

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

инициалы, фамилия

Студент КИ23-17/2Б, 052323070

Я.С. Дябкин

номер группы, зачетной книжки

подпись, дата

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

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	MB	MB	MB		
B	λ													λ											MB	λ		
C		HD																										
D	λ	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	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	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	C	V	V	V	V	V					
R		λ												=E												λ		
S		P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	V=E				
U																									V=E			
V																									~U			

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.

Derivation Table	
Input	g=h=0
Input Remaining \$	
Stack	
Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the text fields)	
Table Text Size	
S→V=E	S
V→g	V=E
E→AR	g=E
A→MB	g=AR
M→UN	g=MBR
U→P	g=UNBR
P→V	g=PNBR
V→h	g=VNBR
N→λ	g=hNBR
B→λ	g=hBR
R→=E	g=h=R
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»

Derivation Table	
Input	k=j&~h
Input Remaining \$	
Stack	
Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the text fields)	
Table Text Size	
S→V=E	S
V→k	V=E
E→AR	k=E
A→MB	k=AR
M→UN	k=MBR
U→P	k=UNBR
P→V	k=PNBR
V→j	k=VNBR
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»

Derivation Table	
Input	h=¬g a&1
Input Remaining	\$
Stack	
Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the text fields)	
Table Text Size	
S→V=E	S
V→h	V=E
E→AR	h=E
A→MB	h=AR
M→UN	h=MBR
U→U	h=UNBR
U→P	h=¬UNBR
P→V	h=PNBR
V→g	h=¬PNBR
N→λ	h=VNBR
B→IMB	h=¬VNBR
M→UN	h=gBR
U→P	h=¬gBR
P→C	h=g PNBR
C→HD	h=g CNBR
H→a	h=g HDNBR
D→λ	h=g aDnbr
N→&UN	h=g aNBR
U→P	h=g a&UNBR
P→C	h=g a&PNBR
C→HD	h=g a&CNBR
H→1	h=g a&HDNBR
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»

Derivation Table	
Input	i=(j h)&¬k
Input Remaining	\$
Stack	
Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the text fields)	
Table Text Size	
S→V=E	S
V→i	V=E
E→AR	i=E
A→MB	i=AR
M→UN	i=MBR
U→P	i=UNBR
P→(E)	i=PNBR
E→AR	i=(E)NBR
A→MB	i=(AR)NBR
M→UN	i=(MBR)NBR
U→P	i=(UNBR)NBR
P→V	i=(PNBR)NBR
V→j	i=(VNBR)NBR
N→λ	i=(JNBR)NBR
B→IMB	i=(JBR)NBR
M→UN	i=(JMBR)NBR
U→P	i=(JUNBR)NBR
P→V	i=(JPNBR)NBR
V→h	i=(JVNBR)NBR
N→λ	i=(JhNBR)NBR
B→λ	i=(JhR)NBR
R→λ	i=(Jh)NBR
N→&UN	i=(Jh)&UNBR
U→U	i=(Jh)&¬UNBR
U→P	i=(Jh)&PNBR
P→V	i=(Jh)&VNBR
V→k	i=(Jh)&¬VNBR
N→λ	i=(Jh)&¬kBR
B→λ	i=(Jh)&¬kR
R→λ	i=(Jh)&¬k

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

Derivation Table	
Input	j=(h g)
Input Remaining	\$
Stack	
Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the text fields)	
↓	
↓	
Table Text Size	
↓	
S→V=E	S
V→j	V=E
E→AR	j=E
A→MB	j=AR
M→UN	j=MBR
U→P	j=UNBR
P→(E)	j=PNBR
E→AR	j=(E)NBR
A→MB	j=(AR)NBR
M→UN	j=(MBR)NBR
U→P	j=(UNBR)NBR
P→V	j=(PNBR)NBR
V→h	j=(VNBR)NBR
N→λ	j=(hNBR)NBR
B→IMB	j=(hBR)NBR
M→UN	j=(hIMBR)NBR
U→P	j=(hUNBR)NBR
P→V	j=(hPNBR)NBR
V→g	j=(hVNBR)NBR
N→λ	j=(hgNBR)NBR
B→λ	j=(hgBR)NBR
R→λ	j=(hg)NBR
N→λ	j=(hg)BR
B→λ	j=(hg)R
R→λ	j=(hg)

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

Derivation Table	
Input	~g a&12
Input Remaining	~g a&12\$
Stack	
Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the text fields)	
↓	
↓	
Table Text Size	
↓	
	S

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

Start	Step	Derivation Table
Input	h=~ i	
Input Remaining	i\$	
Stack	NBR	
Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the text fields)		
Table Text Size		
S→V=E	V=E	S
V→h	h=E	
E→AR	h=AR	
A→MB	h=MBR	
M→UN	h=UNBR	
U→~U	h=~UNBR	

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  $\sim$ ,  $\&$ ,  $|$ ,  $=$  и разделителем ;. Полученная грамматика была введена в JFLAP, её вид приведён на рисунке 11.

LHS	
Z	$\rightarrow R$
R	$\rightarrow T$
R	$\rightarrow R;T$
T	$\rightarrow I=X$
X	$\rightarrow Y$
X	$\rightarrow I=X$
Y	$\rightarrow K$
Y	$\rightarrow Y K$
K	$\rightarrow N$
K	$\rightarrow K&N$
N	$\rightarrow F$
N	$\rightarrow \sim N$
F	$\rightarrow I$
F	$\rightarrow C$
F	$\rightarrow (Y)$
I	$\rightarrow g$
I	$\rightarrow h$
I	$\rightarrow i$
I	$\rightarrow j$
I	$\rightarrow k$
C	$\rightarrow 0Q$
C	$\rightarrow 0xJ$
Q	$\rightarrow O$
Q	$\rightarrow QO$
O	$\rightarrow 0$
O	$\rightarrow 1$
LHS	
O	$\rightarrow 1$
O	$\rightarrow 2$
O	$\rightarrow 3$
O	$\rightarrow 4$
O	$\rightarrow 5$
O	$\rightarrow 6$
O	$\rightarrow 7$
J	$\rightarrow H$
J	$\rightarrow JH$
H	$\rightarrow 0$
H	$\rightarrow 1$
H	$\rightarrow 2$
H	$\rightarrow 3$
H	$\rightarrow 4$
H	$\rightarrow 5$
H	$\rightarrow 6$
H	$\rightarrow 7$
H	$\rightarrow 8$
H	$\rightarrow 9$
H	$\rightarrow a$
H	$\rightarrow b$
H	$\rightarrow c$
H	$\rightarrow d$
H	$\rightarrow e$
H	$\rightarrow f$

