

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №5

Синтаксический анализ контекстно-свободных языков

тема

Преподаватель

подпись, дата

А. С. Кузнецов
инициалы, фамилия

Студент КИ18-17/16 031830504

номер группы, зачетной книжки

подпись, дата

Е.В. Железкин
инициалы, фамилия

Красноярск 2021

1 Цель работы

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

2 Задача работы

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.

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. Результатом работы синтаксического анализатора является выдача сообщения «Accepted» или «Rejected».

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. Полученная грамматика не должна повторять LL(1)-грамматику, конструируемую в части 1.

Вариант (1, 1, 1)

Часть 1: Язык оператора присваивания, в правой части которого задано арифметическое выражение. Элементами выражений являются целочисленные константы в двоичной системе счисления, имена переменных из одного символа (от а до f), знаки операций и скобки для изменения порядка вычисления подвыражений. Операции (в сторону уменьшения приоритета): унарный минус, мультипликативные, аддитивные, присваивание.

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

Часть 3: Элементами арифметического выражения являются целочисленные константы в 2- и 10-чной системах счисления, имена переменных из одного символа (от а до f), знаки операций и скобки для изменения порядка вычисления подвыражений. Операции (в сторону уменьшения приоритета): унарный минус, мультипликативные, аддитивные, присваивание.

2.1 Инструкция по запуску

Необходимо установить *python*, желательно версии 3 и выше (выполнено на версии 3.9.4):

- Страница загрузки для Windows: <https://www.python.org/downloads/>
- Для Linux есть несколько способов, один из них инструмент apt-get:

```
$ sudo apt-get update
```

```
$ sudo apt-get install python3.8
```
- Или загрузить, распаковать и установить образ:

```
$ wget https://www.python.org/ftp/python/3.8.2/Python-3.8.2.tgz
```

```
$ tar -xvf Python-3.8.2.tgz
```

Для следующего шага понадобится компилятор gcc, но, думаю, это не проблема. Переходим в распакованную папку и собираем+устанавливаем:

```
$ cd Python-3.8.2
```

```
$ ./configure
```

```
$ make
```

```
$ sudo make install
```

Далее на любой из двух систем перейти в каталог с распакованным архивом Lab_5 и выполнить:

```
$ python recursive_descent_parser.py <argv>
```

```
$ pytest RDP_tests.py
```

Реализация поддерживает как ввод тестовой строки как в качестве аргумента командной строки, так и непосредственно во входном потоке.

Для запуска тестов:

Установить библиотеку pytest:

```
$ pip install pytest
```

Запуск:

```
$ pytest RDP_tests.py
```

3 Ход работы

Часть 1

Реализована LL(1)-грамматика с помощью системы JFLAP:

JFLAP : (LL(1).jff)

File Input Test Convert Help

Editor

Table Text Size

LHS		RHS
S	→	aBC
S	→	bBC
S	→	cBC
S	→	dBC
S	→	eBC
S	→	fBC
B	→	=
C	→	{C}D
C	→	aD
C	→	bD
C	→	cD
C	→	dD
C	→	eD
C	→	fD
C	→	0FD
C	→	1FD
F	→	1F
F	→	0F
F	→	λ
C	→	-C
D	→	*C
D	→	/C
D	→	%C
D	→	λ
D	→	E
E	→	+C
E	→	-C

Рисунок 1 – полученная LL(1)-грамматика (файл LL(1)-1.jff)

Editor Build LL(1) Parse		Do Selected Do Step Do All Next Parse		Table Text Size	
S → aBC				Table Text Size	
S → bBC					
S → cBC					
S → dBC					
S → eBC					
S → fBC					
B → =					
C → {C}D					
C → aD					
C → bD					
C → cD					
C → dD					
C → eD					
C → fD					
C → 0FD					
C → 1FD					
F → 1F					
F → 0F					
F → λ					
C → -C					
D → *C					
D → /C					
D → %C					
D → λ					
D → E					
E → +C					
E → -C					

	FIRST	FOLLOW
B	{=}	{0, a, 1, b, c, d, e, f, {, -}
C	{0, a, 1, b, c, d, e, f, {, -}	{\$, }
D	{λ, %, *, +, -, /}	{\$, }
E	{+, -}	{\$, }
F	{0, λ, 1}	{\$, %, *, +, -, /}
S	{a, b, c, d, e, f}	{\$}

	%	*	+	-	/	0	1	=	a	b	c	d	e	f	{	}	\$
B				-C		0FD	1FD	=	aD	bD	cD	dD	eD	fD	{C}D		
C	%C	*C	E	E	/C											λ	λ
D			+C	-C													
E	λ	λ	λ	λ	λ	0F	1F		aBC	bBC	cBC	dBC	eBC	fBC		λ	λ
F																	
S																	

Рисунок 2 – таблица синтаксического LL(1)-анализа

Input	Result
a=a*b	Accept
b={a*1010}	Accept
c=a*1111111111111111*a	Accept
d={b*1}*c	Accept
e=a*{d*e}*f	Accept
f=a*{a}*a	Accept
a={a*-1010}	Accept
b=a*-1111111111111111*a	Accept
c={b*1}*c	Accept
d=1+{a-{b*c}/f}	Accept
e=a*-{d*e}*f	Accept
f=a*{{-a}}*a	Accept
a=a--b	Accept
reject_below	Reject
b=a**b	Reject
c=abc	Reject
d=a*-{de}*f	Reject
e=a*{-a}}*a	Reject
a+b	Reject

Рисунок 3 – Тестирование полученной грамматики

Editor
Build LL(1) Parse
LL(1) Parsing

Table Text Size

	%	*	+	-	/	0	1	=	a	b	c	d	e	f	{	}	\$
B																	
C				-C							cD			fD	{		
D		*	C	E	E	/C											
E				-C													
F						0F	1F										
S											c...			f...			

