$I \sim U \& 0xa2f$
$U \rightarrow \sim U$	$I = U \& 0xa2f$
$M \rightarrow U$	$I = M \& 0xa2f$
$H \rightarrow a$	$I = M \& 0xH2f$
$J \rightarrow H$	$I = M \& 0xJ2f$
$H \rightarrow 2$	$I = M \& 0xJHf$
$J \rightarrow JH$	$I = M \& 0xJf$
$H \rightarrow f$	$I = M \& 0xJH$
$J \rightarrow JH$	$I = M \& 0xJ$
$C \rightarrow 0xJ$	$I = M \& C$
$P \rightarrow C$	$I = M \& P$
$U \rightarrow P$	$I = M \& U$
$M \rightarrow M \& U$	$I = M$
$A \rightarrow M$	$I = A$
$E \rightarrow A$	$I = E$
$S \rightarrow I = E$	$S$
$L \rightarrow S$	$L$
$Z \rightarrow L$	$Z$

Рисунок 14 – Тест для цепочки « $g \sim (h|071) \& 0xa2f$ »

Start	Step	Derivation Table
Input	k=j=02&g	
Input Remaining	\$	
Stack	Z0	
Input Field Text Size (For optimization, move one of the window size adjustment sliders)		
Table Text Size		
	k=j=02&g	
I→k	I=j=02&g	
I→j	I=I=02&g	
O→2	I=I=0O&g	
Q→O	I=I=0Q&g	
C→0Q	I=I=C&g	
P→C	I=I=P&g	
U→P	I=I=U&g	
M→U	I=I=M&g	
I→g	I=I=M&I	
P→I	I=I=M&P	
U→P	I=I=M&U	
M→M&U	I=I=M	
A→M	I=I=A	
E→A	I=I=E	
E→I=E	I=E	
S→I=E	S	
L→S	L	
Z→L	Z	

Рисунок 15 – Тест для цепочки «k=j=02&g»

Start Step Derivation Table	
Input	$k=\sim h 0x1;j=k\&07$
Input Remaining	\$
Stack	Z0
Input Field Text Size (For optimization, move one of the window size ad	
Table Text Size	
	$k=\sim h 0x1;j=k\&07$
$I \rightarrow k$	$I=\sim h 0x1;j=k\&07$
$I \rightarrow h$	$I=\sim I 0x1;j=k\&07$
$P \rightarrow I$	$I=\sim P 0x1;j=k\&07$
$U \rightarrow P$	$I=\sim U 0x1;j=k\&07$
$U \rightarrow \sim U$	$I=U 0x1;j=k\&07$
$M \rightarrow U$	$I=M 0x1;j=k\&07$
$A \rightarrow M$	$I=A 0x1;j=k\&07$
$H \rightarrow 1$	$I=A 0xH;j=k\&07$
$J \rightarrow H$	$I=A 0xJ;j=k\&07$
$C \rightarrow 0xJ$	$I=A C;j=k\&07$
$P \rightarrow C$	$I=A P;j=k\&07$
$U \rightarrow P$	$I=A U;j=k\&07$
$M \rightarrow U$	$I=A M;j=k\&07$
$A \rightarrow A M$	$I=A;j=k\&07$
$E \rightarrow A$	$I=E;j=k\&07$
$S \rightarrow I=E$	$S;j=k\&07$
$L \rightarrow S$	$L;j=k\&07$
$I \rightarrow j$	$L;I=k\&07$
$I \rightarrow k$	$L;I=I\&07$
$P \rightarrow I$	$L;I=P\&07$
$U \rightarrow P$	$L;I=U\&07$
$M \rightarrow U$	$L;I=M\&07$
$O \rightarrow 7$	$L;I=M\&00$
$Q \rightarrow O$	$L;I=M\&0Q$
$C \rightarrow 0Q$	$L;I=M\&C$
$P \rightarrow C$	$L;I=M\&P$
$U \rightarrow P$	$L;I=M\&U$
$M \rightarrow M\&U$	$L;I=M$
$A \rightarrow M$	$L;I=A$
$E \rightarrow A$	$L;I=E$
$S \rightarrow I=E$	$L;S$
$L \rightarrow L;S$	$L$
$Z \rightarrow L$	$Z$