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

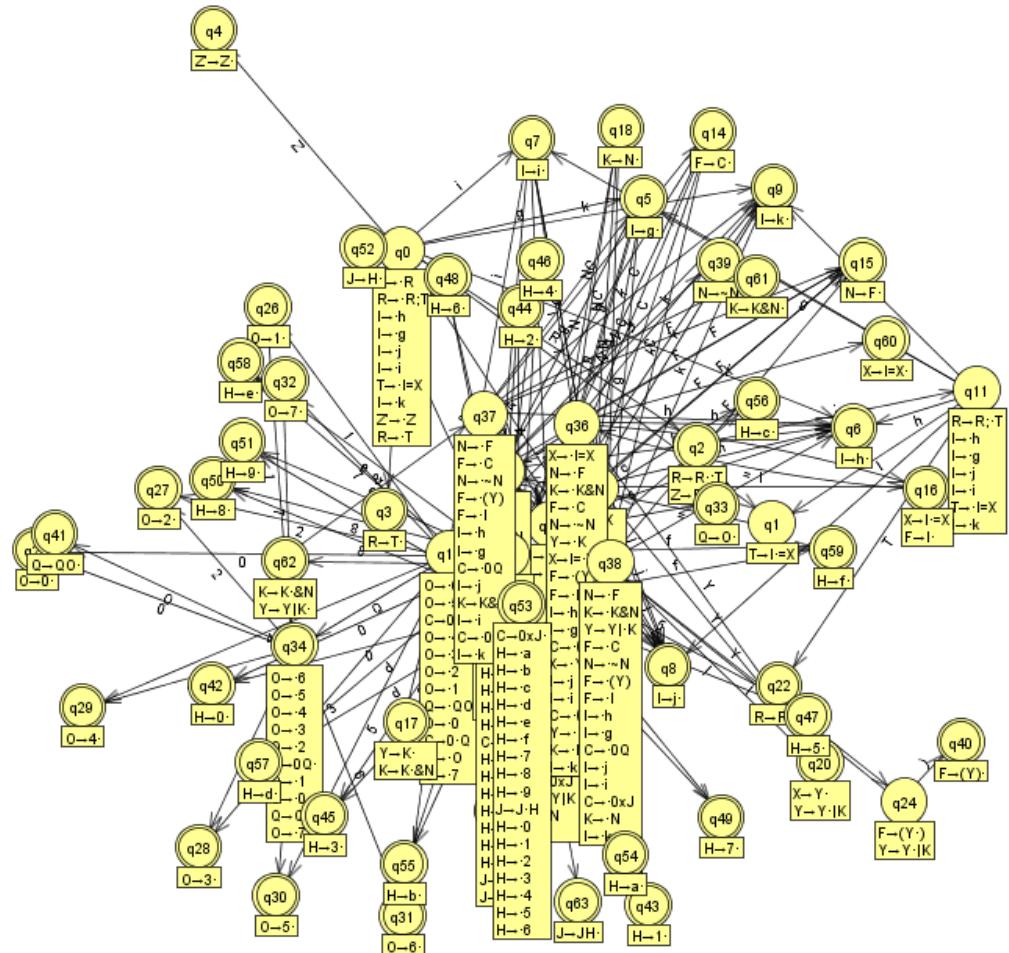


Рисунок 13 – Канонической набор LR(0)-ситуаций

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.

Start Step Derivation Table

Input g=~(h|075)&0x1b3  
Input Remaining \$  
Stack Z0

Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the text fields)

Table Text Size

I→g	g=~(h 075)&0x1b3
I→h	I=~(h 075)&0x1b3
F→I	I=~(I 075)&0x1b3
N→F	I=~(F 075)&0x1b3
K→N	I=~(N 075)&0x1b3
Y→K	I=~(K 075)&0x1b3
O→7	I=~(Y 005)&0x1b3
Q→O	I=~(Y 005)&0x1b3
O→5	I=~(Y 005)&0x1b3
Q→QO	I=~(Y 00Q)&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=K8&0xHb3
J→H	I=K8&0xJb3
H→b	I=K8&0xJh3
J→JH	I=K8&0xJ3
H→3	I=K8&0xJH
J→JH	I=K8&0xJ
C→0J	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»

Start Step Derivation Table

Input k=h=0x2f&03  
Input Remaining \$  
Stack Z0

Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the text fields)

Table Text Size

I→k	k=h=0x2f&03
I→h	I=h=0x2f&03
H→2	I=I=0x2f&03
J→H	I=I=0xHf&03
H→f	I=I=0xJf&03
J→JH	I=I=0xJH&03
C→0J	I=I=0xJ&03
F→C	I=I=C&03
N→F	I=I=F&03
K→N	I=I=N&03
O→3	I=I=K&03
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»

Derivation Table	
Input	$i=(g 0x7a)&05;j=\sim k 071$
Input Remaining \$	
Stack	Z0
Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the text fields)	
J→JH	$I=(Y 0xJ)&05;j=\sim k 071$
C→0xJ	$I=(Y C)&05;j=\sim k 071$
F→C	$I=(Y F)&05;j=\sim k 071$
N→F	$I=(Y N)&05;j=\sim k 071$
K→N	$I=(Y K)&05;j=\sim k 071$
Y→YIK	$I=(Y)&05;j=\sim k 071$
F→(Y)	$I=F&05;j=\sim k 071$
N→F	$I=N&05;j=\sim k 071$
K→N	$I=K&05;j=\sim k 071$
O→5	$I=K&O j=\sim k 071$
Q→0	$I=K&Q j=\sim k 071$
C→0Q	$I=K&C;j=\sim k 071$
F→C	$I=K&F;j=\sim k 071$
N→F	$I=K&N;j=\sim k 071$
K→K&N	$I=K;j=\sim k 071$
Y→K	$I=Y;j=\sim k 071$
X→Y	$I=X;j=\sim k 071$
T→I=X	$T;j=\sim k 071$
R→T	$R;j=\sim k 071$
I→j	$R;I=\sim k 071$
I→k	$R;I=\sim k 071$
F→I	$R;I=\sim F 071$
N→F	$R;I=\sim N 071$
N→~N	$R;I=\sim N 071$
K→N	$R;I=\sim K 071$
Y→K	$R;I=Y 071$
O→7	$R;I=Y 001$
Q→0	$R;I=Y 0Q1$
O→1	$R;I=Y 0Q$
Q→QO	$R;I=Y C$
C→0Q	$R;I=Y F$
F→C	$R;I=Y N$
N→F	$R;I=Y K$
K→N	$R;I=Y$
Y→YIK	$R;I=X$
X→Y	$R;T$
T→I=X	$R$
R→R;T	$Z$
Z→R	

