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 _____ А.С.Кузнецов
подпись, дата — инициалы, фамилия

Студент КИ23-17/2Б, 052323070 _____ Я.С. Дябкин
номер группы, зачетной книжки — подпись, дата — инициалы, фамилия

Красноярск 2025

ХОД РАБОТЫ

Цель работы:

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

Задачи

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.

Описание варианта задания

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

Ход выполнения

Часть 1, создание LL(1)-грамматики

Для описания операций были выбраны следующие символы: ~ (логическое отрицание), | (дизъюнкция, «ИЛИ»), & (конъюнкция, «И»), а также = (присваивание). Были заданы скобки для явного выделения приоритета. Полученная LL(1)-грамматика реализована в JFLAP (см. рисунок 1).

LHS		
S	→	V=E
V	→	g
V	→	h
V	→	i
V	→	j
V	→	k
E	→	AR
A	→	MB
B	→	|MB
B	→	λ
M	→	UN
N	→	&UN
N	→	λ
U	→	~U
U	→	P
P	→	V
P	→	C
C	→	HD
D	→	HD
D	→	λ
H	→	0

LHS		
D	→	λ
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)
R	→	=E
R	→	λ

Рисунок 1 – Составленная LL(1)-грамматика

Затем был выполнен «Build LL(1) Parse» и получено множество первых порождаемых символов, символов последователей и таблица синтаксического анализа. Результаты предоставлены 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	{ λ, | }	{ $,), = }
C	{ a, b, c, d, e, f, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }	{ $, &,), |, = }
D	{ λ, 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	{ λ, & }	{ $,), |, = }
P	{ a, b, c, d, e, f, g, h, (, i, j, k, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }	{ $, &,), |, = }
R	{ λ, = }	{ $,) }
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 }	{ $, &,), |, = }

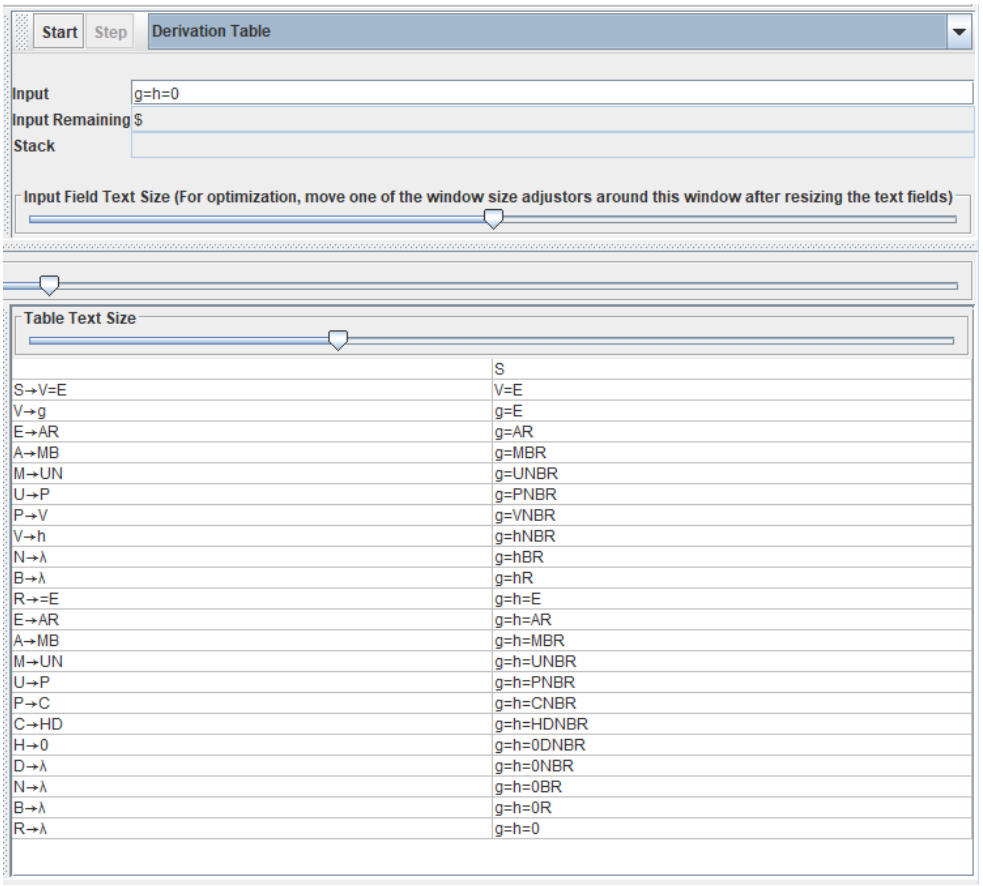
Рисунок 2 – Множество первых порождаемых символов и символов последователей составленной грамматики

	&	(	)	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			λ											λ												|MB		λ
C				HD	HD	HD	HD	HD	HD	HD	HD	HD	HD		HD	HD	HD	HD	HD	HD								
D	λ		λ	HD	HD	HD	HD	HD	HD	HD	HD	HD	HD	λ	HD	HD	HD	HD	HD	HD						λ		λ
E		AR		AR	AR	AR	AR	AR	AR	AR	AR	AR	AR		AR	AR	AR	AR	AR	AR	AR	AR	AR	AR	AR		AR	
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	
N	&UN		λ											λ											λ		λ
P		(E)		C	C	C	C	C	C	C	C	C	C		C	C	C	C	C	C	V	V	V	V	V			
R			λ											=E														λ
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

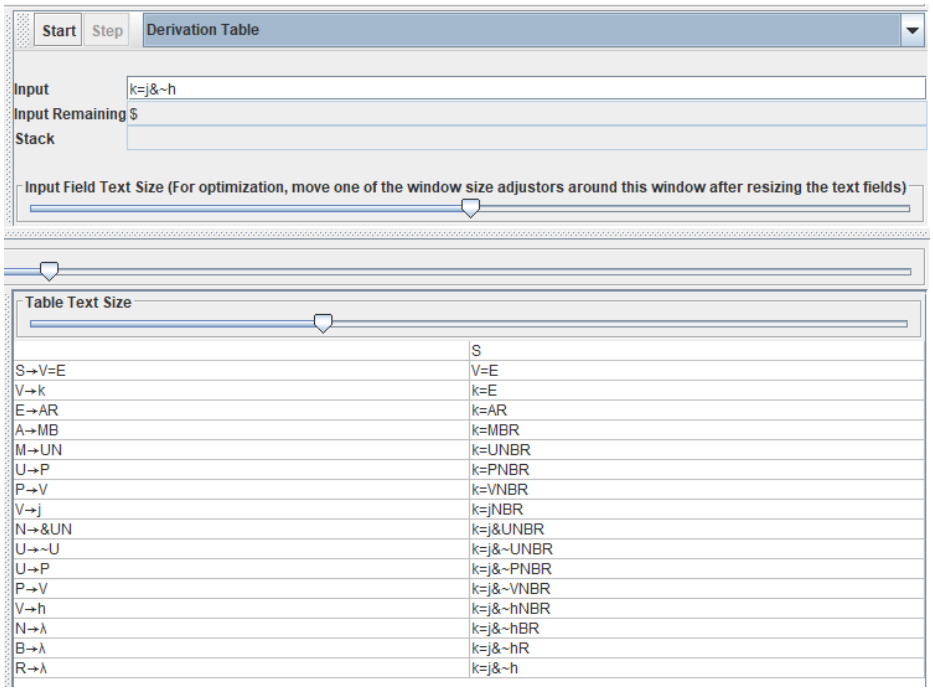
Как видно из таблицы, в каждой ячейке содержится не более одного правила — это подтверждает, что грамматика действительно принадлежит к классу LL(1).

Затем проводилось тестирование распознавания цепочек с помощью функции разбора («Parse») JFLAP — были подобраны 6 тестовых цепочек для проверки корректности работы анализатора. Итоги проверки отображены на рисунках 4–9.



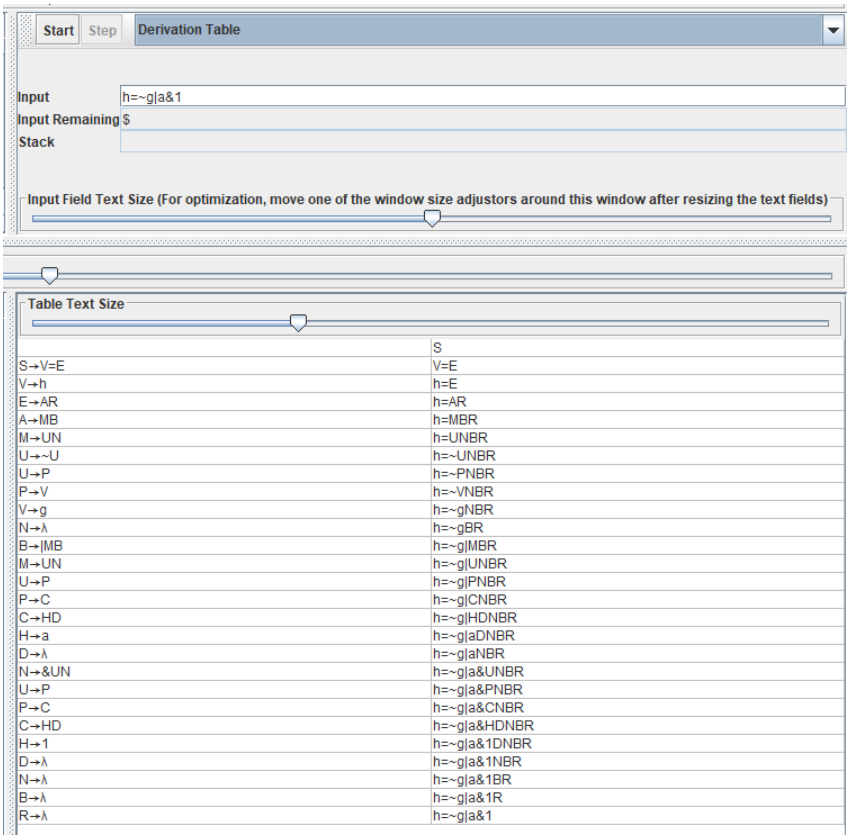
	S
S→V=E	V=E
V→g	g=E
E→AR	g=AR
A→MB	g=MBR
M→UN	g=UNBR
U→P	g=PNBR
P→V	g=VNBR
V→h	g=hNBR
N→λ	g=hBR
B→λ	g=hR
R→=E	g=h=E
E→AR	g=h=AR
A→MB	g=h=MBR
M→UN	g=h=UNBR
U→P	g=h=PNBR
P→C	g=h=CNBR
C→HD	g=h=HDNBR
H→0	g=h=0DNBR
D→λ	g=h=0NBR
N→λ	g=h=0BR
B→λ	g=h=0R
R→λ	g=h=0

Рисунок 4 - Тест для цепочки «g=h=0»



