Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

**ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №4**

Синтаксический анализ контекстно-свободных языков

тема

Преподаватель \_\_\_\_\_\_\_\_\_ А. С. Кузнецов

подпись, дата инициалы, фамилия

Студент КИ18-17/1б 031830504 \_\_\_\_\_\_\_\_\_ Е.В. Железкин

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2021

# Цель работы

Исследование свойств универсальных алгоритмов синтаксического анализа контекстно-свободных языков.

# Задача работы

Необходимо с использованием системы JFLAP, построить грамматику, определяющую заданный язык для анализа его методом Кока-Янгера-Касами, или формально доказать невозможность этого.

*Вариант 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 a до f), знаки операций и скобки для изменения порядка вычисления подвыражений. Операции (в сторону уменьшения приоритета): унарный минус, мультипликативные, аддитивные, присваивание.

## Инструкция по запуску

Необходимо установить *python*, желательно версии 3 и выше (выполнено на версии 3.8.2):

* 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*](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\_4 и выполнить:

*$ python PyPDA/main.py*

*($ python main.py*; если из папки *PyPDA)*

Ввести тестовую цепочку, нажать «ввод»

Для запуска тестов:

Установить библиотеку pytest:

*$ pip install pytest*

Запуск:

*$ pytest CYK\_tests.py*

# Ход работы

Был реализован МПА (согласно заданию первого варианта), допускающий входную цепочку по заключительному состоянию:

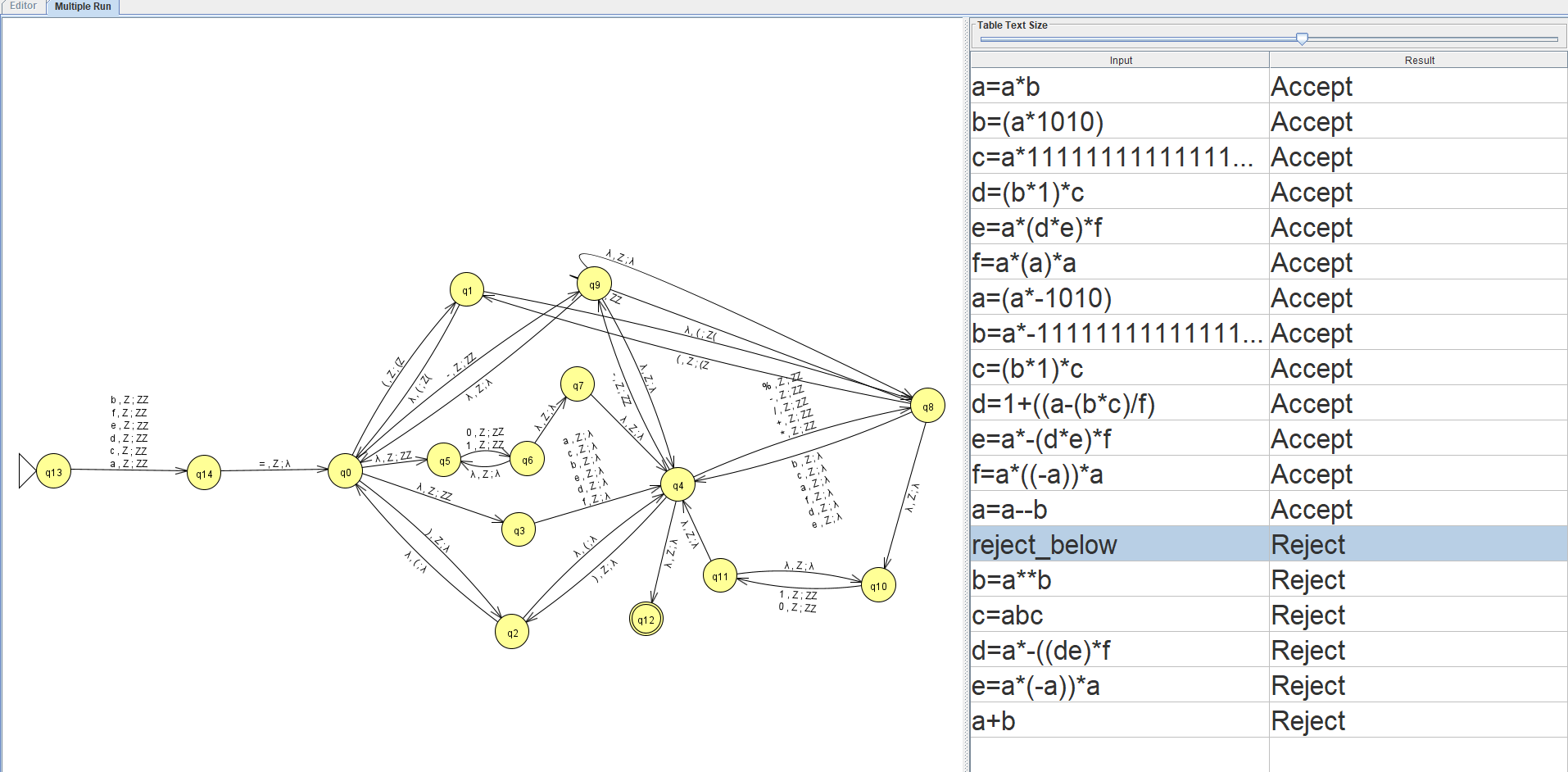


Рисунок 1 – Полученный МПА (PDA.jff)

Предполагалось конвертировать МПА в грамматику и привести её к НФХ, но какое-то ограничение системы JFLAP не позволило этого сделать (предположительно недостаточно нетерминалов латинского алфавита для конвертации).

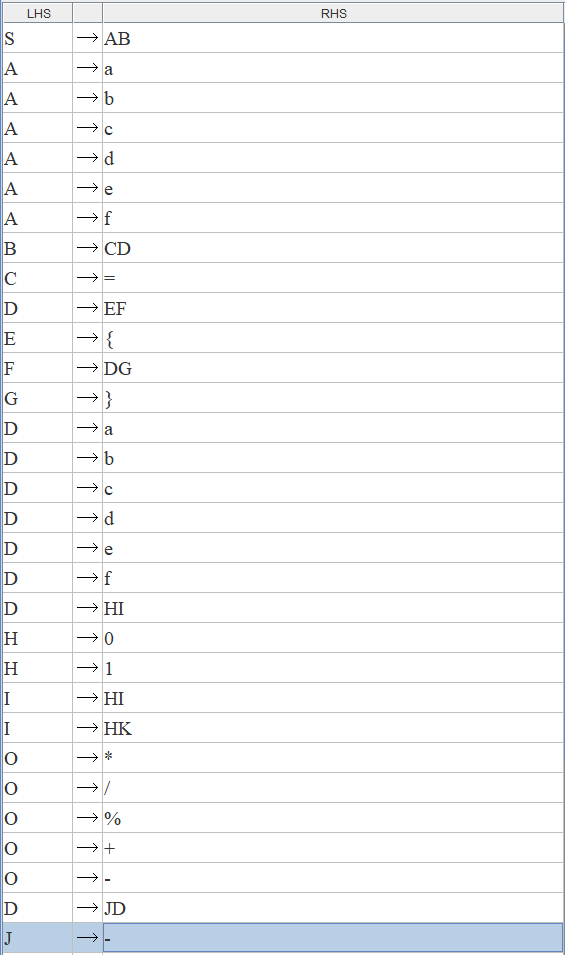


Рисунок 2 – Грамматика в НФХ, полученная вручную (файл grammar.jff)