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

Derivation Table	
Input	g=h;i=07;j=0x3c
Input Remaining \$	\$
Stack	Z0
Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the text fields)	
Table Text Size	
I→g	g=h;i=07;j=0x3c
I→h	I=h;i=07;j=0x3c
F→I	I=i;i=07;j=0x3c
N→F	I=F;i=07;j=0x3c
K→N	I=N;i=07;j=0x3c
Y→K	I=K;i=07;j=0x3c
X→Y	I=Y;i=07;j=0x3c
T→I=X	I=X;i=07;j=0x3c
R→T	T;j=07;j=0x3c
I→I	R;j=07;j=0x3c
O→7	R;j=07;j=0x3c
Q→O	R;j=0O;j=0x3c
C→0Q	R;j=0Q;j=0x3c
F→C	R;j=C;j=0x3c
N→F	R;j=F;j=0x3c
K→N	R;j=N;j=0x3c
Y→K	R;j=K;j=0x3c
X→Y	R;j=Y;j=0x3c
T→I=X	R;j=X;j=0x3c
R→R;T	R;j=T;j=0x3c
I→J	R;j=R;j=0x3c
H→3	R;j=3;j=0x3c
J→H	R;j=H;j=0x3c
H→c	R;j=c;j=0x3c
J→JH	R;j=JH;j=0x3c
C→0J	R;j=0J;j=0x3c
F→C	R;j=C;j=0x3c
N→F	R;j=F;j=0x3c
K→N	R;j=N;j=0x3c
Y→K	R;j=K;j=0x3c
X→Y	R;j=Y;j=0x3c
T→I=X	R;j=X;j=0x3c
R→R;T	R;j=T;j=0x3c
Z→R	R;j=R;j=0x3c
	Z

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

Derivation Table	
Input	h=~k&03;g
Input Remaining \$	\$
Stack	I111;R0
Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the text fields)	
Table Text Size	
I→h	h=~k&03;g
I→k	I=~k&03;g
F→I	I=~&03;g
N→F	I=~F&03;g
N→N	I=~N&03;g
K→N	I=&03;g
O→3	I=K&03;g
Q→O	I=K&0O;g
C→0Q	I=K&0Q;g
F→C	I=K&C;g
N→F	I=K&F;g
K→K&N	I=K&N;g
Y→K	I=K;g
X→Y	I=Y;g
T→I=X	I=X;g
R→T	T;g
I→g	R;g
	R;I

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

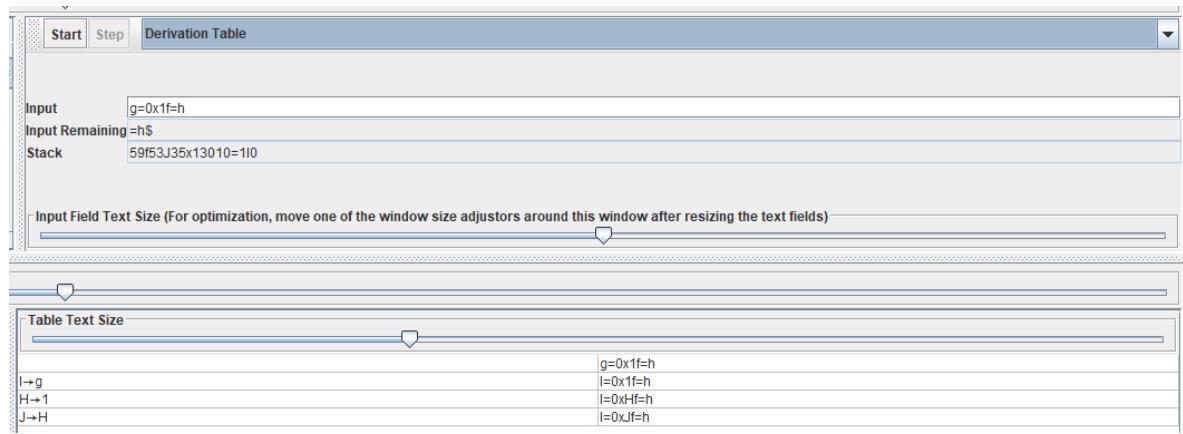


Рисунок 20 - Тест для цепочки « $g=0x1f=h$ »

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  $\sim$ ,  $\&$ ,  $=$  и разделителем  $;$ . По данной грамматике реализован синтаксический анализатор, схема грамматики показана на рисунке 11.

```

#ID, CONST, '~', '&', '|', '=', '(', ')', ';', EOP
#LIST      -> ASSIGN LIST_TAIL
#LIST_TAIL -> ';' ASSIGN LIST_TAIL | ε
#ASSIGN     -> EXPR ASSIGN_TAIL
#ASSIGN_TAIL -> '=' ASSIGN | ε
#EXPR      -> NEXT_EXPR ADD
#ADD       -> '|' NEXT_EXPR ADD | ε
#NEXT_EXPR -> UNARY MUL
#MUL       -> '&' UNARY MUL | ε
#UNARY     -> '~' UNARY | PRIMARY
#PRIMARY   -> ID | CONST | '(' EXPR ')'
#CONST     -> '0' CONST_TAIL | NZDIGIT DIGIT_TAIL
#CONST_TAIL -> 'b' BINARY | 'B' BINARY | 'x' HEX | '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' | 'b' | 'c' | 'd' | 'e' | 'f'
#                   | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'
#DIGIT_TAIL -> DIGIT DIGIT_TAIL | ε
#NZDIGIT   -> '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
#DIGIT     -> NZDIGIT | '0'
# 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

```

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  $A \rightarrow \alpha$  есть  $\epsilon \in \text{FIRST}(\alpha)$ , то  $\text{FOLLOW}(A) \cap \text{FIRST}(\beta) = \emptyset$  для всех других продукции  $A \rightarrow \beta$ ». В-четверых, в таблице синтаксического анализа каждая ячейка содержит не больше одной продукции, как показано на рисунке 22.