Start Step Derivation Table

Input
Input Remaining \$
Stack

Input Field Text Size (For optimization, move one of the window size

Table Text Size

LHS		RHS
S	→	aBC
S	→	bBC
S	→	cBC
S	→	dBC
S	→	eBC
S	→	fBC
B	→	=
C	→	{C}D
C	→	aD
C	→	bD
C	→	cD
C	→	dD
C	→	eD

Table Text Size

S	→	dBC
B	→	=
C	→	1FD
F	→	
D	→	E
E	→	+C
C	→	{C}D
C	→	aD
D	→	E
E	→	C
C	→	{C}D
C	→	bD
D	→	*C
C	→	cD
D	→	
D	→	/C
C	→	fD
D	→	
D	→	

String successfully parsed!

Рисунок 4 – Перехват экрана распознавания

EditorBuild LL(1) ParseLL(1) Parsing

Table Text Size

	%	*	+	-	/	0	1	=	a	b	c	d	e	f	{	}	\$
B								=									
C				-C							cD			fD	{		
D			*C	E	E	/C											
E					-C												
F						0F	1F										
S											c			f			

StartStepDerivation Table

Input

a={a*-1010}

Input Remaining \$

Stack

Input Field Text Size (For optimization, move one of the window size)

Table Text Size

LHS		RHS
S	→	aBC
S	→	bBC
S	→	cBC
S	→	dBC
S	→	eBC
S	→	fBC
B	→	=
C	→	{C}D
C	→	aD
C	→	bD
C	→	cD
C	→	dD
C	→	eD

Table Text Size

S	→	aBC
B	→	=
C	→	{C}D
C	→	aD
D	→	*C
C	→	-C
C	→	1FD
F	→	0F
F	→	1F
F	→	0F
F	→	λ
D	→	λ
D	→	λ

String successfully parsed!

Рисунок 5 – Перехват экрана распознавания

EditorBuild LL(1) ParseLL(1) Parsing

Table Text Size

	%	*	+	-	/	0	1	=	a	b	c	d	e	f	{	}	\$
B								=									
C				-C		cD	fD	{...		
D	...	*C	E	E	/C											λ	λ
E				-C													
F	λ	λ	λ	λ	λ	0F	1F									λ	λ
S									c...	f...			

StartStepDerivation Table

Input

a=a-b

Input Remaining \$

Stack

Input Field Text Size (For optimization, move one of t

Table Text Size

LHS		RHS
S	→	aBC
S	→	bBC
S	→	cBC
S	→	dBC
S	→	eBC
S	→	fBC
B	→	=
C	→	{C}D
C	→	aD
C	→	bD
C	→	cD
C	→	dD
C	→	eD

Table Text Size

S→aBC	S
B→=	aBC
C→aD	a=C
D→E	a=aD
E→C	a=aE
C→C	a=a-C
C→bD	a=a-C
D→λ	a=a-bD
	a=a-b

String successfully parsed!

Рисунок 6 – Перехват экрана распознавания

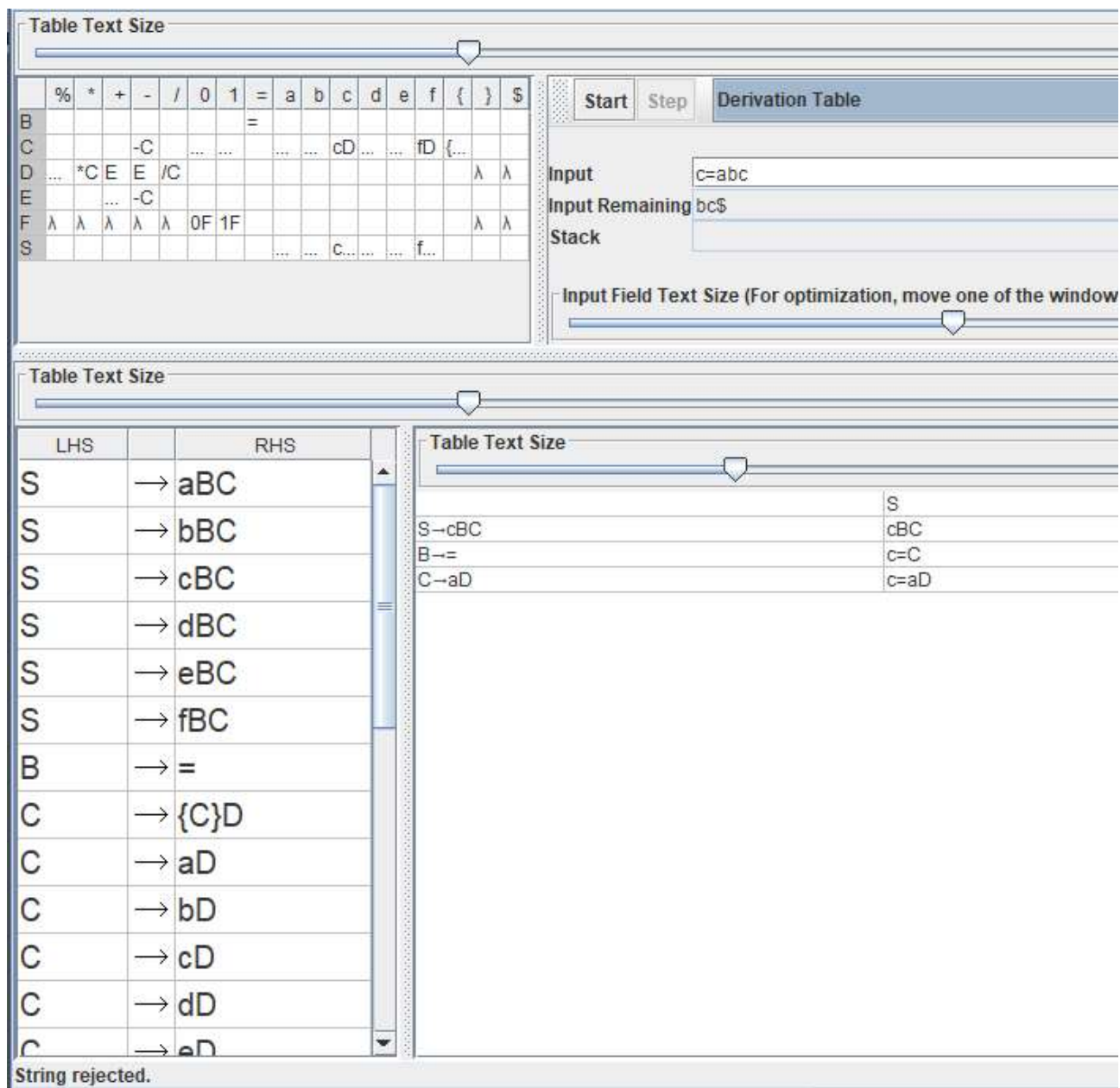


Рисунок 7 – Перехват экрана распознавания (для неверной строки)