Рисунок 16 – Тест для цепочки « $k=\sim h|0x1;j=k\&07$ »

Start	Step	Derivation Table
Input		<b>g=h;i=02;j=0xa</b>
Input Remaining		<b>\$</b>
Stack		<b>Z0</b>
Input Field Text Size (For optimization, move one of the window size sliders)		
Table Text Size		
		<b>g=h;i=02;j=0xa</b>
I→g		<b>I=h;i=02;j=0xa</b>
I→h		<b>I=l;i=02;j=0xa</b>
P→I		<b>I=P;i=02;j=0xa</b>
U→P		<b>I=U;i=02;j=0xa</b>
M→U		<b>I=M;i=02;j=0xa</b>
A→M		<b>I=A;i=02;j=0xa</b>
E→A		<b>I=E;i=02;j=0xa</b>
S→I=E		<b>S;i=02;j=0xa</b>
L→S		<b>L;i=02;j=0xa</b>
I→i		<b>L;l=02;j=0xa</b>
O→2		<b>L;l=00;j=0xa</b>
Q→O		<b>L;l=0Q;j=0xa</b>
C→0Q		<b>L;l=C;j=0xa</b>
P→C		<b>L;l=P;j=0xa</b>
U→P		<b>L;l=U;j=0xa</b>
M→U		<b>L;l=M;j=0xa</b>
A→M		<b>L;l=A;j=0xa</b>
E→A		<b>L;l=E;j=0xa</b>
S→I=E		<b>L;S;j=0xa</b>
L→L;S		<b>L;j=0xa</b>
I→j		<b>L;l=0xa</b>
H→a		<b>L;l=0xH</b>
J→H		<b>L;l=0xJ</b>
C→0xJ		<b>L;l=C</b>
P→C		<b>L;l=P</b>
U→P		<b>L;l=U</b>
M→U		<b>L;l=M</b>
A→M		<b>L;l=A</b>
E→A		<b>L;l=E</b>
S→I=E		<b>L;S</b>
L→L;S		<b>L</b>
Z→L		<b>Z</b>

Рисунок 17 – Тест для цепочки «g=h;i=02;j=0xa»

Start Step Derivation Table	
Input	$g=\sim k;i$
Input Remaining	\$
Stack	1111;2L0
Input Field Text Size (For optimization, move one of the window s	
Table Text Size	
	$g=\sim k;i$
$l \rightarrow g$	$l=\sim k;i$
$l \rightarrow k$	$l=\sim l;i$
$P \rightarrow l$	$l=\sim P;i$
$U \rightarrow P$	$l=\sim U;i$
$U \rightarrow \sim U$	$l=U;i$
$M \rightarrow U$	$l=M;i$
$A \rightarrow M$	$l=A;i$
$E \rightarrow A$	$l=E;i$
$S \rightarrow l=E$	$S;i$
$L \rightarrow S$	$L;i$
$l \rightarrow i$	$L;l$

Рисунок 17 – Тест для цепочки « $g=\sim k;i$ »

Start Step Derivation Table	
Input	$g=\sim j=k$
Input Remaining	$=k\$$
Stack	24l21~10=1l0
Input Field Text Size (For optimization, move one of the window s	
Table Text Size	
	$g=\sim j=k$
$l \rightarrow g$	$l=\sim j=k$
$l \rightarrow j$	$l=\sim l=k$

Рисунок 17 – Тест для цепочки « $g=\sim j=k$ »



В итоге все тесты были успешно пройдены, и полученная грамматика правильно определяет исходный язык.