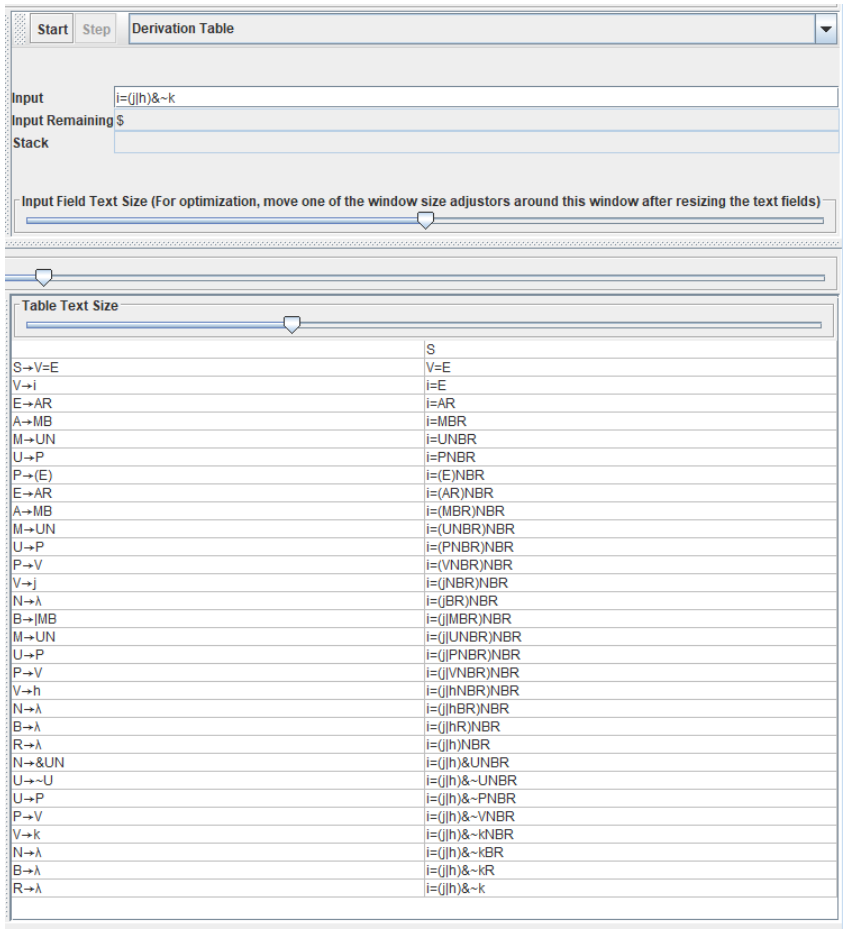
	S
S→V=E	V=E
V→k	k=E
E→AR	k=AR
A→MB	k=MBR
M→UN	k=UNBR
U→P	k=PNBR
P→V	k=VNBR
V→j	k=jNBR
N→&UN	k=j&UNBR
U→~U	k=j&~UNBR
U→P	k=j&~PNBR
P→V	k=j&~VNBR
V→h	k=j&~hNBR
N→λ	k=j&~hBR
B→λ	k=j&~hR
R→λ	k=j&~h

Рисунок 5 – Тест для цепочки «k=j&~h»



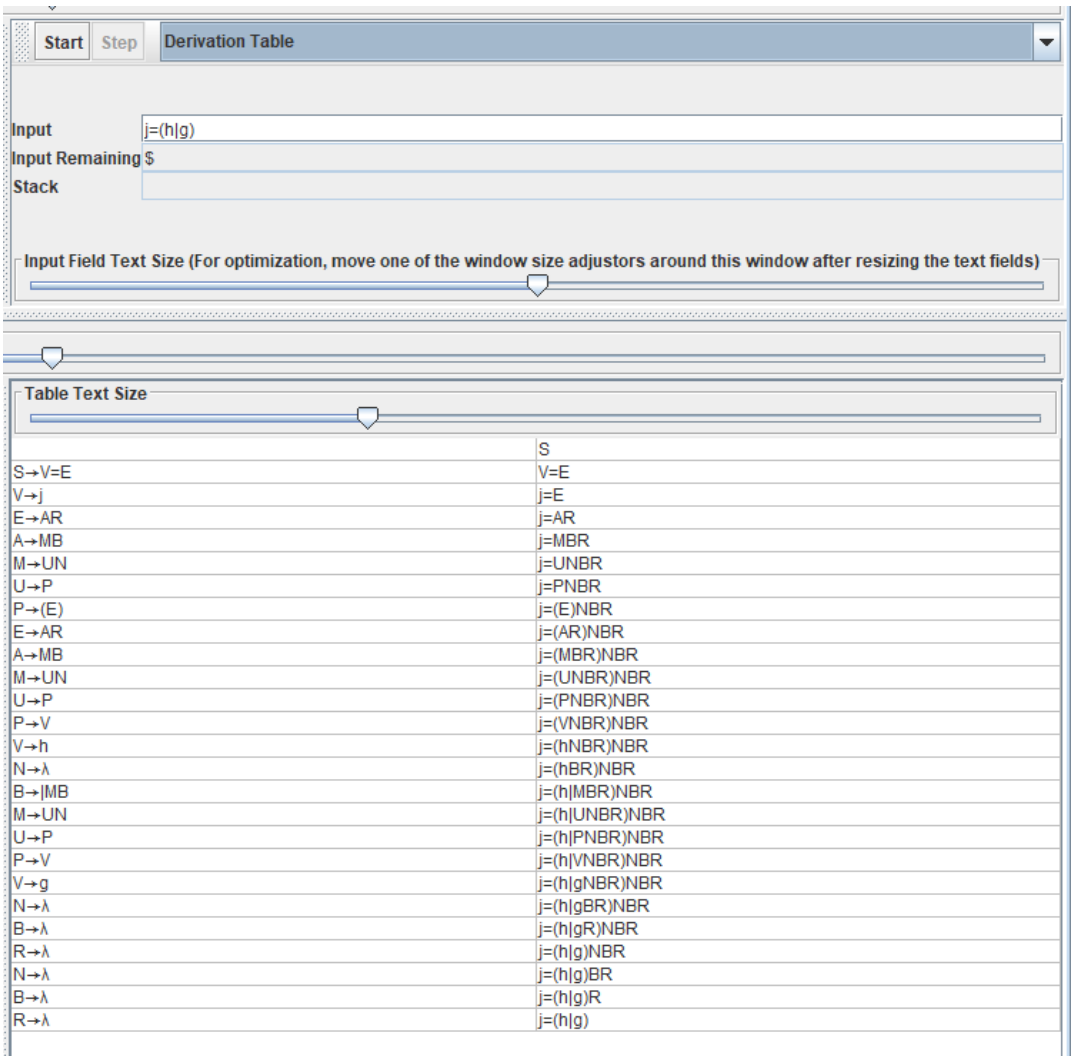
	S
S→V=E	V=E
V→h	h=E
E→AR	h=AR
A→MB	h=MBR
M→UN	h=UNBR
U→~U	h=~UNBR
U→P	h=~PNBR
P→V	h=~VNBR
V→g	h=~gNBR
N→λ	h=~gBR
B→|MB	h=~g|MBR
M→UN	h=~g|UNBR
U→P	h=~g|PNBR
P→C	h=~g|CNBR
C→HD	h=~g|HDNBR
H→a	h=~g|aDNBR
D→λ	h=~g|aNBR
N→&UN	h=~g|a&UNBR
U→P	h=~g|a&PNBR
P→C	h=~g|a&CNBR
C→HD	h=~g|a&HDNBR
H→1	h=~g|a&1DNBR
D→λ	h=~g|a&1NBR
N→λ	h=~g|a&1BR
B→λ	h=~g|a&1R
R→λ	h=~g|a&1

Рисунок 6 – Тест для цепочки «h=~g|a&1»



	S
S→V=E	V=E
V→i	i=E
E→AR	i=AR
A→MB	i=MBR
M→UN	i=UNBR
U→P	i=PNBR
P→(E)	i=(E)NBR
E→AR	i=(AR)NBR
A→MB	i=(MBR)NBR
M→UN	i=(UNBR)NBR
U→P	i=(PNBR)NBR
P→V	i=(VNBR)NBR
V→j	i=(jNBR)NBR
N→λ	i=(jBR)NBR
B→|MB	i=(j|MBR)NBR
M→UN	i=(j|UNBR)NBR
U→P	i=(j|PNBR)NBR
P→V	i=(j|VNBR)NBR
V→h	i=(j|hNBR)NBR
N→λ	i=(j|hBR)NBR
B→λ	i=(j|hR)NBR
R→λ	i=(j|h)NBR
N→&UN	i=(j|h)&UNBR
U→~U	i=(j|h)&~UNBR
U→P	i=(j|h)&~PNBR
P→V	i=(j|h)&~VNBR
V→k	i=(j|h)&~kNBR
N→λ	i=(j|h)&~kBR
B→λ	i=(j|h)&~kR
R→λ	i=(j|h)&~k

Рисунок 7 – Тест для цепочки «i=(j|h)&~k»



	S
S→V=E	V=E
V→j	j=E
E→AR	j=AR
A→MB	j=MBR
M→UN	j=UNBR
U→P	j=PNBR
P→(E)	j=(E)NBR
E→AR	j=(AR)NBR
A→MB	j=(MBR)NBR
M→UN	j=(UNBR)NBR
U→P	j=(PNBR)NBR
P→V	j=(VNBR)NBR
V→h	j=(hNBR)NBR
N→λ	j=(hBR)NBR
B→|MB	j=(h|MBR)NBR
M→UN	j=(h|UNBR)NBR
U→P	j=(h|PNBR)NBR
P→V	j=(h|VNBR)NBR
V→g	j=(h|gNBR)NBR
N→λ	j=(h|gBR)NBR
B→λ	j=(h|gR)NBR
R→λ	j=(h|g)NBR
N→λ	j=(h|g)BR
B→λ	j=(h|g)R
R→λ	j=(h|g)

Рисунок 8 – Тест для цепочки «j=(h|g)»

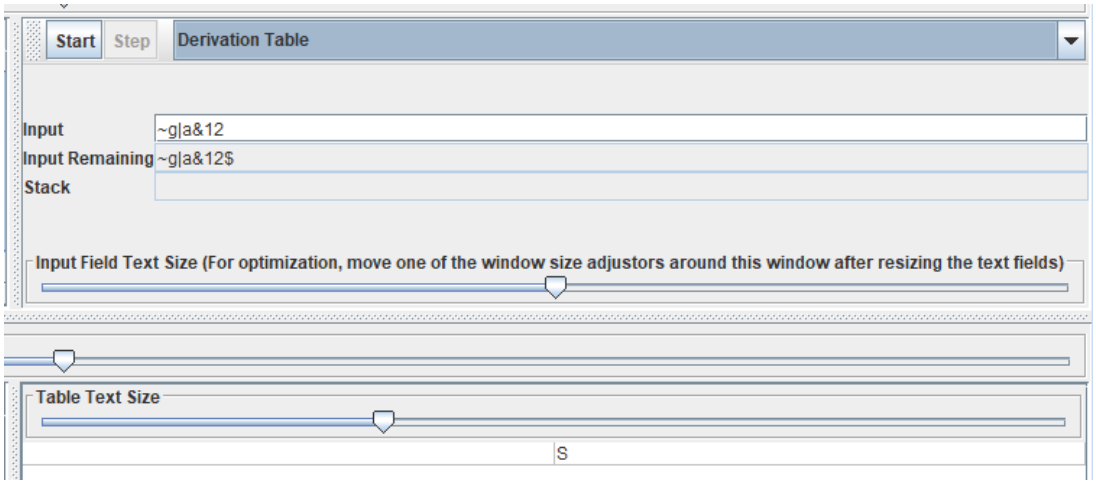


Рисунок 9 – Тест для цепочки «~g|l&12»