Часть 2

LHS		F
S	→	aBCD;E
S	→	bBCD;E
S	→	cBCD;E
S	→	dBCD;E
S	→	eBCD;E
S	→	fBCD;E
B	→	a
B	→	b
B	→	c
B	→	d
B	→	e
B	→	f
B	→	λ
C	→	=
D	→	{D}F
D	→	aBF
D	→	bBF
D	→	cBF
D	→	dBF
D	→	eBF
D	→	fBF
D	→	'GIF
G	→	0
G	→	1
G	→	2
G	→	3
G	→	4
G	→	5
G	→	6

Рисунок 8 – полученная LL(1)-грамматика (файл LL(1)-2.jff; операция унарного минуса обозначена как «@»; идентификаторы систем счисления: «#» - 2, «'» - 8 «'» - 10)

Table Text Size		Table Text Size	
S	→ aBCD;E		
S	→ bBCD;E		
S	→ cBCD;E		
S	→ dBCD;E		
S	→ eBCD;E		
S	→ fBCD;E		
B	→ a		
B	→ b		
B	→ c		
B	→ d		
B	→ e		
B	→ f		
B	→ λ		
C	→ =		
D	→ {D}F		
D	→ aBF		
D	→ bBF		
D	→ cBF		
D	→ dBF		
D	→ eBF		
D	→ fBF		
D	→ 'GIF		
G	→ 0		
G	→ 1		
G	→ 2		
G	→ 3		
G	→ 4		

Table Text Size		Table Text Size	
B	→ a, b, c, d, e, f	FIRST	FOLLOW
C	→ {		
D	→ a, b, c, d, e, f, {		
E	→ {		
F	→ {		
G	→ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}		
H	→ {		
I	→ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}		
J	→ {0, 1, 2, 3, 4, 5, 6, 7}		
K	→ {0, 1, 2, 3, 4, 5, 6, 7}		
L	→ {0, 1}		
M	→ {0, 1}		
S	→ {a, b, c, d, e, f}		

Table Text Size		Table Text Size	
B	→ a, b, c, d, e, f	FIRST	FOLLOW
C	→ {		
D	→ a, b, c, d, e, f, {		
E	→ {		
F	→ {		
G	→ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}		
H	→ {		
I	→ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}		
J	→ {0, 1, 2, 3, 4, 5, 6, 7}		
K	→ {0, 1, 2, 3, 4, 5, 6, 7}		
L	→ {0, 1}		
M	→ {0, 1}		
S	→ {a, b, c, d, e, f}		

Рисунок 9 – Таблица СА

Грамматика, удовлетворяющая следующим правилам, считается LL(1):

- G - LL(1) тогда и только тогда, когда для каждой A -продукции грамматики ($A \rightarrow a_1 | \dots | a_n, n > 0$) выполняются следующие условия:
 - Множества $FIRST(a_1), \dots, FIRST(a_n)$ попарно не пересекаются
 - Если $a_i \Rightarrow^* \epsilon$, то $FIRST(a_j) \cap FOLLOW(A) = \emptyset$ для $1 \leq j \leq n, i \neq j$

Для каждого нетерминала A в грамматике генерируется множество терминалов $First(A)$, определенное следующим образом:

- если в грамматике есть правило с A в левой части и правой частью, начинающейся с терминала, то данный терминал входит в $First(A)$
- если в грамматике есть правило с A в левой части и правой частью, начинающейся с нетерминала (обозначим B), то $First(B)$ строго входит в $First(A)$
- никакие иные терминалы не входят в $First(A)$

Для каждого правила генерируется множество **направляющих символов**, определенное следующим образом:

- если правая часть правила начинается с терминала, то множество направляющих символов состоит из одного этого терминала
- иначе правая часть начинается с нетерминала A , тогда множество направляющих символов есть $First(A)$

Возможны обобщения этих определений для случая наличия правил вида $A \rightarrow \text{null}$.

Понятно, что $First(A)$ есть объединение множеств направляющих символов для всех правил с A в левой части.

Грамматика *разбираема по LL(1)*, если для любой пары правил с одинаковой левой частью множества направляющих символов не пересекаются.

Рисунок 10 – Правила, которым удовлетворяет LL(1)-грамматика

Для того, чтобы формально доказать, что описанная грамматика является LL(1)-грамматикой, обратимся к правилам, описанным выше:

- множества $FIRST(a_1), \dots, FIRST(a_n)$ попарно не пересекаются (каждая продукция начинается с уникального терминала);
- множества $FIRST$ и $FOLLOW$ не имеют пересечений для каждой продукции)

Формально грамматика является LL(1)-грамматикой.

The screenshot displays a software interface for parsing a string. The top part shows a 'Derivation Table' with 'Input' and 'Stack' fields. The bottom part shows a 'Table Text Size' window with a list of grammar rules (LHS and RHS) and a list of derived strings.

Derivation Table:

Start	Step	Derivation Table
Input		ab=cd-ef*(#1+7/9)-@@@3;a=a;
Input Remaining		S
Stack		

Table Text Size:

LHS	RHS
S	→ aBCD;E
S	→ bBCD;E
S	→ cBCD;E
S	→ dBCD;E
S	→ eBCD;E
S	→ fBCD;E
B	→ a
B	→ b
B	→ c
B	→ d
B	→ e
B	→ f
B	→ λ
C	→ =
D	→ {D}F
D	→ aBF
D	→ bBF
D	→ cBF
D	→ dBF
D	→ eBF

String successfully parsed!