### 3.3 Программная реализация синтаксического анализатора методом рекурсивного спуска

Затем была выполнена 2 часть задания практической работы. Сначала была составлена LL(1)-грамматика, определяющая язык, описанный во 2 части задания, которая затем легла в основу разработанного синтаксического анализатора. Описание составленной грамматики показано на рисунке 18.

```
/*!  
*   LIST -> ASSIGN LIST_TAIL  
*   LIST_TAIL -> ε  
*   LIST_TAIL -> ; ASSIGN LIST_TAIL  
*   ASSIGN -> EXPR ASSIGN_TAIL  
*   ASSIGN_TAIL -> = ASSIGN  
*   ASSIGN_TAIL -> ε  
*  
*   EXPR -> NEXT_EXPR ADD  
*   ADD -> + NEXT_EXPR ADD  
*   ADD -> ε  
*  
*   NEXT_EXPR -> UNARY MUL  
*   MUL -> * UNARY MUL  
*   MUL -> ε  
*  
*   UNARY -> ~ UNARY  
*   UNARY -> PRIMARY  
*  
*   PRIMARY -> ID  
*   PRIMARY -> CONST  
*   PRIMARY -> ( EXPR )  
*  
*   ID -> ONE_SYM SEC_SYM  
*   SEC_SYM -> ONE_SYM  
*   SEC_SYM -> ε  
*   ONE_SYM -> a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z  
*  
*   CONST -> 0 CONST_TAIL  
*   CONST -> NZDIGIT DIGIT_TAIL  
*   CONST_TAIL -> b BINARY  
*   CONST_TAIL -> x HEX  
*   CONST_TAIL -> DIGIT_TAIL  
*  
*   BINARY -> BIT BIT_TAIL  
*   BIT_TAIL -> BIT BIT_TAIL  
*   BIT_TAIL -> ε  
*   BIT -> 0, 1  
*  
*   HEX -> HEXDIGIT HEX_TAIL  
*   HEX_TAIL -> HEXDIGIT HEX_TAIL  
*   HEX_TAIL -> ε  
*   HEXDIGIT -> DIGIT  
*   HEXDIGIT -> a, b, c, d, e, f  
*  
*   DIGIT_TAIL -> DIGIT DIGIT_TAIL  
*   DIGIT_TAIL -> ε  
*  
*   NZDIGIT -> 1, 2, 3, 4, 5, 6, 7, 8, 9  
*   DIGIT -> NZDIGIT  
*   DIGIT -> 0  
*/
```