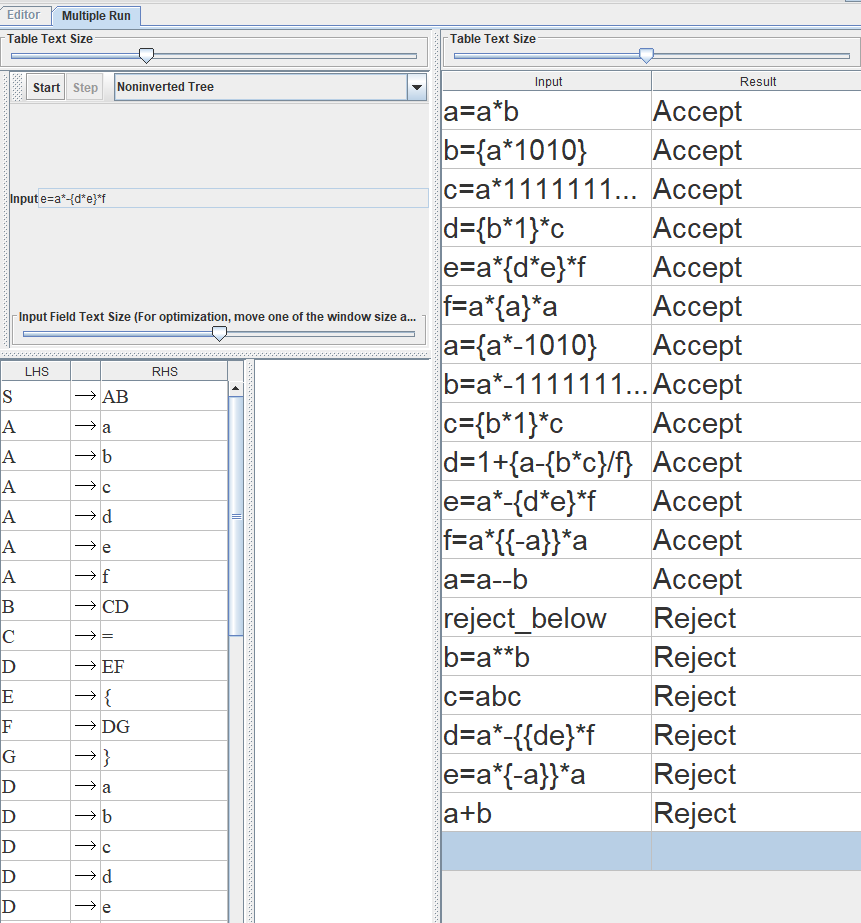


Рисунок 3 – Тестирование полученной грамматики с помощью алгоритма Кока-Янгера-Касами