Рисунок 11 – Перехват экрана распознавания

```
a=b+c;  
S  
aBCD;E  
aCD;E  
a=D;E  
a=bBF;E  
a=bF;E  
a=bH;E  
a=b+D;E  
a=b+cBF;E  
a=b+cF;E  
a=b+c;E  
a=b+c;  
Accepted!  
  
Process finished with exit code 0  
|
```

Рисунок 12 – Наглядное тестирование программной реализации

```
give  
Rejected!  
Unknown input symbol!  
  
Process finished with exit code 0
```

Рисунок 13 – Наглядное тестирование программной реализации


```
ab='987867564*bc-a;  
S  
aBCD;E  
abCD;E  
ab=D;E  
ab=-D;E  
ab=-'GIF;E  
ab=-'9IF;E  
ab=-'98IF;E  
ab=-'987IF;E  
ab=-'9878IF;E  
ab=-'98786IF;E  
ab=-'987867IF;E  
ab=-'9878675IF;E  
ab=-'98786756IF;E  
ab=-'987867564IF;E  
ab=-'987867564F;E  
ab=-'987867564*D;E  
ab=-'987867564*bBF;E  
ab=-'987867564*bcF;E  
ab=-'987867564*bCH;E  
ab=-'987867564*bc-D;E  
ab=-'987867564*bc-aBF;E  
ab=-'987867564*bc-aF;E  
ab=-'987867564*bc-a;E  
ab=-'987867564*bc-a;  
Accepted!  
  
Process finished with exit code 0  
|
```

Рисунок 14 – Наглядное тестирование программной реализации

```

d=a;b=b;c=c;
S
aBCD;E
aCD;E
a=D;E
a=aBF;E
a=aF;E
a=a;E
a=a;bBCD;E
a=a;bCD;E
a=a;b=D;E
a=a;b=bBF;E
a=a;b=bF;E
a=a;b=b;E
a=a;b=b;cBCD;E
a=a;b=b;cCD;E
a=a;b=b;c=D;E
a=a;b=b;c=cBF;E
a=a;b=b;c=cF;E
a=a;b=b;c=c;E
a=a;b=b;c=c;
Accepted!

Process finished with exit code 0

```

Рисунок 15 – Наглядное тестирование программной реализации

```

a;
S
aBCD;E
aCD;E
Rejected!
Wrong input!

Process finished with exit code 0

```

Рисунок 16 – Наглядное тестирование программной реализации


```
a='b;b=#1+"7+'9;  
S  
aBCD;E  
aCD;E  
a=D;E  
a=-D;E  
a=-bBF;E  
a=-bF;E  
a=-b;E  
a=-b;bBCD;E  
a=-b;bCD;E  
a=-b;b=D;E  
a=-b;b=#LMF;E  
a=-b;b=#1MF;E  
a=-b;b=#1F;E  
a=-b;b=#1H;E  
a=-b;b=#1+D;E  
a=-b;b=#1+"JKF;E  
a=-b;b=#1+"7KF;E  
a=-b;b=#1+"7F;E  
a=-b;b=#1+"7H;E  
a=-b;b=#1+"7+D;E  
a=-b;b=#1+"7+'GIF;E  
a=-b;b=#1+"7+'9IF;E  
a=-b;b=#1+"7+'9F;E  
a=-b;b=#1+"7+'9;E  
a=-b;b=#1+"7+'9;  
Accepted!  
  
Process finished with exit code 0  
|
```

Рисунок 18 – Наглядное тестирование программной реализации

```

a=#2;
S
aBCD;E
aCD;E
a=D;E
a=#LMF;E
Rejected!
Wrong input!

Process finished with exit code 0
|

```

Рисунок 19 – Наглядное тестирование программной реализации

```

a="S;
S
aBCD;E
aCD;E
a=D;E
a="JKF;E
Rejected!
Wrong input!

Process finished with exit code 0

```

Рисунок 20 – Наглядное тестирование программной реализации

```

a=#1;
S
aBCD;E
aCD;E
a=D;E
a=#LMF;E
a=#1MF;E
a=#1F;E
a=#1;E
a=#1;
Accepted!

Process finished with exit code 0

```

Рисунок 21 – Наглядное тестирование программной реализации

```

a="7;
S
aBCD;E
aCD;E
a=D;E
a="JKF;E
a="7KF;E
a="7F;E
a="7;E
a="7;
Accepted!

Process finished with exit code 0

```

Рисунок 22 – Наглядное тестирование программной реализации

```

D:\Dev\University\T-PofPLD\Lab_5\Alg's>pytest RDP_tests.py
===== test session starts =====
platform win32 -- Python 3.9.4, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: D:\Dev\University\T-PofPLD\Lab_5\Alg's
collected 10 items

RDP_tests.py ..... [100%]

===== 10 passed in 0.03s =====
D:\Dev\University\T-PofPLD\Lab_5\Alg's>_

```

Рисунок 23 – Тестирование программной реализации с помощью набора тестов

Для запуска тестов необходимо установить библиотеку *pytest* (`pip install pytest`) для python и выполнить `pytest CYK_tests.py`

Часть 3

LHS		RHS
S	→	L=R;Z
Z	→	L=R;Z
Z	→	λ
L	→	a
L	→	b
L	→	c
L	→	d
L	→	e
L	→	f
R	→	-R
R	→	{R}A
A	→	*R
A	→	/R
A	→	%R
A	→	B
B	→	+R
B	→	-R
B	→	λ
R	→	CA
C	→	a
C	→	b
C	→	c
C	→	d
C	→	e
C	→	f
C	→	'DE
C	→	#FG
D	→	0
D	→	1
D	→	2
D	→	3

Рисунок 24 – Полученная грамматика для SLR анализа (SLR.jff;
идентификаторы систем счисления: «#» - 2, «'» - 10)

Editor Build SLR(1) Parse SLR(1) Parsing

Table Text Size

	#	%	*	+	-	/	0	1	2	3	4	5	6	7	8	9	:	=	a	b
0																				
1																				
2																				
3																		r4		
4																		r5		
5																		r6		
6																		r7		
7																		r8		
8																		r9		
9																				
...																				
...																				
...																				
...																				
...																				
...																				
...																				