Рисунок 18 – Составленная LL(1)-грамматика



После составления грамматики был написан код программы на C++, реализующей синтаксический анализатор методом рекурсивного спуска на основе составленной LL(1)-грамматики. Он показан на рисунках 20, 21 и 22.

```

54 #include <iostream>
55 #include <cctype>
56 #include <cstdlib>
57 #include <cstring>
58
59 // Terminals
60 const int NEG_SIGN = 0; // ~
61 const int MUL_SIGN = 1; // &
62 const int ADD_SIGN = 2; // +
63 const int ASSIGN_SIGN = 3; // =
64 const int LPAR_SIGN = 4; // (
65 const int RPAR_SIGN = 5; // )
66 const int SEMI_SIGN = 6; // ;
67 const int ID_SIGN = 7; // 1-2 symbols [a-zA-Z] (except a,b,c,d,e,f,g,h,x)
68 const int DIGIT_SIGN = 8; // 2, 3, 4, 5, 6, 7, 8, 9 (except 0,1)
69 const int ZERO_SIGN = 9; // 0
70 const int BIT_SIGN = 10; // 1
71 const int HEX_SIGN = 11; // a, c, d, e, f (except b)
72 const int B_SIGN = 12; // b
73 const int X_SIGN = 13; // x
74
75 // End of program
76 const int EOP = 14;
77
78 const int UNDEF = 15;
79
80 // Lexeme class (token)
81 int lexeme = 0;
82
83 // Non terminals
84 void LIST();
85 void LIST_TAIL();
86 void ASSIGN();
87 void ASSIGN_TAIL();
88 void EXPR();
89 void ADD();
90 void NEXT_EXPR();
91 void MUL();
92 void UNARY();
93 void PRIMARY();
94 void ID();
95 void ONE_SYM();
96 void SEC_SYM();
97 void CONST();
98 void CONST_TAIL();
99 void BINARY();
100 void BIT();
101 void BIT_TAIL();
102 void HEX();
103 void HEXDIGIT();
104 void HEX_TAIL();
105 void DIGIT_TAIL();
106 void DIGIT();
107 void NZDIGIT();
108
109 // Reject input line
110 void error();
111
112 // Look ahead lexeme
113 int get_token();
114
115 char g_inputBuffer[1024] = "";
116 char* g_prog = nullptr;
117
118 int main(int argc, char* argv[])
119 {
120     if (1 == argc)
121     {
122         std::cout << "Enter your input line: ";
123         std::cin.getline(g_inputBuffer, 1024);
124     }
125     else
126     {
127         std::strcpy(g_inputBuffer, argv[1]);
128     }
129
130     g_prog = g_inputBuffer;
131     lexeme = get_token();
132     LIST();
133     if (lexeme == EOP)
134     {
135         std::cout << "Accepted.\n";
136     }
137     else
138     {
139         std::cout << "Rejected.\n";
140     }
141     return 0;
142 }
143
144 void LIST()
145 {
146     ASSIGN();
147     LIST_TAIL();
148 }
149
150 void LIST_TAIL()
151 {
152     if (lexeme == SEMI_SIGN)
153     {
154         lexeme = get_token();
155         ASSIGN();
156         LIST_TAIL();
157     }
158     // Min E
159 }
160
161 void ASSIGN()
162 {
163     EXPR();
164     ASSIGN_TAIL();
165 }
166
167 void ASSIGN_TAIL()
168 {
169     if (lexeme == ASSIGN_SIGN)
170     {
171         lexeme = get_token();
172         ASSIGN();
173     }
174     // Min E
175 }
176
177 void EXPR()
178 {
179     NEXT_EXPR();
180     ADD();
181 }
182
183 void ADD()
184 {
185     if (lexeme == ADD_SIGN)
186     {
187         lexeme = get_token();
188         NEXT_EXPR();
189         ADD();
190     }
191     // Min E
192 }
193
194 void NEXT_EXPR()
195 {
196     UNARY();
197     MUL();
198 }
199
200 void MUL()
201 {
202     if (lexeme == MUL_SIGN)
203     {
204         lexeme = get_token();
205         UNARY();
206         MUL();
207     }
208     // Min E
209 }
210
211 void UNARY()
212 {
213     if (lexeme == NEG_SIGN)
214     {
215         lexeme = get_token();
216         UNARY();
217     }
218     else
219     {
220         PRIMARY();
221     }
222 }
223
224 void PRIMARY()
225 {
226     if (lexeme == ID_SIGN || lexeme == HEX_SIGN || lexeme == B_SIGN || lexeme == X_SIGN)
227     {
228         ID();
229     }
230     else if (lexeme == ZERO_SIGN || lexeme == BIT_SIGN || lexeme == DIGIT_SIGN)
231     {
232         CONST();
233     }
234     else if (lexeme == LPAR_SIGN)
235     {
236         lexeme = get_token();
237         EXPR();
238         if (lexeme != RPAR_SIGN)
239         {
240             error();
241         }
242         lexeme = get_token();
243     }
244     else
245     {
246         error();
247     }
248 }