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

```

1 import re
2 # ТУРН ТОКЕНОВ
3 NEG_SIGN = 'NEG' # -
4 MUL_SIGN = 'MUL' # *
5 ADD_SIGN = 'ADD' # +
6 ASSIGN_SIGN = 'ASSIGN' # =
7 LPAR_SIGN = 'LPAR' # (
8 RPAR_SIGN = 'RPAR' # )
9 SEMI_SIGN = 'SEMI' # ;
10 ID_SIGN = 'ID' # идентификатор (1-2 буквы)
11 CONST_SIGN = 'CONST' # константа (10/2/16)
12 EOP = 'EOP' # конец ввода
13 UNDEF = 'UNDEF' # неизвестное
14
15 # ===== разбор CONST по синтаксику =====
16 # CONST -> 0 CONST_TAIL | NZDIGIT DIGIT_TAIL
17 # CONST_TAIL -> 'b' BINARY | 'x' HEX | DIGIT_TAIL
18 # BINARY -> BIT BIT_TAIL | e
19 # BIT_TAIL -> BIT BIT_TAIL | e
20 # BIT -> '0' | '1'
21 # HEX -> HEXDIGIT HEX_TAIL
22 # HEX_TAIL -> HEXDIGIT HEX_TAIL | e
23 # HEXDIGIT -> DIGIT | a..f
24 # DIGIT_TAIL -> DIGIT DIGIT_TAIL | e
25 # NZDIGIT -> 1..9
26 # DIGIT -> NZDIGIT | 0
27
28 class Lexer: & Yannik
29     def __init__(self, text: str): & Yannik
30         self.text = text
31         self.pos = 0
32
33     def skip_spaces(self): & Yannik
34         while self.pos < len(self.text) and self.text[self.pos].isspace():
35
36             def get_token(self): & Yannik
37                 self.skip_spaces()
38                 if self.pos >= len(self.text):
39                     return (EOP, '')
40
41                 ch = self.text[self.pos]
42
43                 # одиночные символы операций и скобок
44                 if ch == '-': self.pos += 1; return (NEG_SIGN, '-')
45                 if ch == '*': self.pos += 1; return (MUL_SIGN, '*')
46                 if ch == '+': self.pos += 1; return (ADD_SIGN, '+')
47                 if ch == '=': self.pos += 1; return (ASSIGN_SIGN, '=')
48                 if ch == '(': self.pos += 1; return (LPAR_SIGN, '(')
49                 if ch == ')': self.pos += 1; return (RPAR_SIGN, ')')
50                 if ch == ';': self.pos += 1; return (SEMI_SIGN, ';')
51
52                 # ID: ONE_SYM SEC_SYM, где ONE_SYM — буква а..z
53                 if ch.isalpha():
54                     start = self.pos
55                     self.pos += 1
56                     if self.pos < len(self.text) and self.text[self.pos].isalpha():
57                         self.pos += 1 # допускаем 2-символьный ID
58                     return (ID_SIGN, self.text[start:self.pos])
59
60                 # CONST: начинается с цифры, затем буквы/цифры (0, 00..., 0х..., десятичные)
61                 if ch.isdigit():
62                     start = self.pos
63                     self.pos += 1
64
65             # CONST_TAIL -> 0 CONST_TAIL / NZDIGIT DIGIT_TAIL
66             def CONST_TAIL_RULE(self, s: str): & Yannik
67                 ch = self.c_peek(s)
68                 if ch in ('0', 'B'):
69                     self.c_pos += 1
70                     self.CONST_RULE(s)
71                 elif ch in ('x', 'X'):
72                     self.c_pos += 1
73                     self.HEX_RULE(s)
74                 else:
75                     self.DIGIT_TAIL_RULE(s)
76
77             # BINARY -> BIT BIT_TAIL
78             def BINARY_RULE(self, s: str): & Yannik
79                 self.BIT_RULE(s)
80                 self.BIT_TAIL_RULE(s)
81
82             # BIT_TAIL -> BIT BIT_TAIL | e
83             def BIT_TAIL_RULE(self, s: str): & Yannik
84                 ch = self.c_peek(s)
85                 if ch in ('0', '1'):
86                     self.BIT_RULE(s)
87                     self.BIT_TAIL_RULE(s)
88                 elif ch == 'e':
89                     self.c_pos += 1
90                     self.DIGIT_TAIL_RULE(s)
91
92             # CONST_TAIL -> '0' BINARY | 'x' HEX | DIGIT_TAIL
93             def CONST_TAIL_RULE(self, s: str): & Yannik
94                 ch = self.c_peek(s)