Start Step Derivation Table

Input $a = a + b * c / \{a\%a\} + \#1;$

Input Remaining \$

Stack S0

Input Field Text Size (For optimization, move one of the window size)

Table Text Size

LHS	RHS
S'	→ S
S	→ L=R;Z
Z	→ L=R;Z
Z	→ λ
L	→ a
L	→ b
L	→ c
L	→ d
L	→ e
L	→ f
R	→ -R
R	→ {R}A
A	→ *R
A	→ /R
A	→ %R
A	→ B
B	→ +R
B	→ -R
B	→ λ
R	→ CA

String accepted

LHS	RHS
L→a	$a = a + b * c / \{a\%a\} + \#1;$
C→a	$L = a + b * c / \{a\%a\} + \#1;$
C→b	$L = C + b * c / \{a\%a\} + \#1;$
C→c	$L = C + C * c / \{a\%a\} + \#1;$
C→a	$L = C + C * C / \{C\%a\} + \#1;$
C→a	$L = C + C * C / \{C\%C\} + \#1;$
B→λ	$L = C + C * C / \{C\%CB\} + \#1;$
A→B	$L = C + C * C / \{C\%CA\} + \#1;$
R→CA	$L = C + C * C / \{C\%R\} + \#1;$
A→%R	$L = C + C * C / \{CA\} + \#1;$
R→CA	$L = C + C * C / \{R\} + \#1;$
F→1	$L = C + C * C / \{R\} + \#F;$
G→λ	$L = C + C * C / \{R\} + \#FG;$
C→#FG	$L = C + C * C / \{R\} + C;$
B→λ	$L = C + C * C / \{R\} + CB;$
A→B	$L = C + C * C / \{R\} + CA;$
R→CA	$L = C + C * C / \{R\} + R;$
B→+R	$L = C + C * C / \{R\}B;$
A→B	$L = C + C * C / \{R\}A;$
R→{R}A	$L = C + C * C / R;$
A→/R	$L = C + C * CA;$
R→CA	$L = C + C * R;$
A→*R	$L = C + CA;$
R→CA	$L = C + R;$
B→+R	$L = CB;$
A→B	$L = CA;$
R→CA	$L = R;$
Z→λ	$L = R;Z$
S→L=R;Z	S

Рисунок 26 – Перехват экрана распознавания

Table Text Size

% ' * + - / 0 1 2 3 4 5 6 7 8 9 ; = a b

0

1

2

3

4

5

6

7

8

9

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

Start Step Derivation Table

Input

Input Remaining \$

Stack

a=a+#101010/{9};

S0

Input Field Text Size (For optimization, move one of the window

Table Text Size

LHS		RHS
S'	→	S
S	→	L=R;Z
Z	→	L=R;Z
Z	→	λ
L	→	a
L	→	b
L	→	c
L	→	d
L	→	e
L	→	f
R	→	-R
R	→	{R}A
A	→	*R
A	→	/R
A	→	%R
A	→	B
B	→	+R
B	→	-R
B	→	λ
R	→	CA

Table Text Size

L→a	a=a+#101010/{9};
C→a	L=a+#101010/{9};
F→1	L=C+#101010/{9};
G→λ	L=C+#F01010G/{9};
G→0G	L=C+#F0101G/{9};
G→1G	L=C+#F010G/{9};
G→0G	L=C+#F01G/{9};
G→1G	L=C+#F0G/{9};
G→0G	L=C+#FG/{9};
C→#FG	L=C+C/{9};
D→9	L=C+C/{D};
E→λ	L=C+C/{DE};
C→DE	L=C+C/{C};
B→λ	L=C+C/{CB};
A→B	L=C+C/{CA};
R→CA	L=C+C/{R};
B→λ	L=C+C/{R}B;
A→B	L=C+C/{R}A;
R→{R}A	L=C+C/R;
A→/R	L=C+CA;
R→CA	L=C+R;
B→+R	L=CB;
A→B	L=CA;
R→CA	L=R;
Z→λ	L=R;Z
S→L=R;Z	S