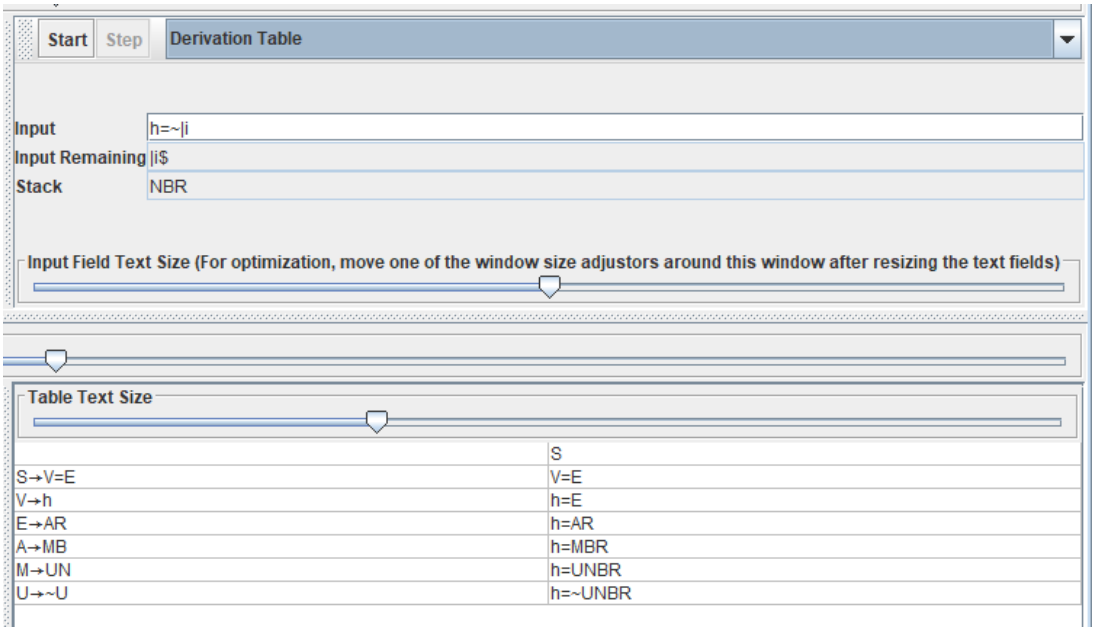


Рисунок 10 – Тест для цепочки «h=~|i»

Все тесты были успешно пройдены, и полученная грамматика правильно определяет исходящий язык.

Часть 2, создание SLR(1)-грамматики.

На данном этапе была построена SLR(1)-грамматика для языка логических выражений с операциями ~, &, |, = и разделителем ;. Полученная грамматика была введена в JFLAP, её вид приведён на рисунке 11.

LHS		
Z	→	R
R	→	T
R	→	R;T
T	→	I=X
X	→	Y
X	→	I=X
Y	→	K
Y	→	Y|K
K	→	N
K	→	K&N
N	→	F
N	→	~N
F	→	I
F	→	C
F	→	(Y)
I	→	g
I	→	h
I	→	i
I	→	j
I	→	k
C	→	0Q
C	→	0xJ
Q	→	O
Q	→	QO
O	→	0
O	→	1

LHS		
O	→	1
O	→	2
O	→	3
O	→	4
O	→	5
O	→	6
O	→	7
J	→	H
J	→	JH
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

Рисунок 11 – Составленная SLR(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 12, 13 и 14.

	FIRST	FOLLOW
C	{ 0 }	{ $, &,), ;, | }
F	{ 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, ;, | }
K	{ 0, g, (, h, i, j, k, ~ }	{ $, &,), ;, | }
N	{ 0, g, (, h, i, j, k, ~ }	{ $, &,), ;, | }
O	{ 0, 1, 2, 3, 4, 5, 6, 7 }	{ $, &,), 0, 1, 2, 3, 4, 5, 6, 7, ;, | }
Q	{ 0, 1, 2, 3, 4, 5, 6, 7 }	{ $, &,), 0, 1, 2, 3, 4, 5, 6, 7, ;, | }
R	{ g, h, i, j, k }	{ $, ; }
T	{ g, h, i, j, k }	{ $, ; }
X	{ 0, g, h, (, i, j, k, ~ }	{ $, ; }
Y	{ 0, g, (, h, i, j, k, ~ }	{ $,), ;, | }
Z	{ g, h, i, j, k }	{ $ }

Рисунок 12 – Множество первых порождаемых символов и символов последователей составленной грамматики

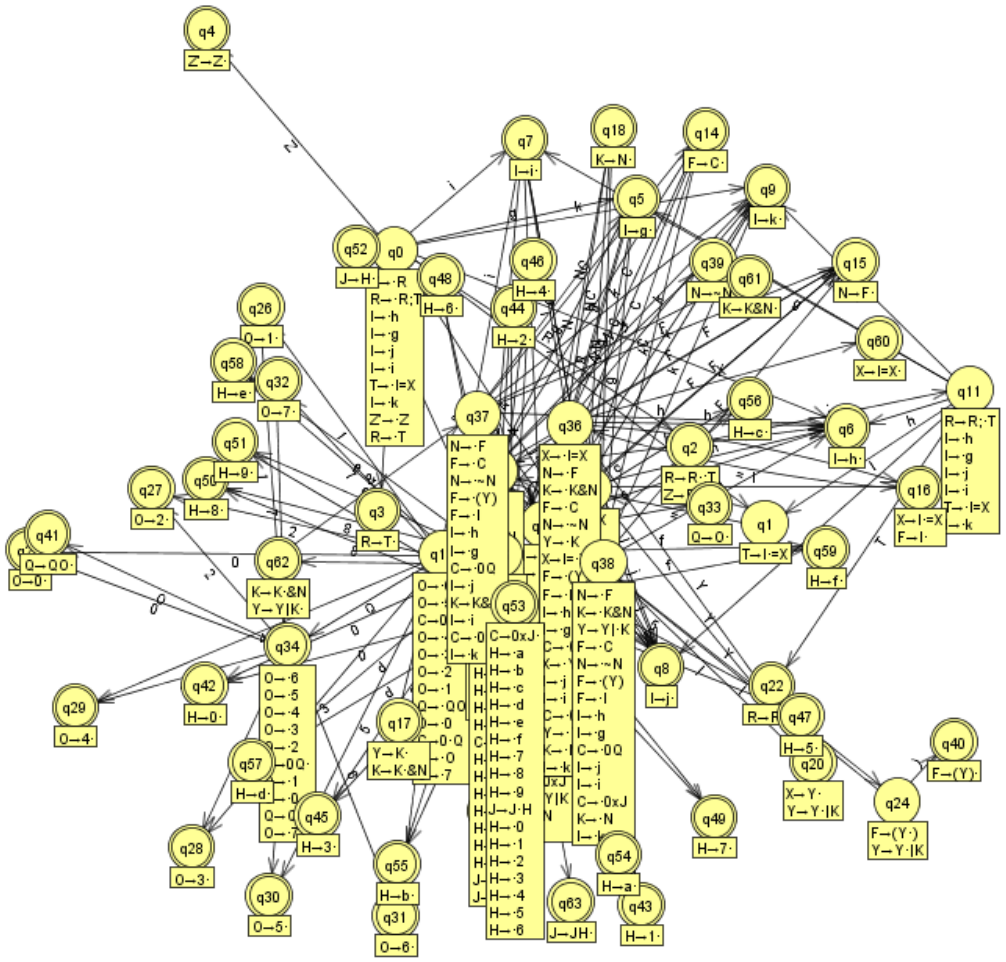
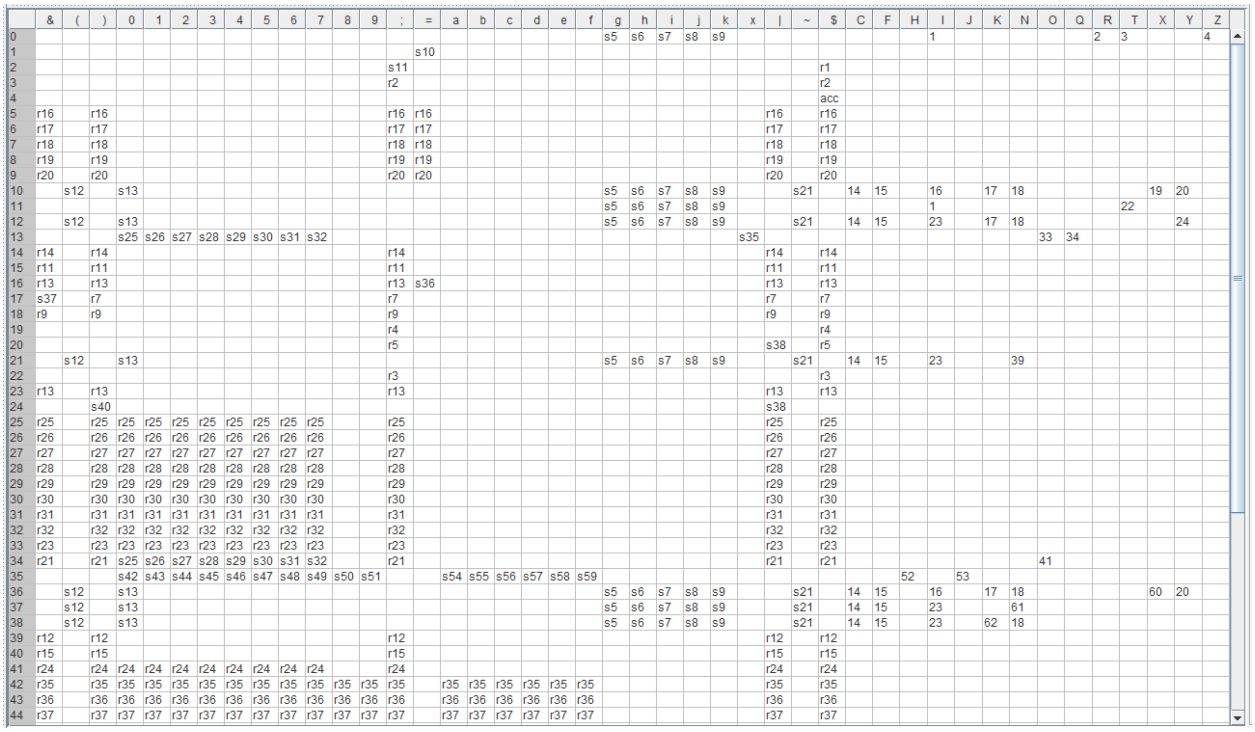