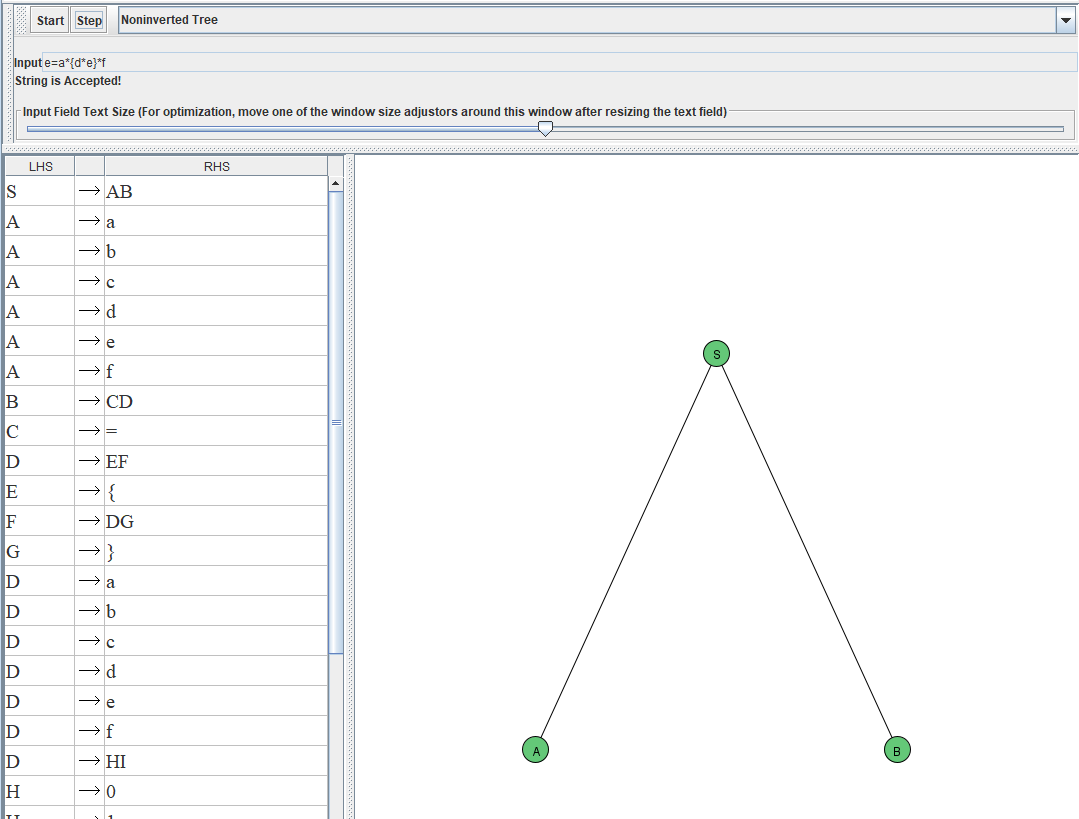


Рисунок 4 – Перехват экрана тестовых цепочек методом CYK

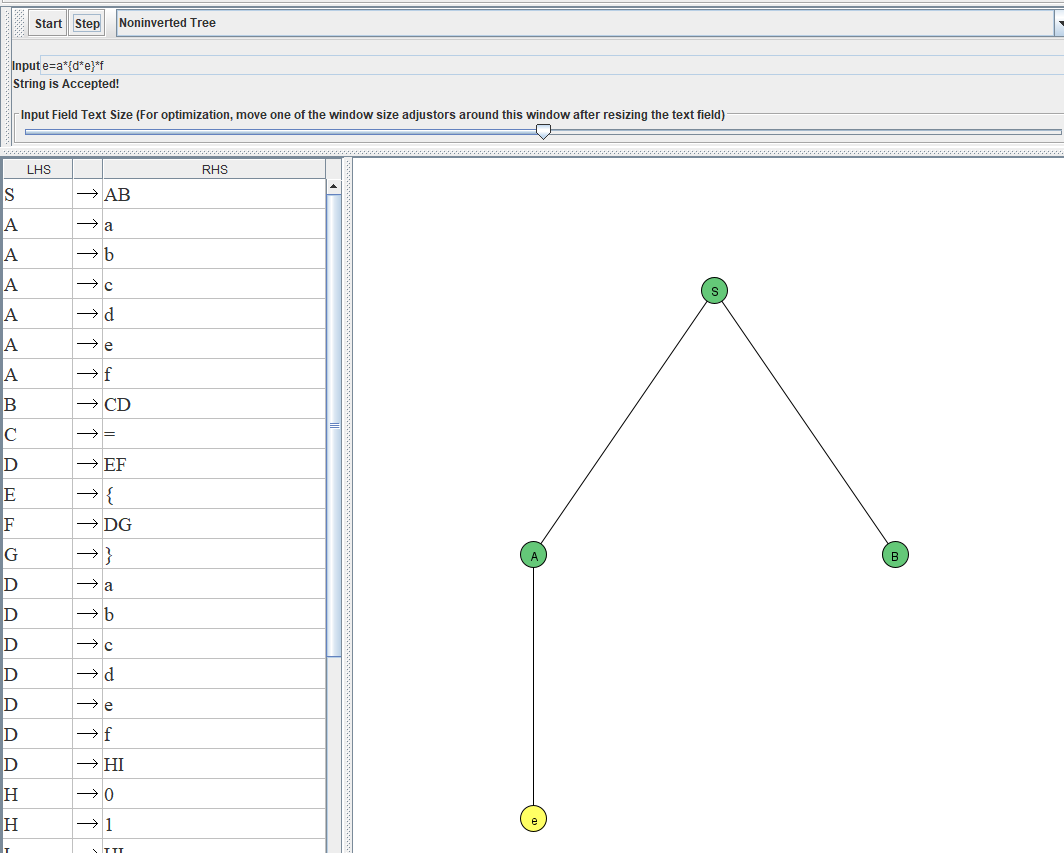