```

Рисунок 20 – Первая часть кода реализованной программы на C++

```

249 void ID()
250 {
251     ONE_SYM();
252     SEC_SYM();
253 }
254 void ONE_SYM()
255 {
256     if (lexeme != ID_SIGN && lexeme != HEX_SIGN && lexeme != B_SIGN && lexeme != X_SIGN) {
257         error();
258     }
259     lexeme = get_token();
260 }
261 void SEC_SYM()
262 {
263     if (lexeme == ID_SIGN || lexeme == HEX_SIGN || lexeme == B_SIGN || lexeme == X_SIGN) {
264         ONE_SYM();
265     }
266     // Wnn ε
267 }
268 void CONST()
269 {
270     if (lexeme == ZERO_SIGN)
271     {
272         lexeme = get_token();
273         CONST_TAIL();
274     }
275     else if (lexeme == DIGIT_SIGN || lexeme == BIT_SIGN)
276     {
277         lexeme = get_token();
278         DIGIT_TAIL();
279     }
280     else
281     {
282         error();
283     }
284 }
285 void CONST_TAIL()
286 {
287     if (lexeme == B_SIGN)
288     {
289         lexeme = get_token();
290         BINARY();
291     }
292     else if (lexeme == X_SIGN)
293     {
294         lexeme = get_token();
295         HEX();
296     }
297     else
298     {
299         DIGIT_TAIL();
300     }
301 }
302 void BINARY()
303 {
304     BIT();
305     BIT_TAIL();
306 }
307 void BIT()
308 {
309     if (lexeme == ZERO_SIGN || lexeme == BIT_SIGN)
310     {
311         lexeme = get_token();
312     }
313     else
314     {
315         error();
316     }
317 }
318 void BIT_TAIL()
319 {
320     if (lexeme == ZERO_SIGN || lexeme == BIT_SIGN)
321     {
322         BIT();
323         BIT_TAIL();
324     }
325     // Wnn ε
326 }
327 void HEX()
328 {
329     HEXDIGIT();
330     HEX_TAIL();
331 }
332 void HEXDIGIT()
333 {
334     if (lexeme == DIGIT_SIGN || lexeme == BIT_SIGN || lexeme == ZERO_SIGN || lexeme == HEX_SIGN || lexeme == B_SIGN)
335     {
336         lexeme = get_token();
337     }
338     else
339     {
340         error();
341     }
342 }
343 void HEX_TAIL()
344 {
345     if (lexeme == DIGIT_SIGN || lexeme == BIT_SIGN || lexeme == ZERO_SIGN || lexeme == HEX_SIGN || lexeme == B_SIGN)
346     {
347         HEXDIGIT();
348         HEX_TAIL();
349     }
350     // Wnn ε
351 }
352 void DIGIT()
353 {
354     if (lexeme == ZERO_SIGN)
355     {
356         lexeme = get_token();
357     }
358     else
359     {
360         NZDIGIT();
361     }
362 }
363 void NZDIGIT()
364 {
365     if (lexeme == DIGIT_SIGN || lexeme == BIT_SIGN || lexeme == ZERO_SIGN)
366     {
367         lexeme = get_token();
368     }
369     else
370     {
371         error();
372     }
373 }
374 void NZDIGIT_TAIL()
375 {
376     if (lexeme == DIGIT_SIGN || lexeme == BIT_SIGN || lexeme == ZERO_SIGN)
377     {
378         lexeme = get_token();
379         NZDIGIT();
380     }
381     // Wnn ε
382 }