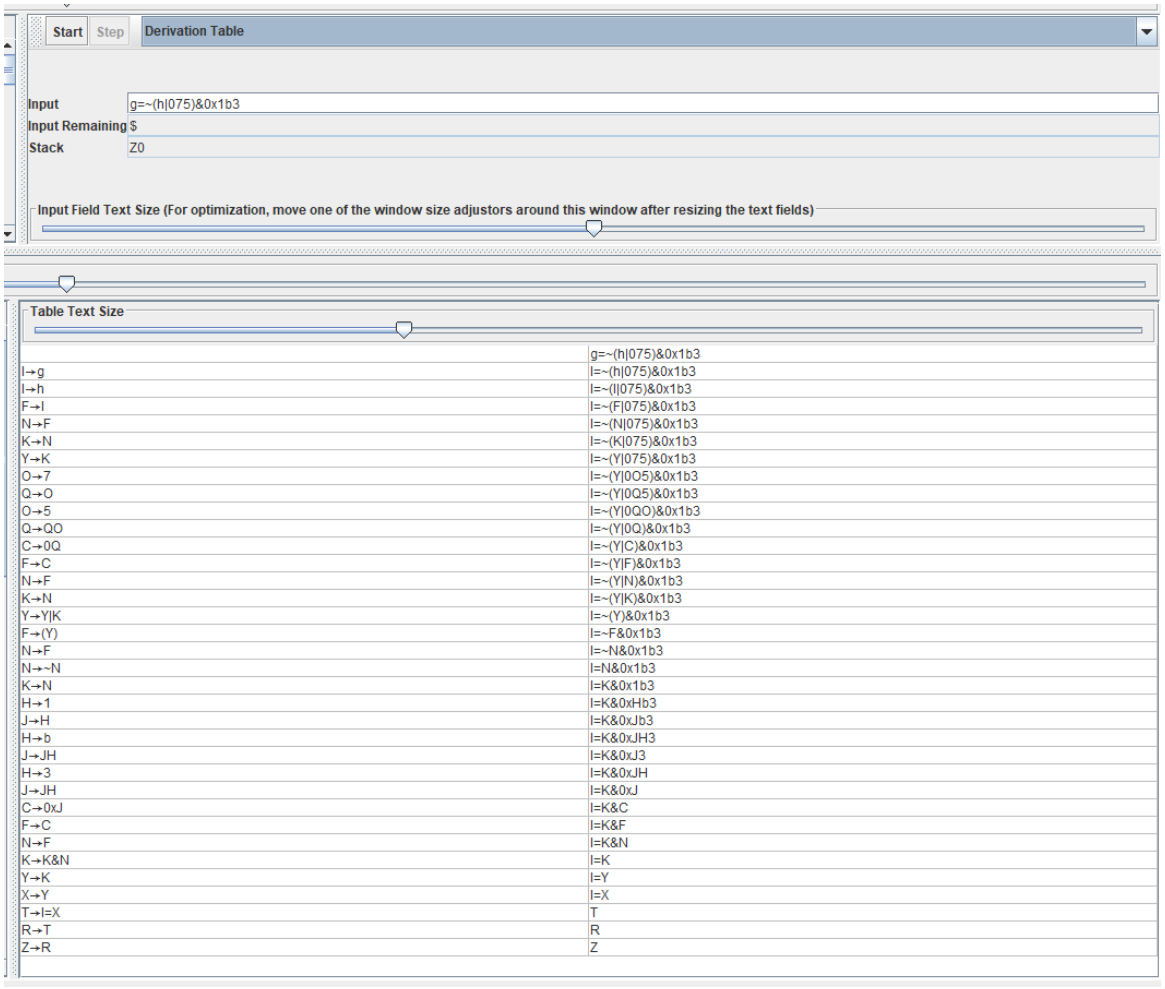
Рисунок 13 – Канонической набор LR(0)-ситуаций



	&	(	)	0	1	2	3	4	5	6	7	8	9	;	=	a	b	c	d	e	f	g	h	i	j	k	x	|	~	$	C	F	H	I	J	K	N	O	Q	R	T	X	Y	Z
0																						s5	s6	s7	s8	s9								1						2	3			4
1															s10																														
2														s11																r1															
3														r2																r2															
4																														acc															
5	r16		r16											r16	r16													r16		r16															
6	r17		r17											r17	r17													r17		r17															
7	r18		r18											r18	r18													r18		r18															
8	r19		r19											r19	r19													r19		r19															
9	r20		r20											r20	r20													r20		r20															
10		s12		s13																		s5	s6	s7	s8	s9			s21		14	15		16		17	18					19	20	
11																						s5	s6	s7	s8	s9								1							22			
12		s12		s13																		s5	s6	s7	s8	s9			s21		14	15		23		17	18						24		
13				s25	s26	s27	s28	s29	s30	s31	s32																s35												33	34					
14	r14		r14											r14														r14		r14															
15	r11		r11											r11														r11		r11															
16	r13		r13											r13	s36													r13		r13															
17	s37		r7											r7														r7		r7															
18	r9		r9											r9														r9		r9															
19														r4																r4															
20														r5														s38		r5															
21		s12		s13																		s5	s6	s7	s8	s9			s21		14	15		23			39							
22														r3																r3															
23	r13		r13											r13														r13		r13															
24			s40																									s38																	
25	r25		r25	r25	r25	r25	r25	r25	r25	r25	r25			r25														r25		r25															
26	r26		r26	r26	r26	r26	r26	r26	r26	r26	r26			r26														r26		r26															
27	r27		r27	r27	r27	r27	r27	r27	r27	r27	r27			r27														r27		r27															
28	r28		r28	r28	r28	r28	r28	r28	r28	r28	r28			r28														r28		r28															
29	r29		r29	r29	r29	r29	r29	r29	r29	r29	r29			r29														r29		r29															
30	r30		r30	r30	r30	r30	r30	r30	r30	r30	r30			r30														r30		r30															
31	r31		r31	r31	r31	r31	r31	r31	r31	r31	r31			r31														r31		r31															
32	r32		r32	r32	r32	r32	r32	r32	r32	r32	r32			r32														r32		r32															
33	r23		r23	r23	r23	r23	r23	r23	r23	r23	r23			r23														r23		r23															
34	r21		r21	s25	s26	s27	s28	s29	s30	s31	s32			r21														r21		r21									41						
35				s42	s43	s44	s45	s46	s47	s48	s49	s50	s51			s54	s55	s56	s57	s58	s59												52		53										
36		s12		s13																		s5	s6	s7	s8	s9			s21		14	15		16		17	18					60	20	
37		s12		s13																		s5	s6	s7	s8	s9			s21		14	15		23			61							
38		s12		s13																		s5	s6	s7	s8	s9			s21		14	15		23		62	18							
39	r12		r12											r12														r12		r12															
40	r15		r15											r15														r15		r15															
41	r24		r24	r24	r24	r24	r24	r24	r24	r24	r24			r24														r24		r24															
42	r35		r35	r35	r35	r35	r35	r35	r35	r35	r35	r35	r35	r35		r35	r35	r35	r35	r35	r35							r35		r35															
43	r36		r36	r36	r36	r36	r36	r36	r36	r36	r36	r36	r36	r36		r36	r36	r36	r36	r36	r36							r36		r36															
44	r37		r37	r37	r37	r37	r37	r37	r37	r37	r37	r37	r37	r37		r37	r37	r37	r37	r37	r37							r37		r37															

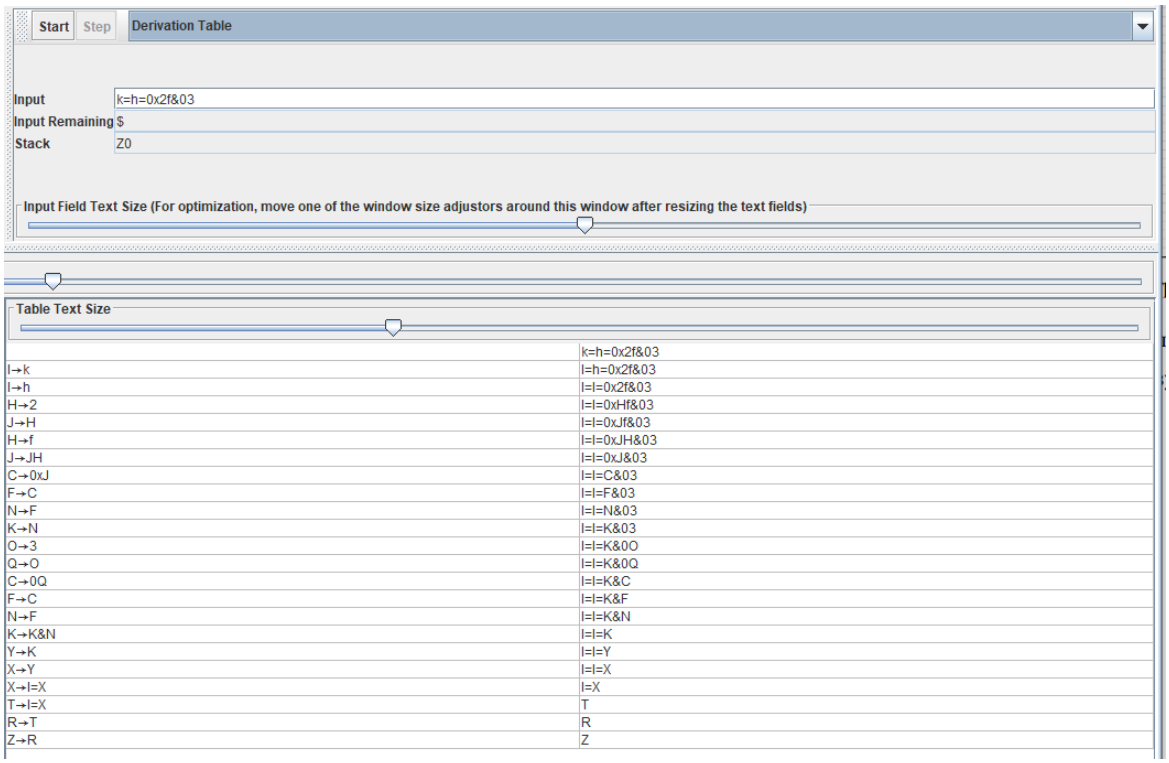
Рисунок 14 – Таблица синтаксического анализа составленной грамматики

После построения таблицы разбора были проведены эксперименты по распознаванию строк языка: путём нажатия на кнопку «Parse» и ввода тестовых цепочек для проверки. Ход и результаты разбора для каждой из них показаны на рисунках 15–20.



	g=~(h|075)&0x1b3
I→g	I=~(h|075)&0x1b3
I→h	I=~(I|075)&0x1b3
F→I	I=~(F|075)&0x1b3
N→F	I=~(N|075)&0x1b3
K→N	I=~(K|075)&0x1b3
Y→K	I=~(Y|075)&0x1b3
O→7	I=~(Y|0O5)&0x1b3
Q→O	I=~(Y|0Q5)&0x1b3
O→5	I=~(Y|0QO)&0x1b3
Q→QO	I=~(Y|0Q)&0x1b3
C→0Q	I=~(Y|C)&0x1b3
F→C	I=~(Y|F)&0x1b3
N→F	I=~(Y|N)&0x1b3
K→N	I=~(Y|K)&0x1b3
Y→Y|K	I=~(Y)&0x1b3
F→(Y)	I=~F&0x1b3
N→F	I=~N&0x1b3
N→~N	I=N&0x1b3
K→N	I=K&0x1b3
H→1	I=K&0xHb3
J→H	I=K&0xJb3
H→b	I=K&0xJH3
J→JH	I=K&0xJ3
H→3	I=K&0xJH
J→JH	I=K&0xJ
C→0xJ	I=K&C
F→C	I=K&F
N→F	I=K&N
K→K&N	I=K
Y→K	I=Y
X→Y	I=X
T→I=X	T
R→T	R
Z→R	Z