String accepted

Рисунок 27 – Перехват экрана распознавания

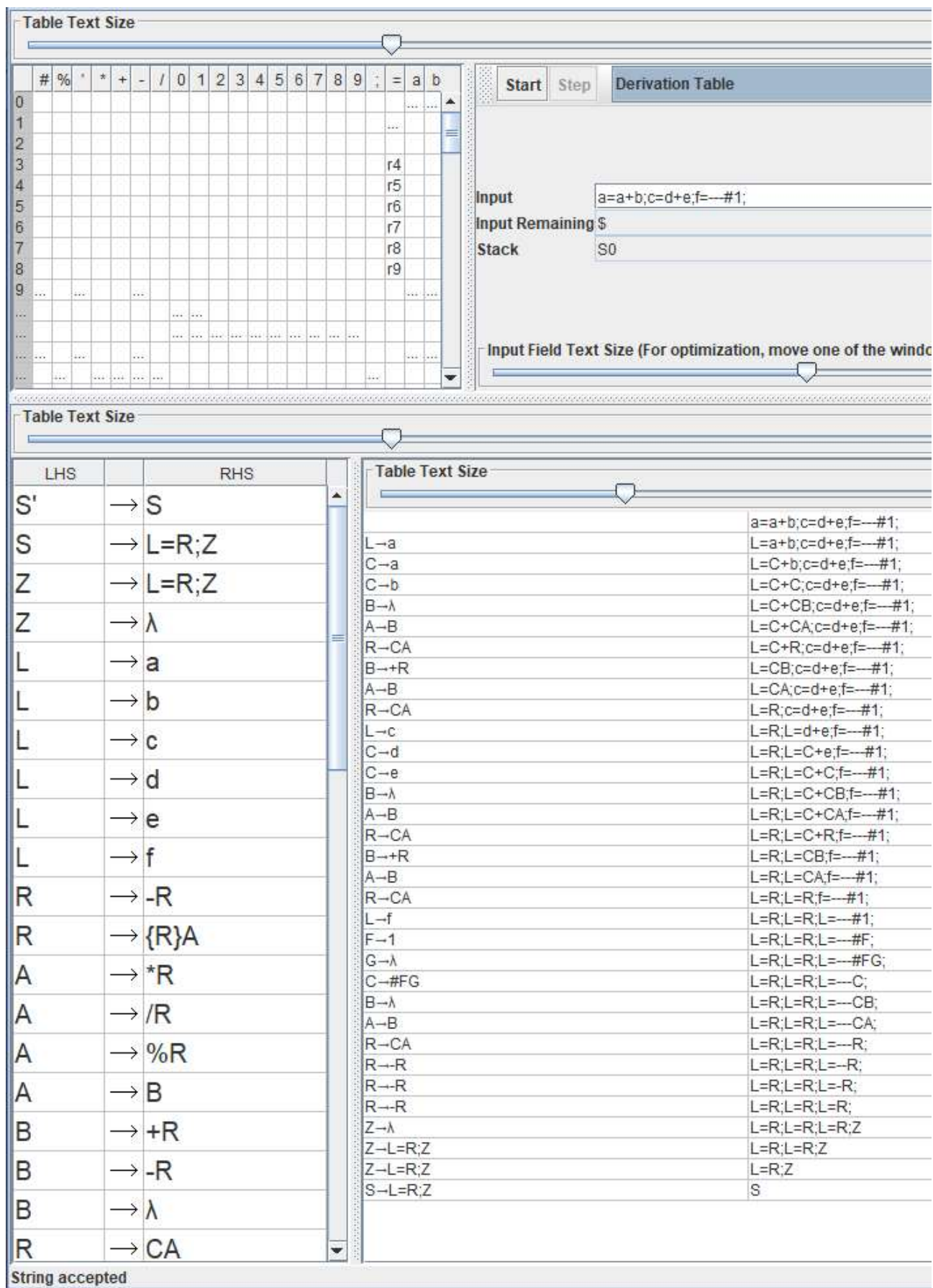


Рисунок 28 – Перехват экрана распознавания

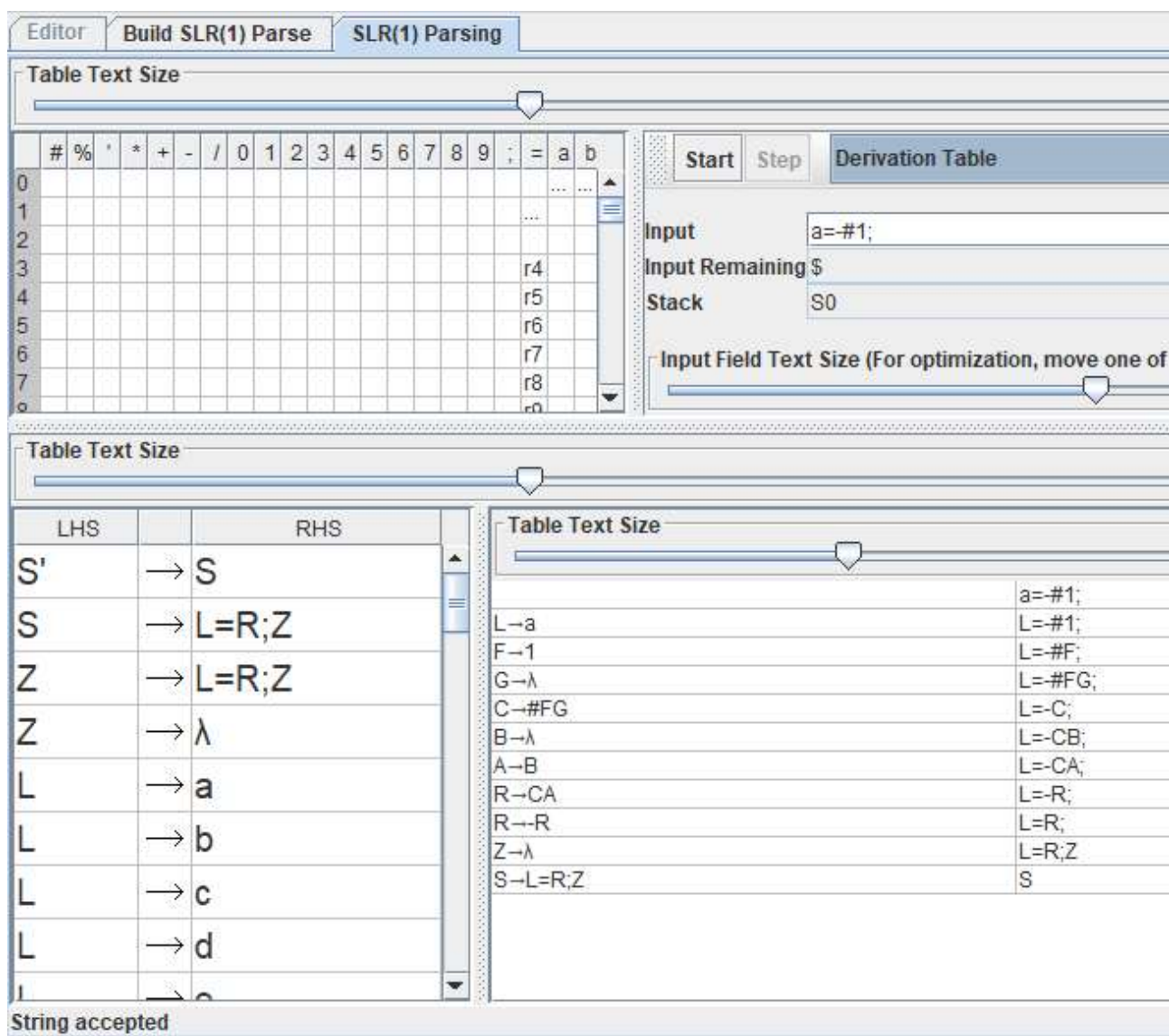


Рисунок 29 – Перехват экрана распознавания

4 Вывод

В ходе данной лабораторной работы были исследованы свойства универсальных алгоритмов синтаксического анализа контекстно-свободных языков.