```

Рисунок 23 – Первая часть кода реализованной программы на python

```

me 75     class Parser: & Yannik
100     def parse_const_grammar(self, s: str): & Yannik
101
102         try:
103             self.CONST_RULE(s)
104             if self.c_pos != len(s):
105                 self.error()
106         finally:
107             del self.c_pos
108
109         def c_peek(self, s: str): & Yannik
110             return s[self.c_pos] if self.c_pos < len(s) else None
111
112         def c_eat(self, s: str, allowed: str): & Yannik
113             ch = self.c_peek(s)
114             if ch is None or ch not in allowed:
115                 self.error()
116             self.c_pos += 1
117             return ch
118
119
120             # CONST -> 0 CONST_TAIL / NZDIGIT DIGIT_TAIL
121             def CONST_RULE(self, s: str): & Yannik
122                 ch = self.c_peek(s)
123                 if ch == '0':
124                     self.c_pos += 1
125                     self.CONST_TAIL_RULE(s)
126                 elif ch is not None and ch in "123456789":
127                     self.c_pos += 1
128                     self.DIGIT_TAIL_RULE(s)
129                 else:
130                     self.error()
131
132             # CONST_TAIL -> '0' BINARY | 'x' HEX | DIGIT_TAIL
133             def CONST_TAIL_RULE(self, s: str): & Yannik
134                 ch = self.c_peek(s)
135
136             # CONST_TAIL -> 'b' BINARY | 'x' HEX | DIGIT_TAIL
137             def CONST_TAIL_RULE(self, s: str): & Yannik
138                 ch = self.c_peek(s)
139
140             # BINARY -> BIT BIT_TAIL
141             def BINARY_RULE(self, s: str): & Yannik
142                 self.BIT_RULE(s)
143                 self.BIT_TAIL_RULE(s)
144
145             # BIT_TAIL -> BIT BIT_TAIL | e
146             def BIT_TAIL_RULE(self, s: str): & Yannik
147                 ch = self.c_peek(s)
148                 if ch in ('0', '1'):
149                     self.BIT_RULE(s)
150                     self.BIT_TAIL_RULE(s)
151                 elif ch == 'e':
152                     self.c_pos += 1
153                     self.DIGIT_TAIL_RULE(s)
154
155             # BIT -> 0 | 1
156             def BIT_RULE(self, s: str): & Yannik
157                 self.c_eat(s, allowed: "01")
158
159             # HEX -> HEXDIGIT HEX_TAIL
160             def HEX_RULE(self, s: str): & Yannik
161                 self.HEXDIGIT_RULE(s)
162                 self.HEX_TAIL_RULE(s)
163
164             # HEXDIGIT -> DIGIT | a..f
165             def HEXDIGIT_RULE(self, s: str): & Yannik
166                 ch = self.c_peek(s)
167
168             # DIGIT_TAIL -> DIGIT DIGIT_TAIL | e
169             def DIGIT_TAIL_RULE(self, s: str): & Yannik
170                 ch = self.c_peek(s)
171
172             # DIGIT -> NZDIGIT | 0
173             def DIGIT_RULE(self, s: str): & Yannik
174                 ch = self.c_peek(s)
175
176             # NZDIGIT -> 1..9
177             def NZDIGIT_RULE(self, s: str): & Yannik
178                 ch = self.c_peek(s)
179
180             # EOP -> EOP
181             def EOP_RULE(self, s: str): & Yannik
182                 pass
183
184             # ID -> ID
185             def ID_RULE(self, s: str): & Yannik
186                 start = self.c_pos
187                 self.c_pos += 1
188                 if self.c_pos < len(s) and self.c_peek(s).isalpha():
189                     self.c_pos += 1
190                     self.ID_RULE(s)
191
192             # ASSIGN -> ASSIGN
193             def ASSIGN_RULE(self, s: str): & Yannik
194                 pass
195
196             # ADD -> ADD
197             def ADD_RULE(self, s: str): & Yannik
198                 pass
199
200             # MUL -> MUL
201             def MUL_RULE(self, s: str): & Yannik
202                 pass
203
204             # NEG -> NEG
205             def NEG_RULE(self, s: str): & Yannik
206                 pass
207
208             # RPAR -> RPAR
209             def RPAR_RULE(self, s: str): & Yannik
210                 pass
211
212             # LPAR -> LPAR
213             def LPAR_RULE(self, s: str): & Yannik
214                 pass
215
216             # SEMI -> SEMI
217             def SEMI_RULE(self, s: str): & Yannik
218                 pass
219
220             # EOP -> EOP
221             def EOP_RULE(self, s: str): & Yannik
222                 pass
223
224             # UNDEF -> UNDEF
225             def UNDEF_RULE(self, s: str): & Yannik
226                 pass
227
228             # EOF -> EOF
229             def EOF_RULE(self, s: str): & Yannik
230                 pass
231
232             # UNDEF -> UNDEF
233             def UNDEF_RULE(self, s: str): & Yannik
234                 pass
235
236             # UNDEF -> UNDEF
237             def UNDEF_RULE(self, s: str): & Yannik
238                 pass
239
240             # UNDEF -> UNDEF
241             def UNDEF_RULE(self, s: str): & Yannik
242                 pass
243
244             # UNDEF -> UNDEF
245             def UNDEF_RULE(self, s: str): & Yannik
246                 pass
247
248             # UNDEF -> UNDEF
249             def UNDEF_RULE(self, s: str): & Yannik
250                 pass
251
252             # UNDEF -> UNDEF
253             def UNDEF_RULE(self, s: str): & Yannik
254                 pass
255
256             # UNDEF -> UNDEF
257             def UNDEF_RULE(self, s: str): & Yannik
258                 pass
259
260             # UNDEF -> UNDEF
261             def UNDEF_RULE(self, s: str): & Yannik
262                 pass
263
264             # UNDEF -> UNDEF
265             def UNDEF_RULE(self, s: str): & Yannik
266                 pass
267
268             # UNDEF -> UNDEF
269             def UNDEF_RULE(self, s: str): & Yannik
270                 pass
271
272             # UNDEF -> UNDEF
273             def UNDEF_RULE(self, s: str): & Yannik
274                 pass
275
276             # UNDEF -> UNDEF
277             def UNDEF_RULE(self, s: str): & Yannik
278                 pass
279
280             # UNDEF -> UNDEF
281             def UNDEF_RULE(self, s: str): & Yannik
282                 pass
283
284             # UNDEF -> UNDEF
285             def UNDEF_RULE(self, s: str): & Yannik
286                 pass
287
288             # UNDEF -> UNDEF
289             def UNDEF_RULE(self, s: str): & Yannik
290                 pass
291
292             # UNDEF -> UNDEF
293             def UNDEF_RULE(self, s: str): & Yannik
294                 pass
295
296             # UNDEF -> UNDEF
297             def UNDEF_RULE(self, s: str): & Yannik
298                 pass
299
200             # UNDEF -> UNDEF
201             def UNDEF_RULE(self, s: str): & Yannik
202                 pass
203
204             # UNDEF -> UNDEF
205             def UNDEF_RULE(self, s: str): & Yannik
206                 pass
207
208             # UNDEF -> UNDEF
209             def UNDEF_RULE(self, s: str): & Yannik
210                 pass
211
212             # UNDEF -> UNDEF
213             def UNDEF_RULE(self, s: str): & Yannik
214                 pass
215
216             # UNDEF -> UNDEF
217             def UNDEF_RULE(self, s: str): & Yannik
218                 pass