Рисунок 15 – Тест для цепочки «g=~(h|075)&0x1b3»



	k=h=0x2f&03
I→k	I=h=0x2f&03
I→h	I=I=0x2f&03
H→2	I=I=0xHf&03
J→H	I=I=0xJf&03
H→f	I=I=0xJH&03
J→JH	I=I=0xJ&03
C→0xJ	I=I=C&03
F→C	I=I=F&03
N→F	I=I=N&03
K→N	I=I=K&03
O→3	I=I=K&0O
Q→O	I=I=K&0Q
C→0Q	I=I=K&C
F→C	I=I=K&F
N→F	I=I=K&N
K→K&N	I=I=K
Y→K	I=I=Y
X→Y	I=I=X
X→I=X	I=X
T→I=X	T
R→T	R
Z→R	Z

Рисунок 16 – Тест для цепочки «k=h=0x2f&03»

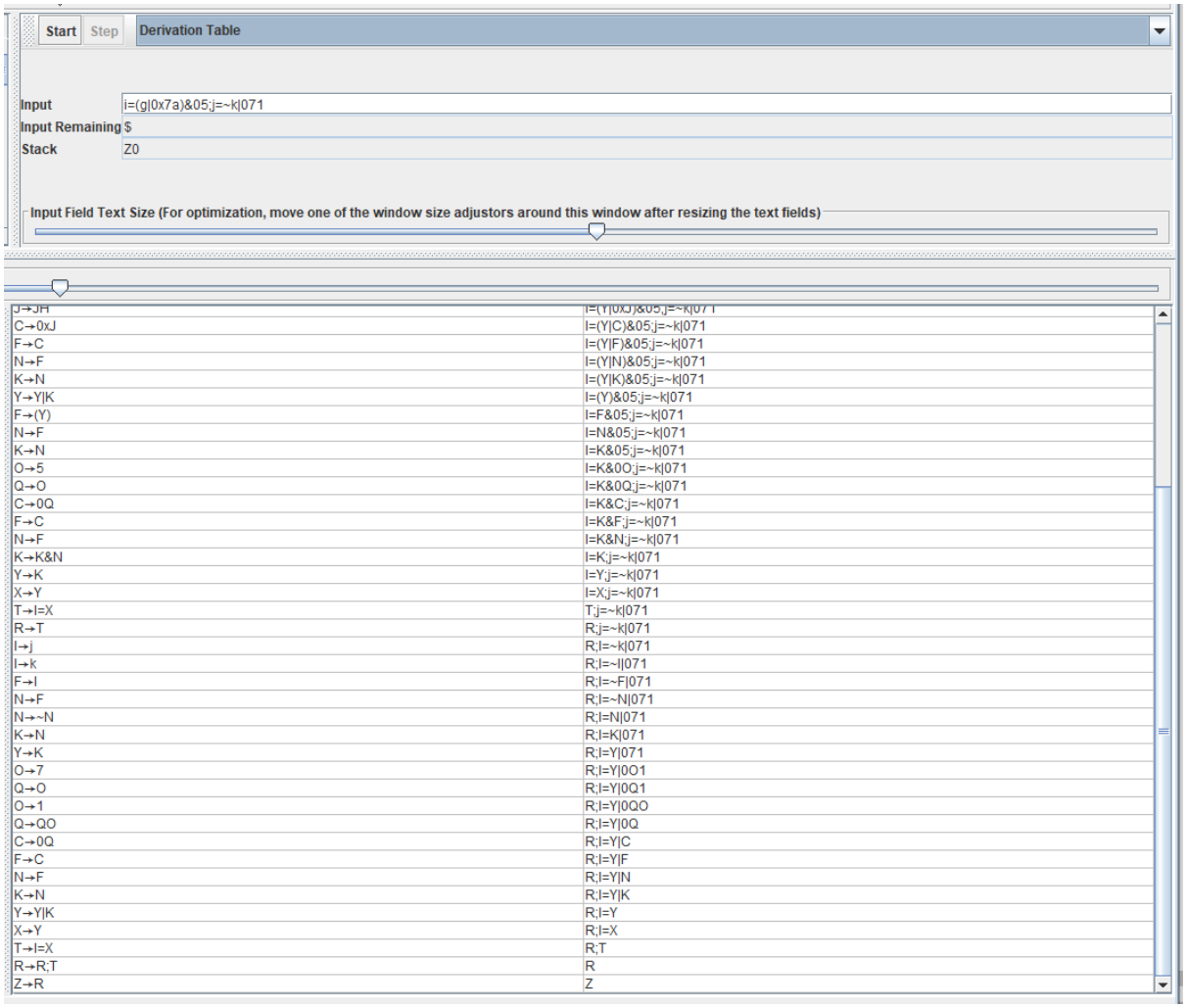
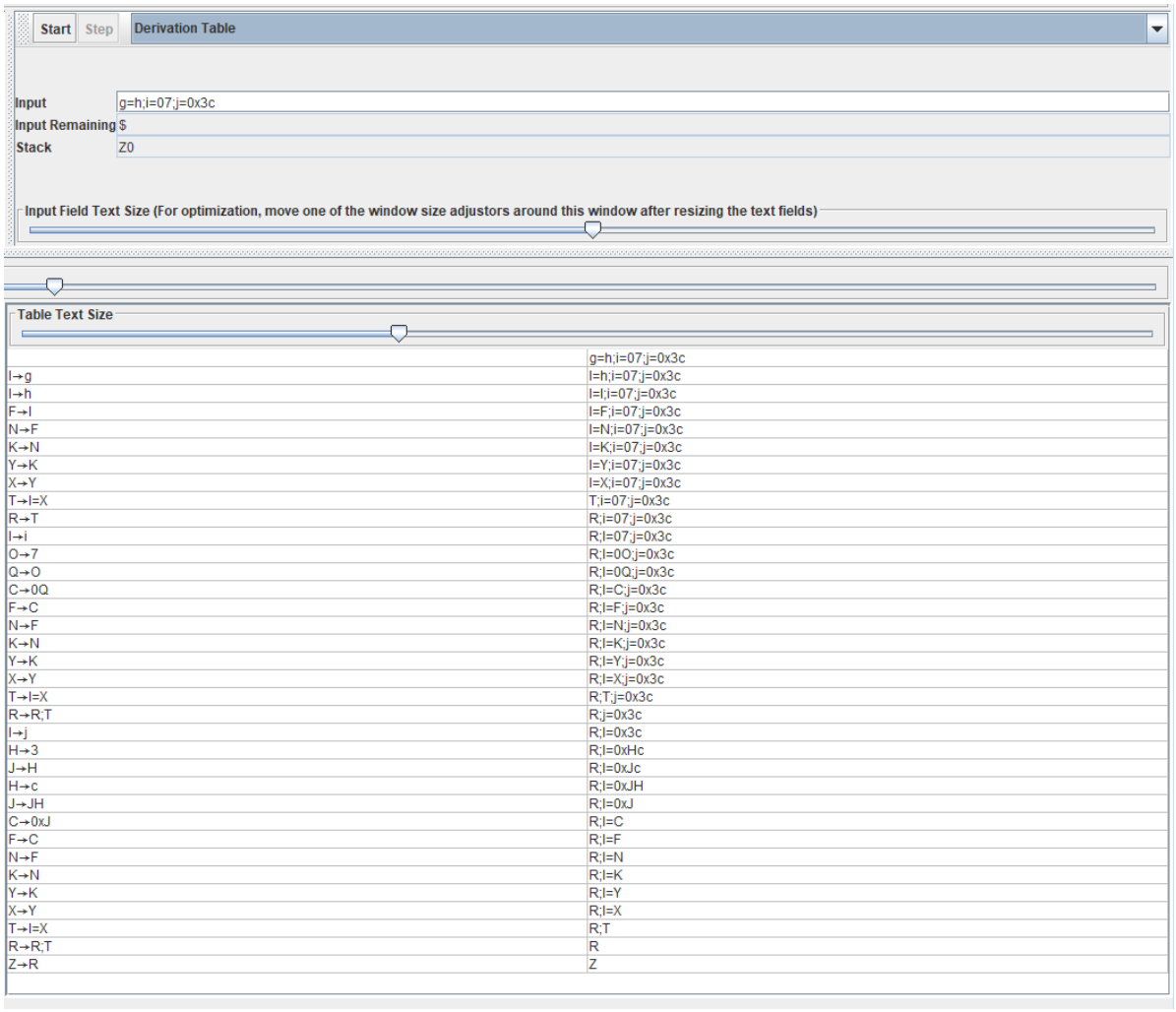
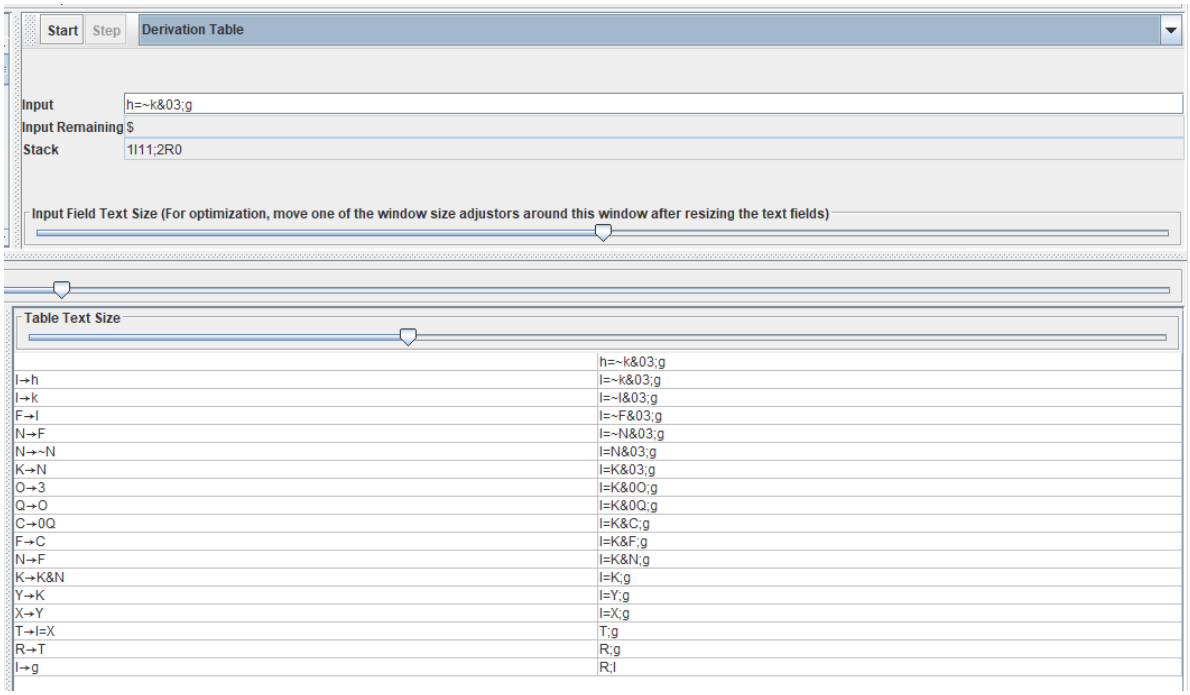