ПРИЛОЖЕНИЕ А

Листинг 1 – файл recursive_descent_parser.py

```
import sys

EOI = 0
NUM2 = 1.2
BIN_DET = '#'
NUM8 = 1.8
OCT_DET = '''
NUM10 = 1.10
DEC_DET = '\''
VAR = 2
NEGATIVE = 3
ADDITIVE = 4
MULTI = 5
EQUAL = 6
LB = 20
RB = 21
SEPARATOR = 13
UNKNOWN = -1

TOKEN = 0
LEXEME = 1

SUCCESS = 0
ERROR = -1

MATCHING_DICT = {
    NUM2: ['#', '0', '1'],
    NUM8: ['"', '0', '1', '2', '3', '4',
           '5', '6', '7'],
    NUM10: ['\\', '0', '1', '2', '3', '4',
            '5', '6', '7', '8', '9'],
    VAR: ['a', 'b', 'c', 'd', 'e', 'f'],
    NEGATIVE: ['!'],
    ADDITIVE: ['+', '-'],
    MULTI: ['*', '/', '%'],
    LB: ['('],
    RB: [')'],
    EQUAL: ['='],
    SEPARATOR: [';'],
    EOI: ['$']
}

class RDParser:
    input_index = 0
    str_for_parse = ''
    stack = 'S'

    @staticmethod
    def get_next_token():
        temp_token = EOI

        if RDParser.str_for_parse[RDParser.input_index] in MATCHING_DICT[NUM2]:
            temp_token = NUM2
        elif RDParser.str_for_parse[RDParser.input_index] in
```

```

MATCHING_DICT[NUM8]:
    temp_token = NUM8
    elif RDParse.str_for_parse[RDParser.input_index] in
MATCHING_DICT[NUM10]:
    temp_token = NUM10
    elif RDParse.str_for_parse[RDParser.input_index] in MATCHING_DICT[VAR]:
    temp_token = VAR
    elif RDParse.str_for_parse[RDParser.input_index] in MATCHING_DICT[LB]:
    temp_token = LB
    elif RDParse.str_for_parse[RDParser.input_index] in MATCHING_DICT[RB]:
    temp_token = RB
    elif RDParse.str_for_parse[RDParser.input_index] in
MATCHING_DICT[NEGATIVE]:
    temp_token = NEGATIVE
    elif RDParse.str_for_parse[RDParser.input_index] in
MATCHING_DICT[MULTI]:
    temp_token = MULTI
    elif RDParse.str_for_parse[RDParser.input_index] in
MATCHING_DICT[ADDITIVE]:
    temp_token = ADDITIVE
    elif RDParse.str_for_parse[RDParser.input_index] in
MATCHING_DICT[EQUAL]:
    temp_token = EQUAL
    elif RDParse.str_for_parse[RDParser.input_index] in
MATCHING_DICT[SEPARATOR]:
    temp_token = SEPARATOR
    elif len(RDParse.str_for_parse) > RDParser.input_index + 1:
    temp_token = UNKNOWN

    if temp_token == UNKNOWN:
        RDParser.raise_error(Exception('Unknown input symbol!'))

    if temp_token != EOI:
        RDParser.input_index += 1
        return temp_token, RDParser.str_for_parse[RDParser.input_index - 1]
    return temp_token, ''

@staticmethod
def token_rollback():
    RDParser.input_index -= 1

@staticmethod
def raise_error(exc=Exception('Wrong input!')):
    print('Rejected!')
    raise exc

@staticmethod
def parse(str_for_parse):

    RDParser.input_index = 0
    RDParser.stack = 'S'
    RDParser.str_for_parse = str_for_parse + '$'

    if RDParser.start() == 0 and len(str_for_parse) == RDParser.input_index
+ 1:
        print('Accepted!')
        return True
    else:
        print('Rejected!')
        return False

@staticmethod

```

```

def print_stack():
    print(RDParser.stack)

@staticmethod
def stack_update(func_name, production):
    RDParser.stack = RDParser.stack.replace(func_name, production, 1)

@staticmethod
def start():
    res = SUCCESS
    token = RDParser.get_next_token()
    RDParser.print_stack()

    if token[TOKEN] == VAR:
        RDParser.stack_update('S', token[LEXEME] + 'BCD;E')
        RDParser.print_stack()
        res += RDParser.b_func()
        res += RDParser.c_func()
        res += RDParser.d_func()
        if RDParser.get_next_token()[TOKEN] != SEPARATOR:
            RDParser.raise_error()
            # res += ERROR
        res += RDParser.e_func()
    else:
        RDParser.raise_error()
        # res += ERROR
    return res

@staticmethod
def b_func():
    token = RDParser.get_next_token()
    if token[TOKEN] == VAR:
        RDParser.stack_update('B', token[LEXEME])
        RDParser.print_stack()
        return SUCCESS
    else:
        RDParser.stack_update('B', '')
        RDParser.print_stack()
        RDParser.token_rollback()
        return SUCCESS

@staticmethod
def c_func():
    token = RDParser.get_next_token()
    if token[TOKEN] == EQUAL:
        RDParser.stack_update('C', '=')
        RDParser.print_stack()
        return SUCCESS
    else:
        RDParser.raise_error()
        # return ERROR

@staticmethod
def d_func():
    res = SUCCESS
    token = RDParser.get_next_token()
    if token[TOKEN] == LB:
        RDParser.stack_update('D', '{D}F')
        RDParser.print_stack()
        res += RDParser.d_func()
        if RDParser.get_next_token()[TOKEN] != RB:
            RDParser.raise_error()

```

```

        # res += ERROR
        res += RDParser.f_func()
    elif token[TOKEN] == VAR:
        RDParser.stack_update('D', token[LEXEME] + 'BF')
        RDParser.print_stack()
        res += RDParser.b_func()
        res += RDParser.f_func()
    elif token[TOKEN] == NUM2 and token[LEXEME] == BIN_DET:
        RDParser.stack_update('D', '#LMF')
        RDParser.print_stack()
        res += RDParser.l_func()
        res += RDParser.m_func()
        res += RDParser.f_func()
    elif token[TOKEN] == NUM8 and token[LEXEME] == OCT_DET:
        RDParser.stack_update('D', '"JKF')
        RDParser.print_stack()
        res += RDParser.j_func()
        res += RDParser.k_func()
        res += RDParser.f_func()
    elif token[TOKEN] == NUM10 and token[LEXEME] == DEC_DET:
        RDParser.stack_update('D', '\GIF')
        RDParser.print_stack()
        res += RDParser.g_func()
        res += RDParser.i_func()
        res += RDParser.f_func()
    elif token[TOKEN] == NEGATIVE:
        RDParser.stack_update('D', '-D')
        RDParser.print_stack()
        res += RDParser.d_func()
    else:
        RDParser.raise_error()
        # res += ERROR
    return res

@staticmethod
def e_func():
    res = SUCCESS
    token = RDParser.get_next_token()
    if token[TOKEN] == VAR:
        RDParser.stack_update('E', token[LEXEME] + 'BCD;E')
        RDParser.print_stack()
        res += RDParser.b_func()
        res += RDParser.c_func()
        res += RDParser.d_func()
        if RDParser.get_next_token()[TOKEN] != SEPARATOR:
            RDParser.raise_error()
            # res += ERROR
        res += RDParser.e_func()
    else:
        RDParser.stack_update('E', '')
        RDParser.print_stack()
        RDParser.token_rollback()
    return res

@staticmethod
def f_func():
    res = SUCCESS
    token = RDParser.get_next_token()
    if token[TOKEN] == MULTI:
        RDParser.stack_update('F', token[LEXEME] + 'D')
        RDParser.print_stack()
        res += RDParser.d_func()