219
220             # UNDEF -> UNDEF
221             def UNDEF_RULE(self, s: str): & Yannik
222                 pass
223
224             # UNDEF -> UNDEF
225             def UNDEF_RULE(self, s: str): & Yannik
226                 pass
227
228             # UNDEF -> UNDEF
229             def UNDEF_RULE(self, s: str): & Yannik
230                 pass
231
232             # UNDEF -> UNDEF
233             def UNDEF_RULE(self, s: str): & Yannik
234                 pass
235
236             # UNDEF -> UNDEF
237             def UNDEF_RULE(self, s: str): & Yannik
238                 pass
239
240             # UNDEF -> UNDEF
241             def UNDEF_RULE(self, s: str): & Yannik
242                 pass
243
244             # UNDEF -> UNDEF
245             def UNDEF_RULE(self, s: str): & Yannik
246                 pass
247
248             # UNDEF -> UNDEF
249             def UNDEF_RULE(self, s: str): & Yannik
250                 pass
251
252             # UNDEF -> UNDEF
253             def UNDEF_RULE(self, s: str): & Yannik
254                 pass
255
256             # UNDEF -> UNDEF
257             def UNDEF_RULE(self, s: str): & Yannik
258                 pass
259
260             # UNDEF -> UNDEF
261             def UNDEF_RULE(self, s: str): & Yannik
262                 pass
263
264             # UNDEF -> UNDEF
265             def UNDEF_RULE(self, s: str): & Yannik
266                 pass
267
268             # UNDEF -> UNDEF
269             def UNDEF_RULE(self, s: str): & Yannik
270                 pass
271
272             # UNDEF -> UNDEF
273             def UNDEF_RULE(self, s: str): & Yannik
274                 pass
275
276             # UNDEF -> UNDEF
277             def UNDEF_RULE(self, s: str): & Yannik
278                 pass
279
280             # UNDEF -> UNDEF
281             def UNDEF_RULE(self, s: str): & Yannik
282                 pass
283
284             # UNDEF -> UNDEF
285             def UNDEF_RULE(self, s: str): & Yannik
286                 pass
287
288             # UNDEF -> UNDEF
289             def UNDEF_RULE(self, s: str): & Yannik
290                 pass
291
292             # UNDEF -> UNDEF
293             def UNDEF_RULE(self, s: str): & Yannik
294                 pass
295
296             # UNDEF -> UNDEF
297             def UNDEF_RULE(self, s: str): & Yannik
298                 pass
299
200             # UNDEF -> UNDEF
201             def UNDEF_RULE(self, s: str): & Yannik
202                 pass
203
204             # UNDEF -> UNDEF
205             def UNDEF_RULE(self, s: str): & Yannik
206                 pass
207
208             # UNDEF -> UNDEF
209             def UNDEF_RULE(self, s: str): & Yannik
210                 pass
211
212             # UNDEF -> UNDEF
213             def UNDEF_RULE(self, s: str): & Yannik
214                 pass
215
216             # UNDEF -> UNDEF
217             def UNDEF_RULE(self, s: str): & Yannik
218                 pass
219
220             # UNDEF -> UNDEF
221             def UNDEF_RULE(self, s: str): & Yannik
222                 pass
223
224             # UNDEF -> UNDEF
225             def UNDEF_RULE(self, s: str): & Yannik
226                 pass
227
228             # UNDEF -> UNDEF
229             def UNDEF_RULE(self, s: str): & Yannik
230                 pass
231
232             # UNDEF -> UNDEF
233             def UNDEF_RULE(self, s: str): & Yannik
234                 pass
235
236             # UNDEF -> UNDEF
237             def UNDEF_RULE(self, s: str): & Yannik
238                 pass
239
230             # UNDEF -> UNDEF
231             def UNDEF_RULE(self, s: str): & Yannik
232                 pass
233
234             # UNDEF -> UNDEF
235             def UNDEF_RULE(self, s: str): & Yannik
236                 pass
237
238             # UNDEF -> UNDEF
239             def UNDEF_RULE(self, s: str): & Yannik
240                 pass
241
242             # UNDEF -> UNDEF
243             def UNDEF_RULE(self, s: str): & Yannik
244                 pass
245
246             # UNDEF -> UNDEF
247             def UNDEF_RULE(self, s: str): & Yannik
248                 pass
249
250             # UNDEF -> UNDEF
251             def UNDEF_RULE(self, s: str): & Yannik
252                 pass
253
254             # UNDEF -> UNDEF
255             def UNDEF_RULE(self, s: str): & Yannik
256                 pass
257
258             # UNDEF -> UNDEF
259             def UNDEF_RULE(self, s: str): & Yannik
260                 pass
261
262             # UNDEF -> UNDEF
263             def UNDEF_RULE(self, s: str): & Yannik
264                 pass
265
266             # UNDEF -> UNDEF
267             def UNDEF_RULE(self, s: str): & Yannik
268                 pass
269
270             # UNDEF -> UNDEF
271             def UNDEF_RULE(self, s: str): & Yannik
272                 pass
273
274             # UNDEF -> UNDEF
275             def UNDEF_RULE(self, s: str): & Yannik
276                 pass
277
278             # UNDEF -> UNDEF
279             def UNDEF_RULE(self, s: str): & Yannik
280                 pass
281
282             # UNDEF -> UNDEF
283             def UNDEF_RULE(self, s: str): & Yannik
284                 pass
285
286             # UNDEF -> UNDEF
287             def UNDEF_RULE(self, s: str): & Yannik
288                 pass
289
290             # UNDEF -> UNDEF
291             def UNDEF_RULE(self, s: str): & Yannik
292                 pass
293
294             # UNDEF -> UNDEF
295             def UNDEF_RULE(self, s: str): & Yannik
296                 pass
297
298             # UNDEF -> UNDEF
299             def UNDEF_RULE(self, s: str): & Yannik
200                 pass
201
202             # UNDEF -> UNDEF
203             def UNDEF_RULE(self, s: str): & Yannik
204                 pass
205
206             # UNDEF -> UNDEF
207             def UNDEF_RULE(self, s: str): & Yannik
208                 pass
209
210             # UNDEF -> UNDEF
211             def UNDEF_RULE(self, s: str): & Yannik
212                 pass
213
214             # UNDEF -> UNDEF
215             def UNDEF_RULE(self, s: str): & Yannik
216                 pass
217
218             # UNDEF -> UNDEF
219             def UNDEF_RULE(self, s: str): & Yannik
220                 pass
221
222             # UNDEF -> UNDEF
223             def UNDEF_RULE(self, s: str): & Yannik
224                 pass
225
226             # UNDEF -> UNDEF
227             def UNDEF_RULE(self, s: str): & Yannik
228                 pass
229
220             # UNDEF -> UNDEF
221             def UNDEF_RULE(self, s: str): & Yannik
222                 pass
223
224             # UNDEF -> UNDEF
225             def UNDEF_RULE(self, s: str): & Yannik
226                 pass
227
228             # UNDEF -> UNDEF
229             def UNDEF_RULE(self, s: str): & Yannik
230                 pass
231
232             # UNDEF -> UNDEF
233             def UNDEF_RULE(self, s: str): & Yannik
234                 pass
235
236             # UNDEF -> UNDEF
237             def UNDEF_RULE(self, s: str): & Yannik
238                 pass
239
230             # UNDEF -> UNDEF
231             def UNDEF_RULE(self, s: str): & Yannik
232                 pass