Рисунок 17 – Тест для цепочки «i=(g|0x7a)&05;j=~k|071»



	g=h;i=07;j=0x3c
I→g	I=h;i=07;j=0x3c
I→h	I=I;i=07;j=0x3c
F→I	I=F;i=07;j=0x3c
N→F	I=N;i=07;j=0x3c
K→N	I=K;i=07;j=0x3c
Y→K	I=Y;i=07;j=0x3c
X→Y	I=X;i=07;j=0x3c
T→I=X	T;i=07;j=0x3c
R→T	R;I=07;j=0x3c
I→i	R;I=07;j=0x3c
O→7	R;I=0O;j=0x3c
Q→O	R;I=0Q;j=0x3c
C→0Q	R;I=C;j=0x3c
F→C	R;I=F;j=0x3c
N→F	R;I=N;j=0x3c
K→N	R;I=K;j=0x3c
Y→K	R;I=Y;j=0x3c
X→Y	R;I=X;j=0x3c
T→I=X	R;T;j=0x3c
R→R;T	R;j=0x3c
I→j	R;I=0x3c
H→3	R;I=0xHc
J→H	R;I=0xJc
H→c	R;I=0xJH
J→JH	R;I=0xJ
C→0xJ	R;I=C
F→C	R;I=F
N→F	R;I=N
K→N	R;I=K
Y→K	R;I=Y
X→Y	R;I=X
T→I=X	R;T
R→R;T	R
Z→R	Z

Рисунок 18 – Тест для цепочки «g=h;i=07;j=0x3c»



	h=~k&03;g
I→h	I=~k&03;g
I→k	I=~I&03;g
F→I	I=~F&03;g
N→F	I=~N&03;g
N→~N	I=N&03;g
K→N	I=K&03;g
O→3	I=K&0O;g
Q→O	I=K&0Q;g
C→0Q	I=K&C;g
F→C	I=K&F;g
N→F	I=K&N;g
K→K&N	I=K;g
Y→K	I=Y;g
X→Y	I=X;g
T→I=X	T;g
R→T	R;g
I→g	R;I

Рисунок 19 – Тест для цепочки «h=~k&03;g»

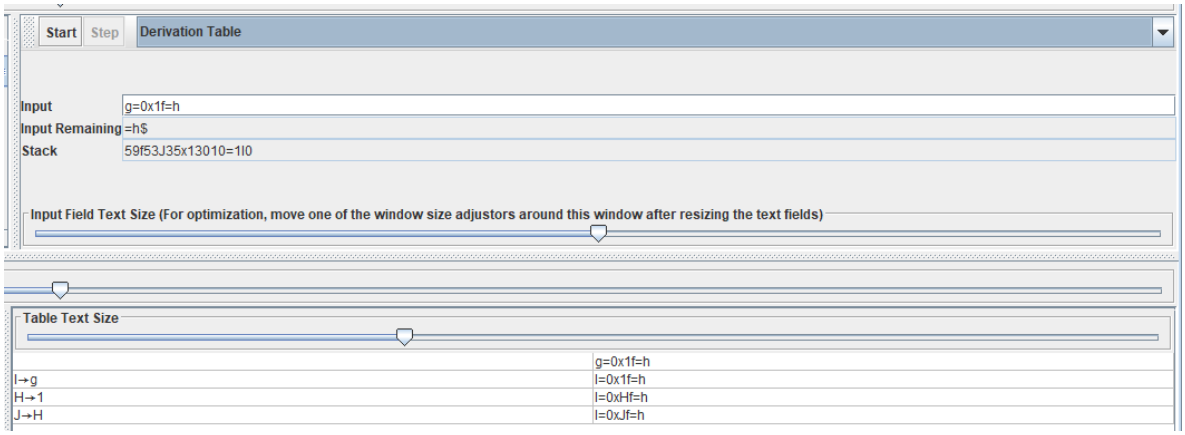


Рисунок 20 - Тест для цепочки «g=0x1f=h»

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

Часть 3, создание программной реализации синтаксического анализатора методом рекурсивного спуска.

На начальном этапе была разработана LL(1)-грамматика для рассматриваемого языка логических выражений с операциями ~, &, |, = и разделителем ;. По данной грамматике реализован синтаксический анализатор, схема грамматики показана на рисунке 11.

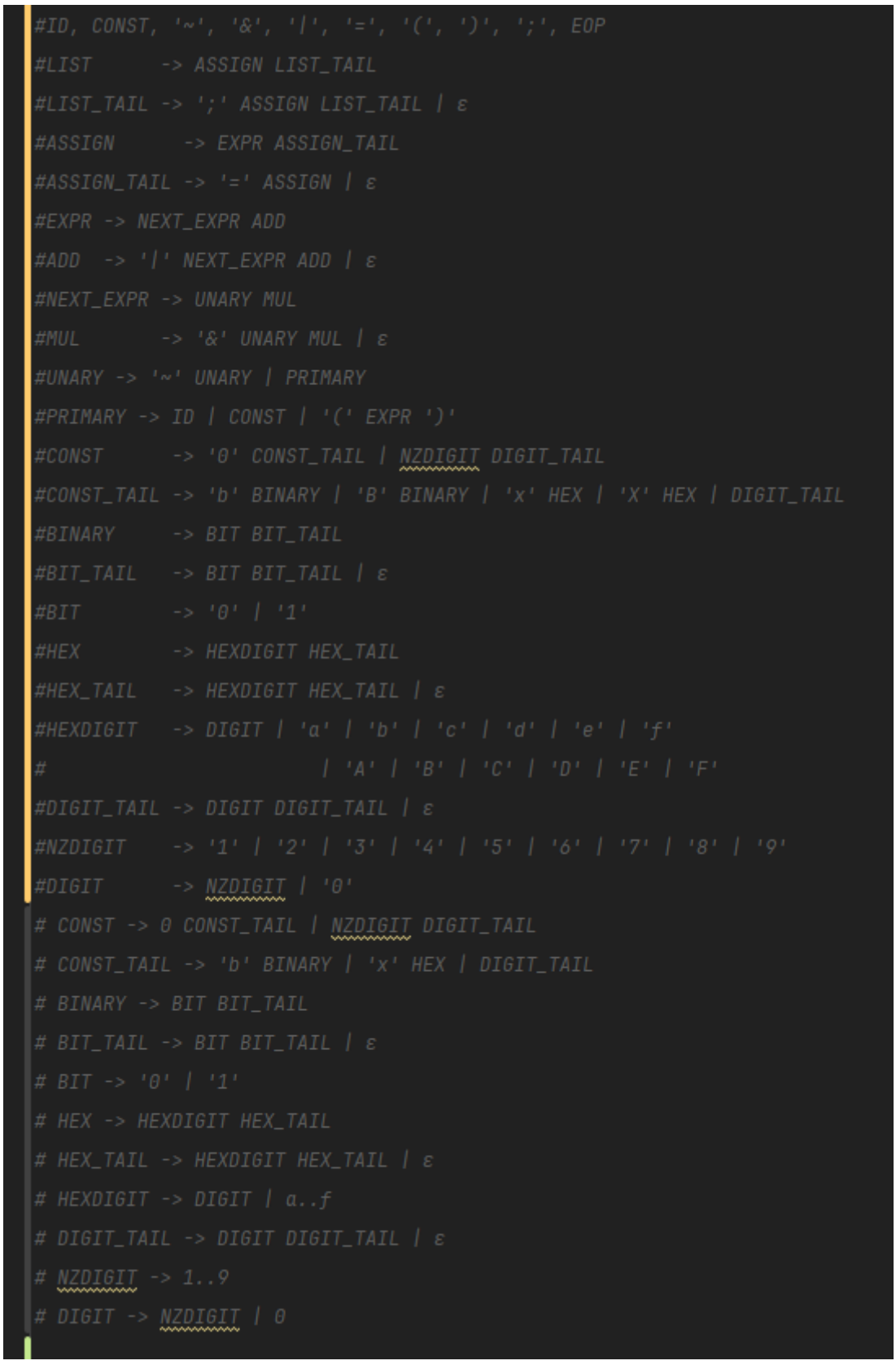


Рисунок 21 – Составленная LL(1)-грамматика

Далее проведён формальный анализ грамматики в JFLAP с целью подтверждения её принадлежности к классу LL(1). Во-первых, данная грамматика не имеет левой рекурсии и неоднозначности. Во-вторых,

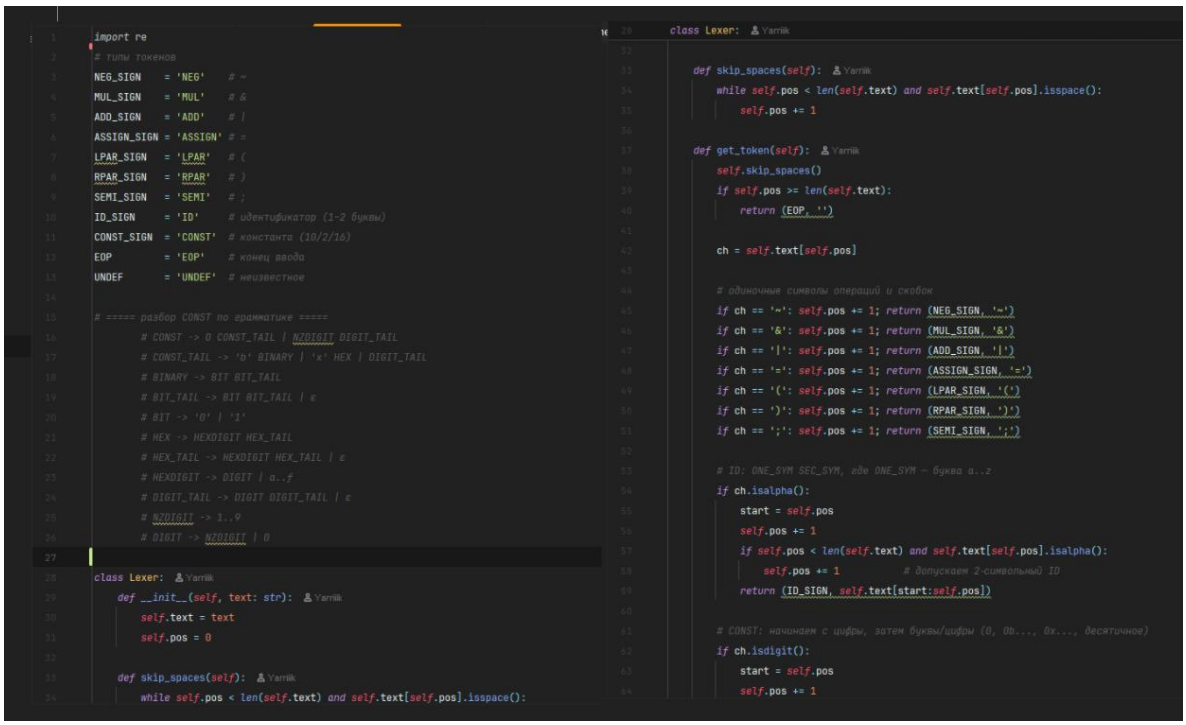
вычисленные множества FIRST для каждого нетерминала не пересекаются. В-третьих, вычисленные множества FOLLOW для каждого нетерминала выполняли равенство: «Если для продукции A → α есть ε ∈ FIRST(α), то FOLLOW(A) ∩ FIRST(β) = ∅ для всех других продукций A → β». В-четверых, в таблице синтаксического анализа каждая ячейка содержит не больше одной продукции, как показано на рисунке22.