```

```

elif token[TOKEN] == ADDITIVE:
    RDParser.stack_update('F', 'H')
    RDParser.print_stack()
    RDParser.token_rollback()
    res += RDParser.h_func()
else:
    RDParser.stack_update('F', '')
    RDParser.print_stack()
    RDParser.token_rollback()
return res

@staticmethod
def g_func():
    res = SUCCESS
    token = RDParser.get_next_token()
    if token[LEXEME] in MATCHING_DICT[NUM10]:
        RDParser.stack_update('G', token[LEXEME])
        RDParser.print_stack()
    else:
        RDParser.raise_error()
    return res

@staticmethod
def h_func():
    res = SUCCESS
    token = RDParser.get_next_token()
    if token[TOKEN] == ADDITIVE:
        RDParser.stack_update('H', token[LEXEME] + 'D')
        RDParser.print_stack()
        res += RDParser.d_func()
    else:
        RDParser.raise_error()
    return res

@staticmethod
def i_func():
    res = SUCCESS
    token = RDParser.get_next_token()
    if token[LEXEME] in MATCHING_DICT[NUM10]:
        RDParser.stack_update('I', token[LEXEME] + 'I')
        RDParser.print_stack()
        res += RDParser.i_func()
    else:
        RDParser.stack_update('I', '')
        RDParser.print_stack()
        RDParser.token_rollback()
    return res

@staticmethod
def j_func():
    res = SUCCESS
    token = RDParser.get_next_token()
    if token[LEXEME] in MATCHING_DICT[NUM8]:
        RDParser.stack_update('J', token[LEXEME])
        RDParser.print_stack()
    else:
        RDParser.raise_error()
    return res

@staticmethod
def k_func():
    res = SUCCESS

```

```

token = RDParser.get_next_token()
if token[LEXEME] in MATCHING_DICT[NUM8]:
    RDParser.stack_update('K', token[LEXEME] + 'K')
    RDParser.print_stack()
    res += RDParser.k_func()
else:
    RDParser.stack_update('K', '')
    RDParser.print_stack()
    RDParser.token_rollback()
return res

@staticmethod
def l_func():
    res = SUCCESS
    token = RDParser.get_next_token()
    if token[LEXEME] in MATCHING_DICT[NUM2]:
        RDParser.stack_update('L', token[LEXEME])
        RDParser.print_stack()
    else:
        RDParser.raise_error()
    return res

@staticmethod
def m_func():
    res = SUCCESS
    token = RDParser.get_next_token()
    if token[LEXEME] in MATCHING_DICT[NUM2]:
        RDParser.stack_update('M', token[LEXEME] + 'M')
        RDParser.print_stack()
        res += RDParser.m_func()
    else:
        RDParser.stack_update('M', '')
        RDParser.print_stack()
        RDParser.token_rollback()
    return res

def main():
    if len(sys.argv) > 1:
        try:
            RDParser.parse(sys.argv[1])
        except Exception as e:
            print(e)
    else:
        temp = input()
        try:
            RDParser.parse(temp)
        except Exception as e:
            print(e)

if __name__ == "__main__":
    main()

```

Листинг 2 – файл RDP_tests.py

```

from recursive_descent_parser import RDParser

def test_my_grammar_1():

```



```

cur_word = "a=a*b;"

t = False

try:
    t = RDParser.parse(cur_word)
except Exception as e:
    _ = e

assert t

def test_my_grammar_2():
    cur_word = "a=b*('987654/cd);"

    t = False

    try:
        t = RDParser.parse(cur_word)
    except Exception as e:
        _ = e

    assert t

def test_my_grammar_3():
    cur_word = "a=b*(\"1/cd\")%#1010;"

    t = False

    try:
        t = RDParser.parse(cur_word)
    except Exception as e:
        _ = e

    assert t

def test_my_grammar_4():
    cur_word = "a=b;"

    t = False

    try:
        t = RDParser.parse(cur_word)
    except Exception as e:
        _ = e

    assert t

def test_my_grammar_5():
    cur_word = "a=a*!b;b=c+d;"

    t = False

    try:
        t = RDParser.parse(cur_word)
    except Exception as e:
        _ = e

    assert t

```

```

def test_my_grammar_6():
    cur_word = "a=a;b=b;c=c;d=!!!d;e=#101010100101010;"

    t = False

    try:
        t = RDParser.parse(cur_word)
    except Exception as e:
        _ = e

    assert t

def test_my_grammar_7():
    cur_word = "bc=ac+(af*('1010/b)+'111)-\"10;"

    t = False

    try:
        t = RDParser.parse(cur_word)
    except Exception as e:
        _ = e

    assert t

def test_my_grammar_8():
    cur_word = "a=a*b"

    t = False

    try:
        t = RDParser.parse(cur_word)
    except Exception as e:
        _ = e

    assert not t

def test_my_grammar_9():
    cur_word = "a=abc"

    t = False

    try:
        t = RDParser.parse(cur_word)
    except Exception as e:
        _ = e

    assert not t

def test_my_grammar_10():
    cur_word = "a=#9;"

    t = False

    try:
        t = RDParser.parse(cur_word)
    except Exception as e:

```

```
    _ = e  
    assert not t
```