233
234             # UNDEF -> UNDEF
235             def UNDEF_RULE(self, s: str): & Yannik
236                 pass
237
238             # UNDEF -> UNDEF
239             def UNDEF_RULE(self, s: str): & Yannik
240                 pass
241
242             # UNDEF -> UNDEF
243             def UNDEF_RULE(self, s: str): & Yannik
244                 pass
245
246             # UNDEF -> UNDEF
247             def UNDEF_RULE(self, s: str): & Yannik
248                 pass
249
240             # UNDEF -> UNDEF
241             def UNDEF_RULE(self, s: str): & Yannik
242                 pass
243
244             # UNDEF -> UNDEF
245             def UNDEF_RULE(self, s: str): & Yannik
246                 pass
247
248             # UNDEF -> UNDEF
249             def UNDEF_RULE(self, s: str): & Yannik
250                 pass
251
252             # UNDEF -> UNDEF
253             def UNDEF_RULE(self, s: str): & Yannik
254                 pass
255
256             # UNDEF -> UNDEF
257             def UNDEF_RULE(self, s: str): & Yannik
258                 pass
259
250             # UNDEF -> UNDEF
251             def UNDEF_RULE(self, s: str): & Yannik
252                 pass
253
254             # UNDEF -> UNDEF
255             def UNDEF_RULE(self, s: str): & Yannik
256                 pass
257
258             # UNDEF -> UNDEF
259             def UNDEF_RULE(self, s: str): & Yannik
260                 pass
261
262             # UNDEF -> UNDEF
263             def UNDEF_RULE(self, s: str): & Yannik
264                 pass
265
266             # UNDEF -> UNDEF
267             def UNDEF_RULE(self, s: str): & Yannik
268                 pass
269
260             # UNDEF -> UNDEF
261             def UNDEF_RULE(self, s: str): & Yannik
262                 pass
263
264             # UNDEF -> UNDEF
265             def UNDEF_RULE(self, s: str): & Yannik
266                 pass
267
268             # UNDEF -> UNDEF
269             def UNDEF_RULE(self, s: str): & Yannik
270                 pass
271
272             # UNDEF -> UNDEF
273             def UNDEF_RULE(self, s: str): & Yannik
274                 pass
275
276             # UNDEF -> UNDEF
277             def UNDEF_RULE(self, s: str): & Yannik
278                 pass
279
270             # UNDEF -> UNDEF
271             def UNDEF_RULE(self, s: str): & Yannik
272                 pass
273
274             # UNDEF -> UNDEF
275             def UNDEF_RULE(self, s: str): & Yannik
276                 pass
277
278             # UNDEF -> UNDEF
279             def UNDEF_RULE(self, s: str): & Yannik
280                 pass
281
282             # UNDEF -> UNDEF
283             def UNDEF_RULE(self, s: str): & Yannik
284                 pass
285
286             # UNDEF -> UNDEF
287             def UNDEF_RULE(self, s: str): & Yannik
288                 pass
289
280             # UNDEF -> UNDEF
281             def UNDEF_RULE(self, s: str): & Yannik
282                 pass
283
284             # UNDEF -> UNDEF
285             def UNDEF_RULE(self, s: str): & Yannik
286                 pass
287
288             # UNDEF -> UNDEF
289             def UNDEF_RULE(self, s: str): & Yannik
290                 pass
291
292             # UNDEF -> UNDEF
293             def UNDEF_RULE(self, s: str): & Yannik
294                 pass
295
296             # UNDEF -> UNDEF
297             def UNDEF_RULE(self, s: str): & Yannik
298                 pass
299
200             # UNDEF -> UNDEF
201             def UNDEF_RULE(self, s: str): & Yannik
202                 pass
203
204             # UNDEF -> UNDEF
205             def UNDEF_RULE(self, s: str): & Yannik
206                 pass
207
208             # UNDEF -> UNDEF
209             def UNDEF_RULE(self, s: str): & Yannik
210                 pass
211
212             # UNDEF -> UNDEF
213             def UNDEF_RULE(self, s: str): & Yannik
214                 pass
215
216             # UNDEF -> UNDEF
217             def UNDEF_RULE(self, s: str): & Yannik
218                 pass
219
210             # UNDEF -> UNDEF
211             def UNDEF_RULE(self, s: str): & Yannik
212                 pass
213
214             # UNDEF -> UNDEF
215             def UNDEF_RULE(self, s: str): & Yannik
216                 pass
217
218             # UNDEF -> UNDEF
219             def UNDEF_RULE(self, s: str): & Yannik
220                 pass
221
222             # UNDEF -> UNDEF
223             def UNDEF_RULE(self, s: str): & Yannik
224                 pass
225
226             # UNDEF -> UNDEF
227             def UNDEF_RULE(self, s: str): & Yannik
228                 pass
229
220             # UNDEF -> UNDEF
221             def UNDEF_RULE(self, s: str): & Yannik
222                 pass
223
224             # UNDEF -> UNDEF
225             def UNDEF_RULE(self, s: str): & Yannik
226                 pass
227
228             # UNDEF -> UNDEF
229             def UNDEF_RULE(self, s: str): & Yannik
230                 pass
231
232             # UNDEF -> UNDEF
233             def UNDEF_RULE(self, s: str): & Yannik
234                 pass
235
236             # UNDEF -> UNDEF
237             def UNDEF_RULE(self, s: str): & Yannik
238                 pass
239
230             # UNDEF -> UNDEF
231             def UNDEF_RULE(self, s: str): & Yannik
232                 pass
233
234             # UNDEF -> UNDEF
235             def UNDEF_RULE(self, s: str): & Yannik
236                 pass
237
238             # UNDEF -> UNDEF
239             def UNDEF_RULE(self, s: str): & Yannik
240                 pass
241
242             # UNDEF -> UNDEF
243             def UNDEF_RULE(self, s: str): & Yannik
244                 pass
245
246             # UNDEF -> UNDEF
247             def UNDEF_RULE(self, s: str): & Yannik
248                 pass
249
240             # UNDEF -> UNDEF
241             def UNDEF_RULE(self, s: str): & Yannik
242                 pass
243
244             # UNDEF -> UNDEF
245             def UNDEF_RULE(self, s: str): & Yannik
246                 pass
247
248             # UNDEF -> UNDEF
249             def UNDEF_RULE(self, s: str): & Yannik
250                 pass
251
252             # UNDEF -> UNDEF
253             def UNDEF_RULE(self, s: str): & Yannik
254                 pass
255
256             # UNDEF -> UNDEF
257             def UNDEF_RULE(self, s: str): & Yannik
258                 pass
259
250             # UNDEF -> UNDEF
251             def UNDEF_RULE(self, s: str): & Yannik
252                 pass
253
254             # UNDEF -> UNDEF
255             def UNDEF_RULE(self, s: str): & Yannik
256                 pass
257
258             # UNDEF -> UNDEF
259             def UNDEF_RULE(self, s: str): & Yannik
260                 pass
261
262             # UNDEF -> UNDEF
263             def UNDEF_RULE(self, s: str): & Yannik
264                 pass
265
266             # UNDEF -> UNDEF
267             def UNDEF_RULE(self, s: str): & Yannik
268                 pass
269
260             # UNDEF -> UNDEF
261             def UNDEF_RULE(self, s: str): & Yannik
262                 pass
263
264             # UNDEF -> UNDEF
265             def UNDEF_RULE(self, s: str): & Yannik
266                 pass
267