	FIRST	FOLLOW
A	{ λ, | }	{ $,), ;, = }
C	{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }	{ $, &,), ;, |, = }
D	{ 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, ;, |, = }
E	{ g, (, h, i, j, k, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ~ }	{ $,), ;, = }
F	{ 0, 1 }	{ 0, 1, $, &,), ;, |, = }
G	{ 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, ;, |, = }
H	{ a, b, c, d, e, f, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }	{ $, &,), ;, |, = }
J	{ λ, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }	{ $, &,), ;, |, = }
K	{ 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	{ λ, ; }	{ $ }
M	{ g, (, h, i, j, k, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ~ }	{ $,), ;, |, = }
N	{ λ, & }	{ $,), ;, |, = }
P	{ g, (, h, i, j, k, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }	{ $, &,), ;, |, = }
Q	{ λ, a, b, c, d, e, f, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }	{ $, &,), ;, |, = }
R	{ λ, = }	{ $, ; }
S	{ g, (, h, i, j, k, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ~ }	{ $, ; }
T	{ λ, b, 0, 1, 2, 3, 4, 5, 6, 7, x, 8, 9 }	{ $, &,), ;, |, = }
U	{ g, (, h, i, j, k, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ~ }	{ $, &,), ;, |, = }
V	{ g, h, i, j, k }	{ $, &,), ;, |, = }
X	{ 0, 1 }	{ $, &,), ;, |, = }
Y	{ λ, 0, 1 }	{ $, &,), ;, |, = }
Z	{ g, (, h, i, j, k, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ~ }	{ $ }

	&	(	)	0	1	2	3	4	5	6	7	8	9	;	=	a	b	c	d	e	f	g	h	i	j	k	x	|	~	$
A			λ											λ	λ													|MA		λ
C				0T	KJ	KJ	KJ	KJ	KJ	KJ	KJ	KJ	KJ																	
D				0	K	K	K	K	K	K	K	K	K																	
E		MA		MA	MA	MA	MA	MA	MA	MA	MA	MA	MA									MA	MA	MA	MA	MA			MA	
F				0	1																									
G				D	D	D	D	D	D	D	D	D	D			a	b	c	d	e	f									
H				GQ	GQ	GQ	GQ	GQ	GQ	GQ	GQ	GQ	GQ			GQ	GQ	GQ	GQ	GQ	GQ									
J	λ		λ	DJ	DJ	DJ	DJ	DJ	DJ	DJ	DJ	DJ	DJ	λ	λ													λ		λ
K					1	2	3	4	5	6	7	8	9																	
L														;SL																λ
M		UN		UN	UN	UN	UN	UN	UN	UN	UN	UN	UN									UN	UN	UN	UN	UN			UN	
N	&...		λ											λ	λ													λ		λ
P		(E)		C	C	C	C	C	C	C	C	C	C									V	V	V	V	V				
Q	λ		λ	GQ	GQ	GQ	GQ	GQ	GQ	GQ	GQ	GQ	GQ	λ	λ	GQ	GQ	GQ	GQ	GQ	GQ							λ		λ
R														λ	=S															λ
S		ER		ER	ER	ER	ER	ER	ER	ER	ER	ER	ER									ER	ER	ER	ER	ER			ER	
T	J		J	J	J	J	J	J	J	J	J	J	J	J	J		bX										xH	J		J
U		P		P	P	P	P	P	P	P	P	P	P									P	P	P	P	P			~U	
V																						g	h	i	j	k				
X				FY	FY																									
Y	λ		λ	FY	FY									λ	λ													λ		λ
Z		SL		SL	SL	SL	SL	SL	SL	SL	SL	SL	SL									SL	SL	SL	SL	SL			SL	

Рисунок 22 - Таблица синтаксического анализа и множество первых порождаемых символов и символов последователей составленной грамматики

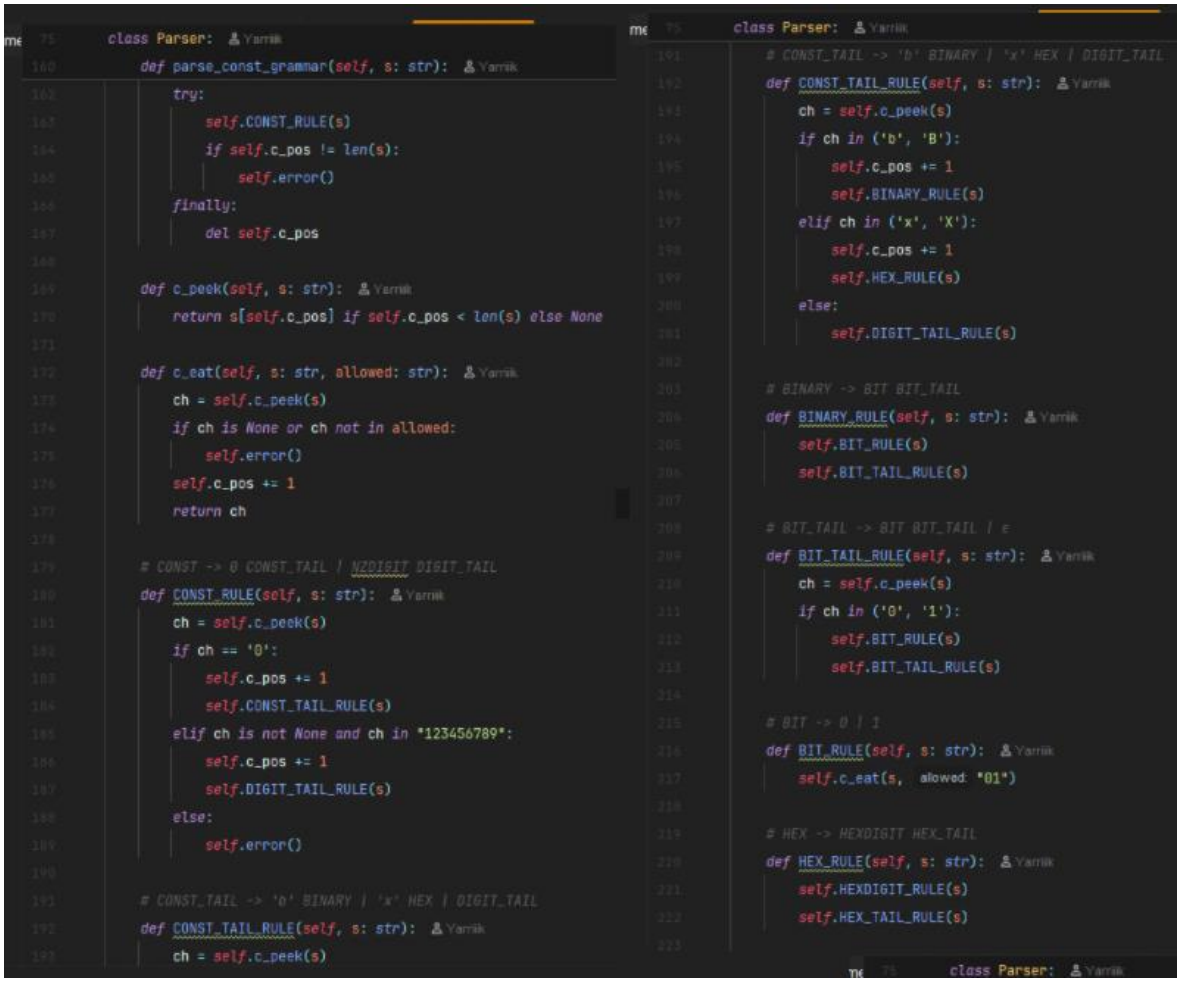
Была реализована программа на python, основанная на рекурсивном спуске, реализующая синтаксический анализ с использованием составленной грамматики. Код программы продемонстрирован на рисунках 23-25.



```python
import re
# типы токенов
NEG_SIGN    = 'NEG'    # ~
MUL_SIGN    = 'MUL'    # &
ADD_SIGN    = 'ADD'    # |
ASSIGN_SIGN = 'ASSIGN' # =
LPAR_SIGN   = 'LPAR'   # (
RPAR_SIGN   = 'RPAR'   # )
SEMI_SIGN   = 'SEMI'   # ;
ID_SIGN     = 'ID'     # идентификатор (1-2 буквы)
CONST_SIGN  = 'CONST'  # константа (10/2/16)
EOP         = 'EOP'    # конец ввода
UNDEF       = 'UNDEF'  # неизвестное

# ===== разбор CONST по грамматике =====
        # CONST -> 0 CONST_TAIL | NZDIGIT DIGIT_TAIL
        # CONST_TAIL -> 'b' BINARY | 'x' HEX | DIGIT_TAIL
        # BINARY -> BIT BIT_TAIL
        # BIT_TAIL -> BIT BIT_TAIL | ε
        # BIT -> '0' | '1'
        # HEX -> HEXDIGIT HEX_TAIL
        # HEX_TAIL -> HEXDIGIT HEX_TAIL | ε
        # HEXDIGIT -> DIGIT | a..f
        # DIGIT_TAIL -> DIGIT DIGIT_TAIL | ε
        # NZDIGIT -> 1..9
        # DIGIT -> NZDIGIT | 0

class Lexer:
    def __init__(self, text: str):
        self.text = text
        self.pos = 0

    def skip_spaces(self):
        while self.pos < len(self.text) and self.text[self.pos].isspace():
```

```python
class Lexer:

    def skip_spaces(self):
        while self.pos < len(self.text) and self.text[self.pos].isspace():
            self.pos += 1

    def get_token(self):
        self.skip_spaces()
        if self.pos >= len(self.text):
            return (EOP, '')

        ch = self.text[self.pos]

        # одиночные символы операций и скобок
        if ch == '~': self.pos += 1; return (NEG_SIGN, '~')
        if ch == '&': self.pos += 1; return (MUL_SIGN, '&')
        if ch == '|': self.pos += 1; return (ADD_SIGN, '|')
        if ch == '=': self.pos += 1; return (ASSIGN_SIGN, '=')
        if ch == '(': self.pos += 1; return (LPAR_SIGN, '(')
        if ch == ')': self.pos += 1; return (RPAR_SIGN, ')')
        if ch == ';': self.pos += 1; return (SEMI_SIGN, ';')

        # ID: ONE_SYM SEC_SYM, где ONE_SYM – буква a..z
        if ch.isalpha():
            start = self.pos
            self.pos += 1
            if self.pos < len(self.text) and self.text[self.pos].isalpha():
                self.pos += 1          # допускаем 2-символьный ID
            return (ID_SIGN, self.text[start:self.pos])

        # CONST: начинаем с цифры, затем буквы/цифры (0, 0b..., 0x..., десятичное)
        if ch.isdigit():
            start = self.pos
            self.pos += 1
```