Рисунок 5 – Перехват экрана тестовых цепочек методом CYK

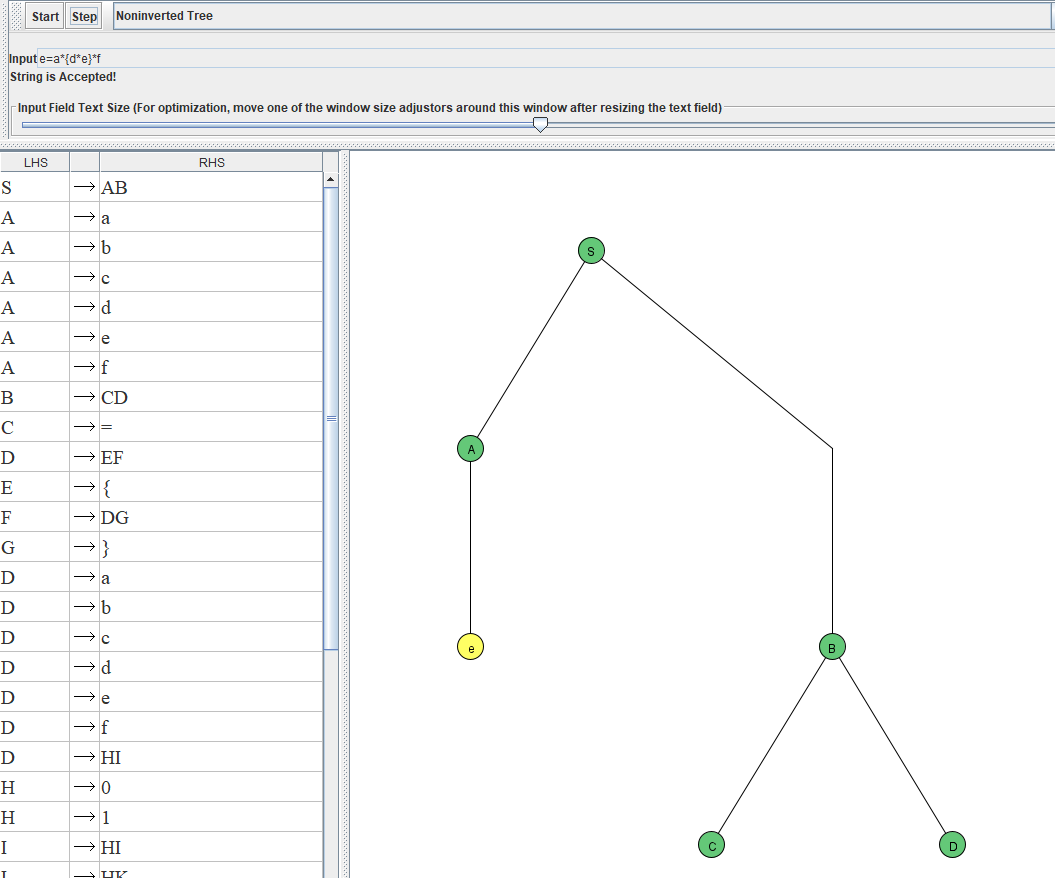


Рисунок 6 – Перехват экрана тестовых цепочек методом CYK