```

Рисунок 21 – Вторая часть кода реализованной программы на C++

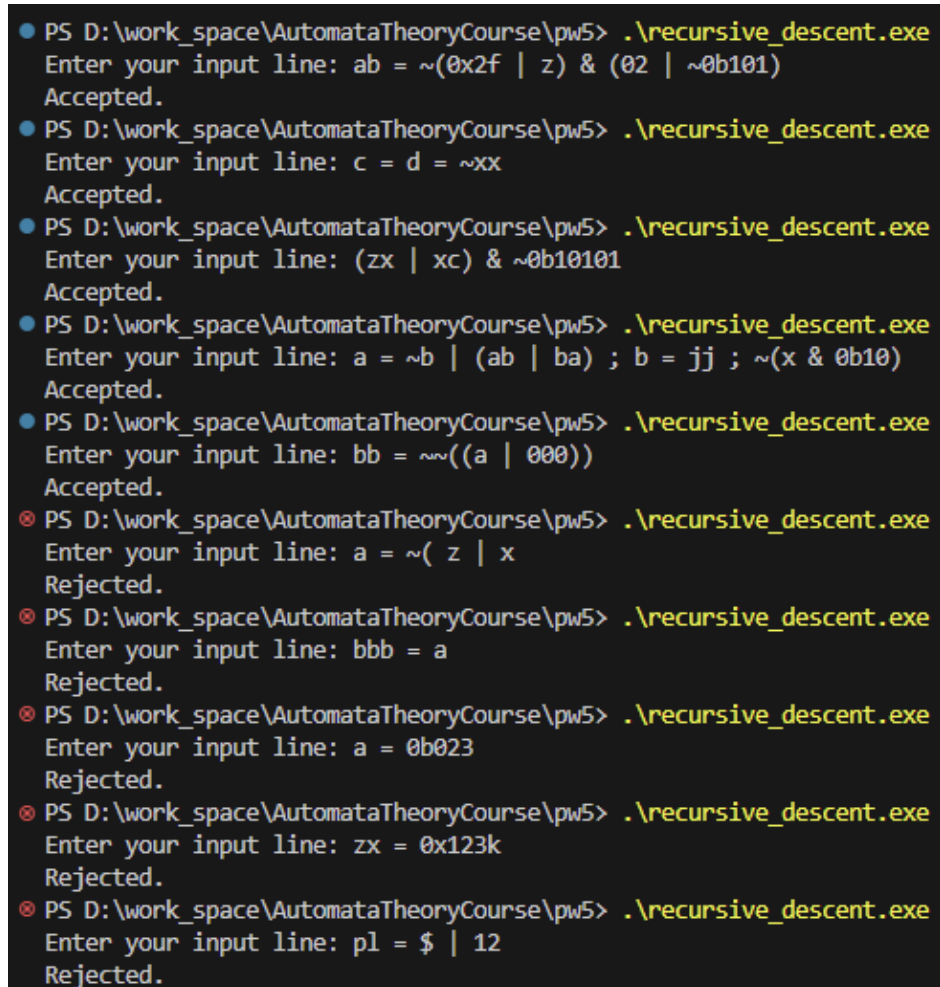
```

375 void DIGIT_TAIL()
376 {
377     if (lexeme == DIGIT_SIGN || lexeme == BIT_SIGN || lexeme == ZERO_SIGN)
378     {
379         DIGIT();
380         DIGIT_TAIL();
381     }
382     // Wnn ε
383 }
384 void NZDIGIT()
385 {
386     if (lexeme == DIGIT_SIGN || lexeme == BIT_SIGN)
387     {
388         lexeme = get_token();
389     }
390     else
391     {
392         error();
393     }
394 }
395 int get_token()
396 {
397     char token[132] = "";
398     char* tok = token;
399     // Skip spaces
400     while (std::isspace(*g_prog))
401         ++g_prog;
402     // End line marker
403     if (*g_prog == '\0')
404         return EOP;
405     // Keywords
406     if (*g_prog == '-')
407     {
408         ++g_prog;
409         return NEG_SIGN;
410     }
411     if (*g_prog == '&')
412     {
413         ++g_prog;
414         return MUL_SIGN;
415     }
416     if (*g_prog == '|')
417     {
418         ++g_prog;
419         return ADD_SIGN;
420     }
421     if (*g_prog == '=')
422     {
423         ++g_prog;
424         return ASSIGN_SIGN;
425     }
426     if (*g_prog == '(')
427     {
428         ++g_prog;
429         return LPAR_SIGN;
430     }
431     if (*g_prog == ')')
432     {
433         ++g_prog;
434         return RPAR_SIGN;
435     }
436     if (*g_prog == ';')
437     {
438         ++g_prog;
439         return SEMI_SIGN;
440     }
441     if (*g_prog == ',')
442     {
443         ++g_prog;
444         return COMMA_SIGN;
445     }
446     if ((*g_prog >= 'h' && *g_prog < 'x') || (*g_prog >= 'y' && *g_prog <= 'z'))
447     {
448         ++g_prog;
449         return ID_SIGN;
450     }
451     if (*g_prog >= '2' && *g_prog <= '9')
452     {
453         ++g_prog;
454         return DIGIT_SIGN;
455     }
456     if (*g_prog == '0')
457     {
458         ++g_prog;
459         return ZERO_SIGN;
460     }
461     if (*g_prog == '1')
462     {
463         ++g_prog;
464         return BIT_SIGN;
465     }
466     if (*g_prog == 'a' || (*g_prog >= 'c' && *g_prog <= 'f'))
467     {
468         ++g_prog;
469         return HEX_SIGN;
470     }
471     if (*g_prog == 'b')
472     {
473         ++g_prog;
474         return B_SIGN;
475     }
476     if (*g_prog == 'x')
477     {
478         ++g_prog;
479         return X_SIGN;
480     }
481     return UNDEF;
482 }
483 void error()
484 {
485     std::cout << "Rejected.\n";
486     std::exit(EXIT_FAILURE);
487 }