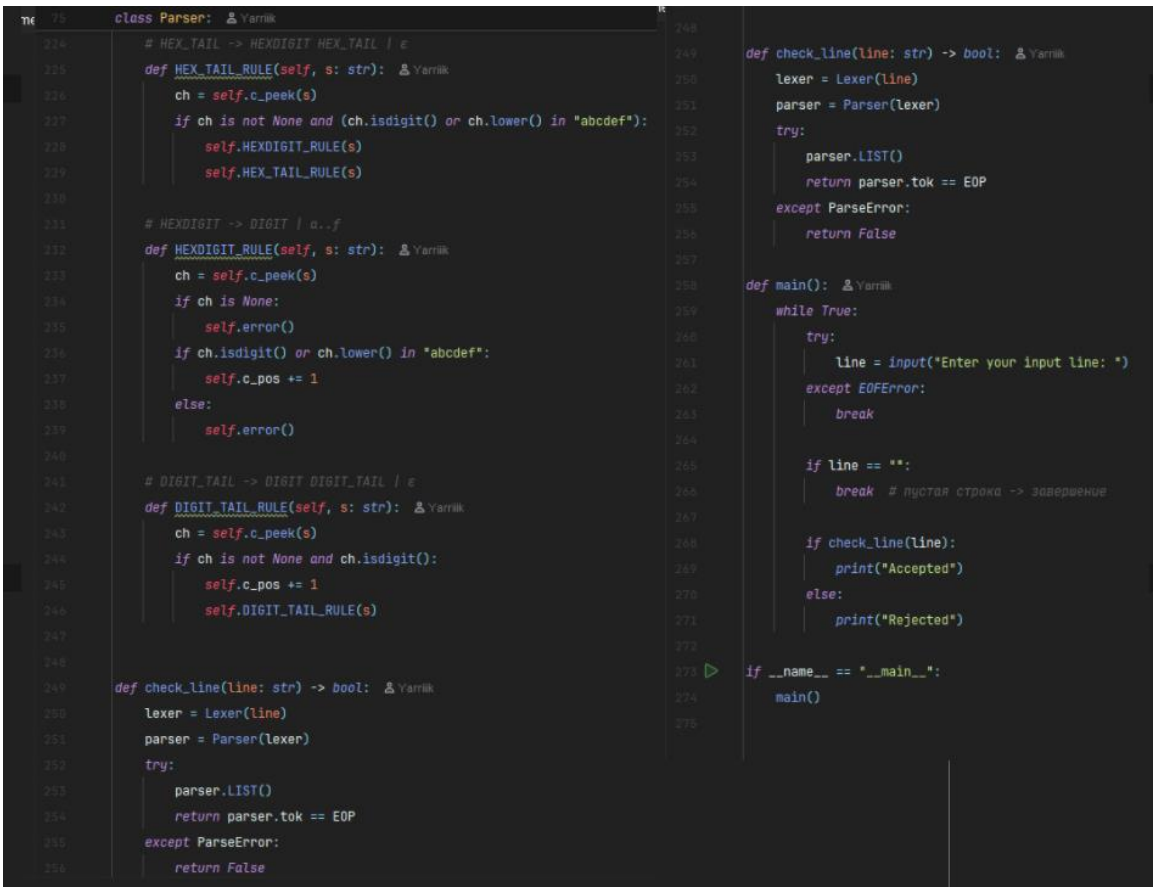
Рисунок 23 – Первая часть кода реализованной программы на python



```python
class Parser:
    def parse_const_grammar(self, s: str):
        try:
            self.CONST_RULE(s)
            if self.c_pos != len(s):
                self.error()
        finally:
            del self.c_pos

    def c_peek(self, s: str):
        return s[self.c_pos] if self.c_pos < len(s) else None

    def c_eat(self, s: str, allowed: str):
        ch = self.c_peek(s)
        if ch is None or ch not in allowed:
            self.error()
        self.c_pos += 1
        return ch

    # CONST -> 0 CONST_TAIL | NZDIGIT DIGIT_TAIL
    def CONST_RULE(self, s: str):
        ch = self.c_peek(s)
        if ch == '0':
            self.c_pos += 1
            self.CONST_TAIL_RULE(s)
        elif ch is not None and ch in "123456789":
            self.c_pos += 1
            self.DIGIT_TAIL_RULE(s)
        else:
            self.error()

    # CONST_TAIL -> 'b' BINARY | 'x' HEX | DIGIT_TAIL
    def CONST_TAIL_RULE(self, s: str):
        ch = self.c_peek(s)
```

```python
class Parser:
    # CONST_TAIL -> 'b' BINARY | 'x' HEX | DIGIT_TAIL
    def CONST_TAIL_RULE(self, s: str):
        ch = self.c_peek(s)
        if ch in ('b', 'B'):
            self.c_pos += 1
            self.BINARY_RULE(s)
        elif ch in ('x', 'X'):
            self.c_pos += 1
            self.HEX_RULE(s)
        else:
            self.DIGIT_TAIL_RULE(s)

    # BINARY -> BIT BIT_TAIL
    def BINARY_RULE(self, s: str):
        self.BIT_RULE(s)
        self.BIT_TAIL_RULE(s)

    # BIT_TAIL -> BIT BIT_TAIL | ε
    def BIT_TAIL_RULE(self, s: str):
        ch = self.c_peek(s)
        if ch in ('0', '1'):
            self.BIT_RULE(s)
            self.BIT_TAIL_RULE(s)

    # BIT -> 0 | 1
    def BIT_RULE(self, s: str):
        self.c_eat(s, allowed: "01")

    # HEX -> HEXDIGIT HEX_TAIL
    def HEX_RULE(self, s: str):
        self.HEXDIGIT_RULE(s)
        self.HEX_TAIL_RULE(s)
```

Рисунок 24 – Вторая часть кода реализованной программы на python



```
class Parser:  Yarrilk
    # HEX_TAIL -> HEXDIGIT HEX_TAIL | ε
    def HEX_TAIL_RULE(self, s: str):  Yarrilk
        ch = self.c_peek(s)
        if ch is not None and (ch.isdigit() or ch.lower() in "abcdef"):
            self.HEXDIGIT_RULE(s)
            self.HEX_TAIL_RULE(s)

    # HEXDIGIT -> DIGIT | a..f
    def HEXDIGIT_RULE(self, s: str):  Yarrilk
        ch = self.c_peek(s)
        if ch is None:
            self.error()
        if ch.isdigit() or ch.lower() in "abcdef":
            self.c_pos += 1
        else:
            self.error()

    # DIGIT_TAIL -> DIGIT DIGIT_TAIL | ε
    def DIGIT_TAIL_RULE(self, s: str):  Yarrilk
        ch = self.c_peek(s)
        if ch is not None and ch.isdigit():
            self.c_pos += 1
            self.DIGIT_TAIL_RULE(s)


def check_line(line: str) -> bool:  Yarrilk
    lexer = Lexer(line)
    parser = Parser(lexer)
    try:
        parser.LIST()
        return parser.tok == EOP
    except ParseError:
        return False
```

```
def check_line(line: str) -> bool:  Yarrilk
    lexer = Lexer(line)
    parser = Parser(lexer)
    try:
        parser.LIST()
        return parser.tok == EOP
    except ParseError:
        return False

def main():  Yarrilk
    while True:
        try:
            line = input("Enter your input line: ")
        except EOFError:
            break

        if line == "":
            break  # пустая строка -> завершение

        if check_line(line):
            print("Accepted")
        else:
            print("Rejected")

if __name__ == "__main__":
    main()
```

Рисунок 25 - Третья часть кода реализованной программы на python

На рисунке 26 показаны тестовые примеры работы программы на тестовых цепочках.

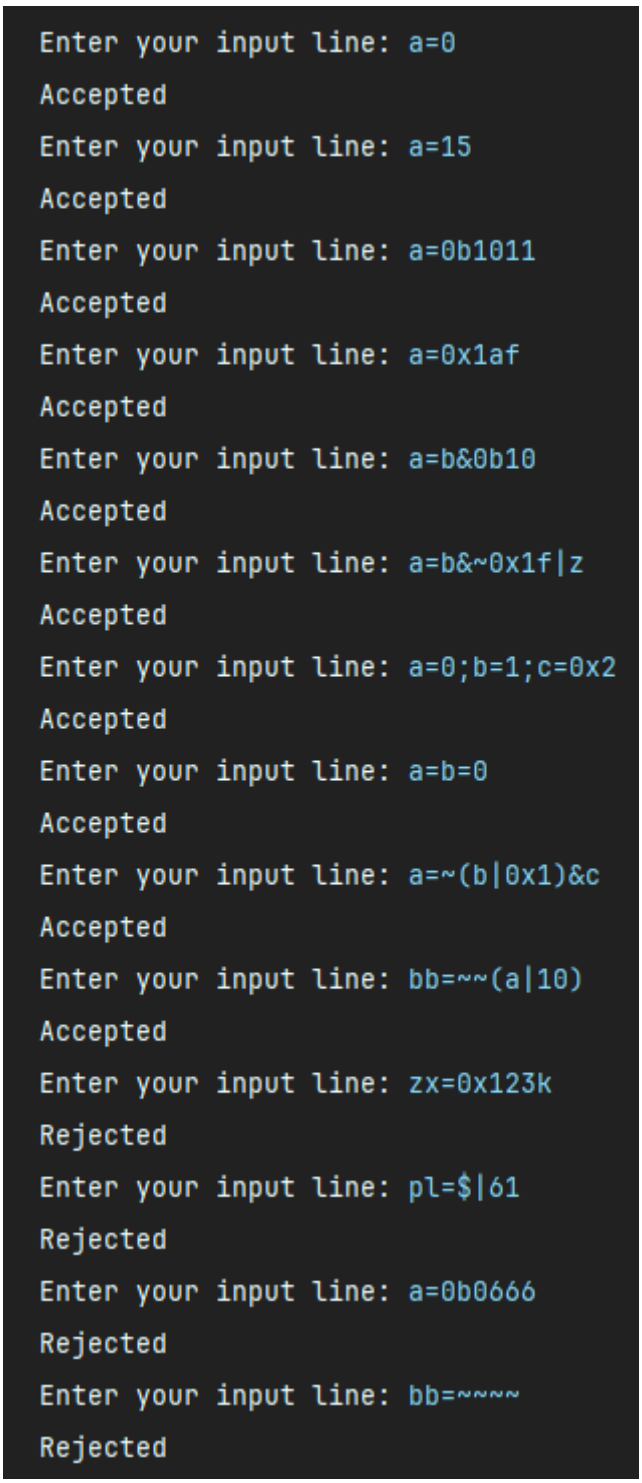


Рисунок 26 – Тест программы на произвольных цепочках

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

Выводы

В ходе данной практической работы были рассмотрены основные методы синтаксического анализа контекстно-свободных языков и их применение к языку логических выражений с операциями присваивания, отрицания, конъюнкции и дизъюнкции. Для заданного языка были построены и исследованы LL(1)- и SLR(1)-грамматики, для которых с использованием множеств FIRST/FOLLOW и таблиц разбора подтверждена корректность и однозначность синтаксического анализа. На основе LL(1)-грамматики был реализован рекурсивно-спускающийся синтаксический анализатор, а для SLR(1)-грамматики — восходящий анализ в среде JFLAP, причём тестирование на наборе корректных и некорректных цепочек показало согласованность реализованных алгоритмов с формальными спецификациями грамматик.