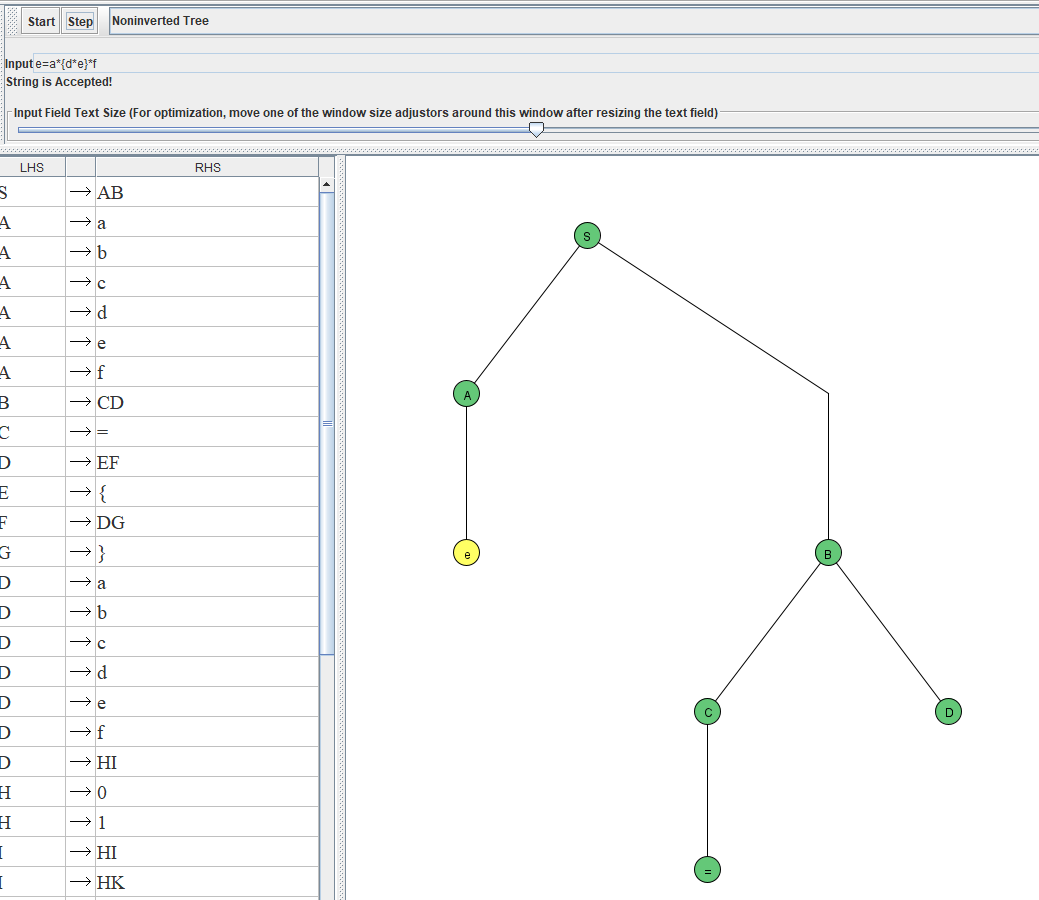


Рисунок 7 – Перехват экрана тестовых цепочек методом CYK

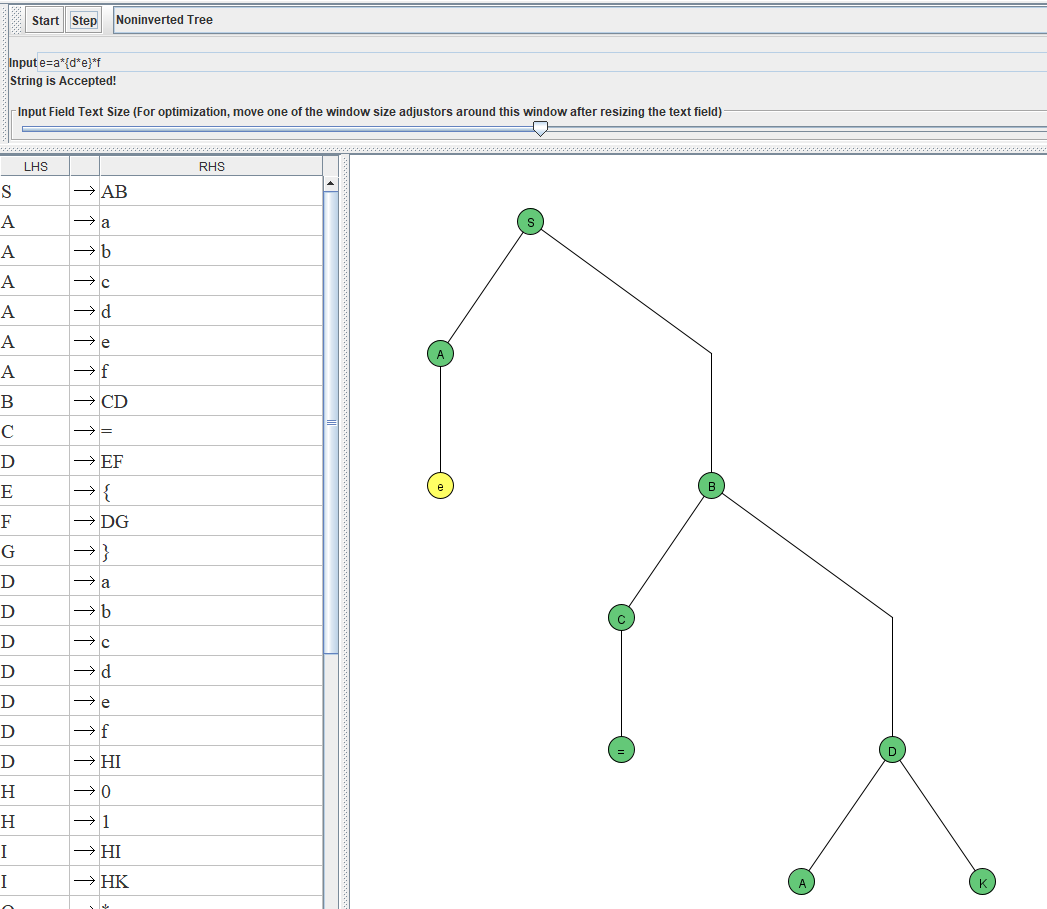