```

Рисунок 22 – Третья часть кода реализованной программы на C++

На рисунке 23 показаны тестовые примеры работы программы на тестовых цепочках.



```
PS D:\work_space\AutomataTheoryCourse\pw5> .\recursive_descent.exe
Enter your input line: ab = ~(0x2f | z) & (02 | ~0b101)
Accepted.
PS D:\work_space\AutomataTheoryCourse\pw5> .\recursive_descent.exe
Enter your input line: c = d = ~xx
Accepted.
PS D:\work_space\AutomataTheoryCourse\pw5> .\recursive_descent.exe
Enter your input line: (zx | xc) & ~0b10101
Accepted.
PS D:\work_space\AutomataTheoryCourse\pw5> .\recursive_descent.exe
Enter your input line: a = ~b | (ab | ba) ; b = jj ; ~(x & 0b10)
Accepted.
PS D:\work_space\AutomataTheoryCourse\pw5> .\recursive_descent.exe
Enter your input line: bb = ~(a | 000)
Accepted.
PS D:\work_space\AutomataTheoryCourse\pw5> .\recursive_descent.exe
Enter your input line: a = ~( z | x
Rejected.
PS D:\work_space\AutomataTheoryCourse\pw5> .\recursive_descent.exe
Enter your input line: bbb = a
Rejected.
PS D:\work_space\AutomataTheoryCourse\pw5> .\recursive_descent.exe
Enter your input line: a = 0b023
Rejected.
PS D:\work_space\AutomataTheoryCourse\pw5> .\recursive_descent.exe
Enter your input line: zx = 0x123k
Rejected.
PS D:\work_space\AutomataTheoryCourse\pw5> .\recursive_descent.exe
Enter your input line: pl = $ | 12
Rejected.
```

Рисунок 23 – Тест программы на произвольных цепочках

Все тесты были пройдены программой успешно. Принимаются только логические выражения и есть возможность разделения логических выражения с помощью точки с запятой.

#### 4 Выводы

В ходе данной практической работы были исследованы свойства универсальных алгоритмов синтаксического анализа контекстно-свободных языков, построены LL(1) и SLR(1) грамматики, а также изучены шаги формального доказательства принадлежности грамматики к LL(1)-грамматике. Также мы узнали про синтаксический анализатор, работающий методом рекурсивного спуска.