268             # UNDEF -> UNDEF
269             def UNDEF_RULE(self, s: str): & Yannik
270                 pass
271
272             # UNDEF -> UNDEF
273             def UNDEF_RULE(self, s: str): & Yannik
274                 pass
275
276             # UNDEF -> UNDEF
277             def UNDEF_RULE(self, s: str): & Yannik
278                 pass
279
270             # UNDEF -> UNDEF
271             def UNDEF_RULE(self, s: str): & Yannik
272                 pass
273
274             # UNDEF -> UNDEF
275             def UNDEF_RULE(self, s: str): & Yannik
276                 pass
277
278             # UNDEF -> UNDEF
279             def UNDEF_RULE(self, s: str): & Yannik
280                 pass
281
282             # UNDEF -> UNDEF
283             def UNDEF_RULE(self, s: str): & Yannik
284                 pass
285
286             # UNDEF -> UNDEF
287             def UNDEF_RULE(self, s: str): & Yannik
288                 pass
289
280             # UNDEF -> UNDEF
281             def UNDEF_RULE(self, s: str): & Yannik
282                 pass
283
284             # UNDEF -> UNDEF
285             def UNDEF_RULE(self, s: str): & Yannik
286                 pass
287
288             # UNDEF -> UNDEF
289             def UNDEF_RULE(self, s: str): & Yannik
290                 pass
291
292             # UNDEF -> UNDEF
293             def UNDEF_RULE(self, s: str): & Yannik
294                 pass
295
296             # UNDEF -> UNDEF
297             def UNDEF_RULE(self, s: str): & Yannik
298                 pass
299
200             # UNDEF -> UNDEF
201             def UNDEF_RULE(self, s: str): & Yannik
202                 pass
203
204             # UNDEF -> UNDEF
205             def UNDEF_RULE(self, s: str): & Yannik
206                 pass
207
208             # UNDEF -> UNDEF
209             def UNDEF_RULE(self, s: str): & Yannik
210                 pass
211
212             # UNDEF -> UNDEF
213             def UNDEF_RULE(self, s: str): & Yannik
214                 pass
215
216             # UNDEF -> UNDEF
217             def UNDEF_RULE(self, s: str): & Yannik
218                 pass
219
210             # UNDEF -> UNDEF
211             def UNDEF_RULE(self, s: str): & Yannik
212                 pass
213
214             # UNDEF -> UNDEF
215             def UNDEF_RULE(self, s: str): & Yannik
216                 pass
217
218             # UNDEF -> UNDEF
219             def UNDEF_RULE(self, s: str): & Yannik
220                 pass
221
222             # UNDEF -> UNDEF
223             def UNDEF_RULE(self, s: str): & Yannik
224                 pass
225
226             # UNDEF -> UNDEF
227             def UNDEF_RULE(self, s: str): & Yannik
228                 pass
229
220             # UNDEF -> UNDEF
221             def UNDEF_RULE(self, s: str): & Yannik
222                 pass
223
224             # UNDEF -> UNDEF
225             def UNDEF_RULE(self, s: str): & Yannik
226                 pass
227
228             # UNDEF -> UNDEF
229             def UNDEF_RULE(self, s: str): & Yannik
230                 pass
231
232             # UNDEF -> UNDEF
233             def UNDEF_RULE(self, s: str): & Yannik
234                 pass
235
236             # UNDEF -> UNDEF
237             def UNDEF_RULE(self, s: str): & Yannik
238                 pass
239
230             # UNDEF -> UNDEF
231             def UNDEF_RULE(self, s: str): & Yannik
232                 pass
233
234             # UNDEF -> UNDEF
235             def UNDEF_RULE(self, s: str): & Yannik
236                 pass
237
238             # UNDEF -> UNDEF
239             def UNDEF_RULE(self, s: str): & Yannik
240                 pass
241
242             # UNDEF -> UNDEF
243             def UNDEF_RULE(self, s: str): & Yannik
244                 pass
245
246             # UNDEF -> UNDEF
247             def UNDEF_RULE(self, s: str): & Yannik
248                 pass
249
240             # UNDEF -> UNDEF
241             def UNDEF_RULE(self, s: str): & Yannik
242                 pass
243
244             # UNDEF -> UNDEF
245             def UNDEF_RULE(self, s: str): & Yannik
246                 pass
247
248             # UNDEF -> UNDEF
249             def UNDEF_RULE(self, s: str): & Yannik
250                 pass
251
252             # UNDEF -> UNDEF
253             def UNDEF_RULE(self, s: str): & Yannik
254                 pass
255
256             # UNDEF -> UNDEF
257             def UNDEF_RULE(self, s: str): & Yannik
```

```

    75      class Parser: __Yarikk
    76          # HEX_TAIL -> HEXDIGIT HEX_TAIL | e
    77          def HEX_TAIL_RULE(self, s: str): __Yarikk
    78              ch = self.c_peek(s)
    79              if ch is not None and (ch.isdigit() or ch.lower() in "abcdef"):
    80                  self.HEXDIGIT_RULE(s)
    81                  self.HEX_TAIL_RULE(s)
    82
    83          # HEXDIGIT -> DIGIT | a..f
    84          def HEXDIGIT_RULE(self, s: str): __Yarikk
    85              ch = self.c_peek(s)
    86              if ch is None:
    87                  self.error()
    88              if ch.isdigit() or ch.lower() in "abcdef":
    89                  self.c_pos += 1
    90              else:
    91                  self.error()
    92
    93          # DIGIT_TAIL -> DIGIT DIGIT_TAIL | e
    94          def DIGIT_TAIL_RULE(self, s: str): __Yarikk
    95              ch = self.c_peek(s)
    96              if ch is not None and ch.isdigit():
    97                  self.c_pos += 1
    98                  self.DIGIT_TAIL_RULE(s)
    99
   100
   101      def check_line(line: str) -> bool: __Yarikk
   102          lexer = Lexer(line)
   103          parser = Parser(lexer)
   104          try:
   105              parser.LIST()
   106              return parser.tok == EOP
   107          except ParseError:
   108              return False
   109
   110
   111      def main(): __Yarikk
   112          while True:
   113              try:
   114                  line = input("Enter your input line: ")
   115                  except EOFError:
   116                      break
   117
   118                  if line == "":
   119                      break # пустая строка -> завершение
   120
   121                  if check_line(line):
   122                      print("Accepted")
   123                  else:
   124                      print("Rejected")
   125
   126
   127      if __name__ == "__main__":
   128          main()

```

Рисунок 25 - Третья часть кода реализованной программы на python

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

```
Enter your input line: a=0
Accepted
Enter your input line: a=15
Accepted
Enter your input line: a=0b1011
Accepted
Enter your input line: a=0x1af
Accepted
Enter your input line: a=b&0b10
Accepted
Enter your input line: a=b&~0x1f|z
Accepted
Enter your input line: a=0;b=1;c=0x2
Accepted
Enter your input line: a=b=0
Accepted
Enter your input line: a=~(b|0x1)&c
Accepted
Enter your input line: bb=~~(a|10)
Accepted
Enter your input line: zx=0x123k
Rejected
Enter your input line: pl=$|61
Rejected
Enter your input line: a=0b0666
Rejected
Enter your input line: bb=~~~~
Rejected
```

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.