Рисунок 8 – Перехват экрана тестовых цепочек методом CYK

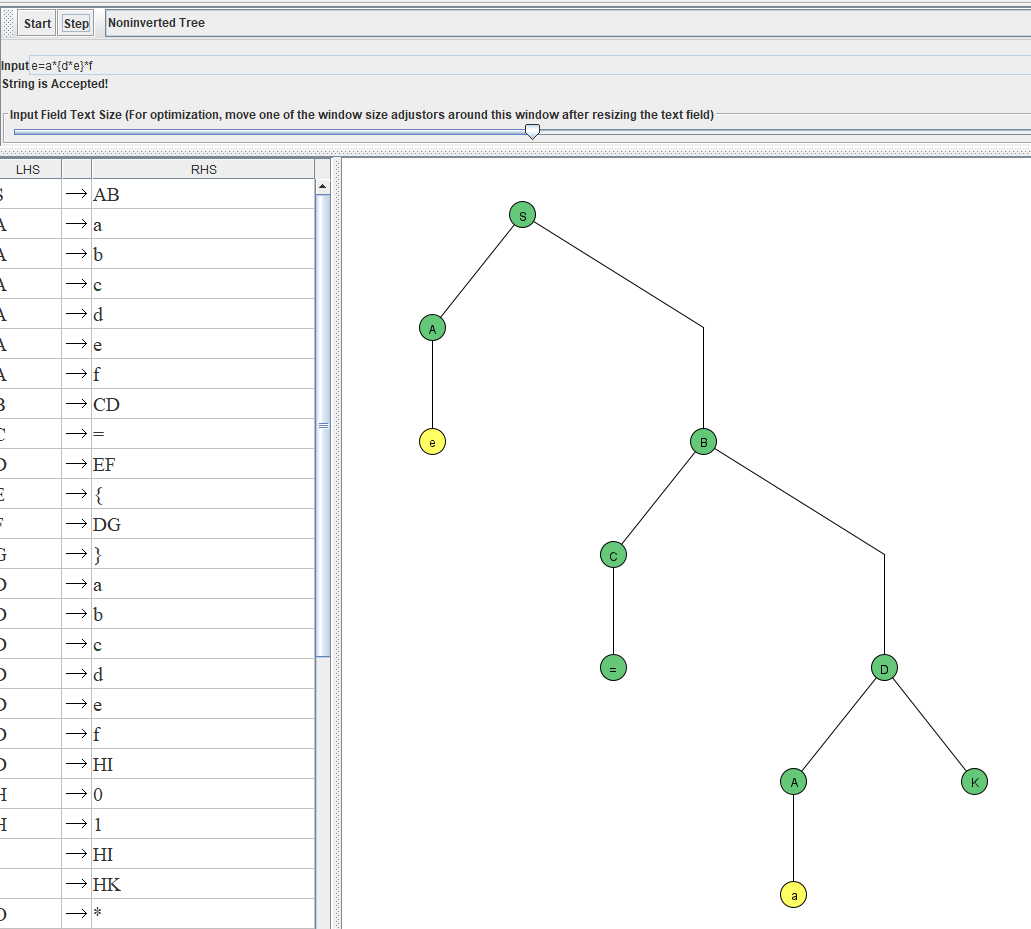


Рисунок 9 – Перехват экрана тестовых цепочек методом CYK

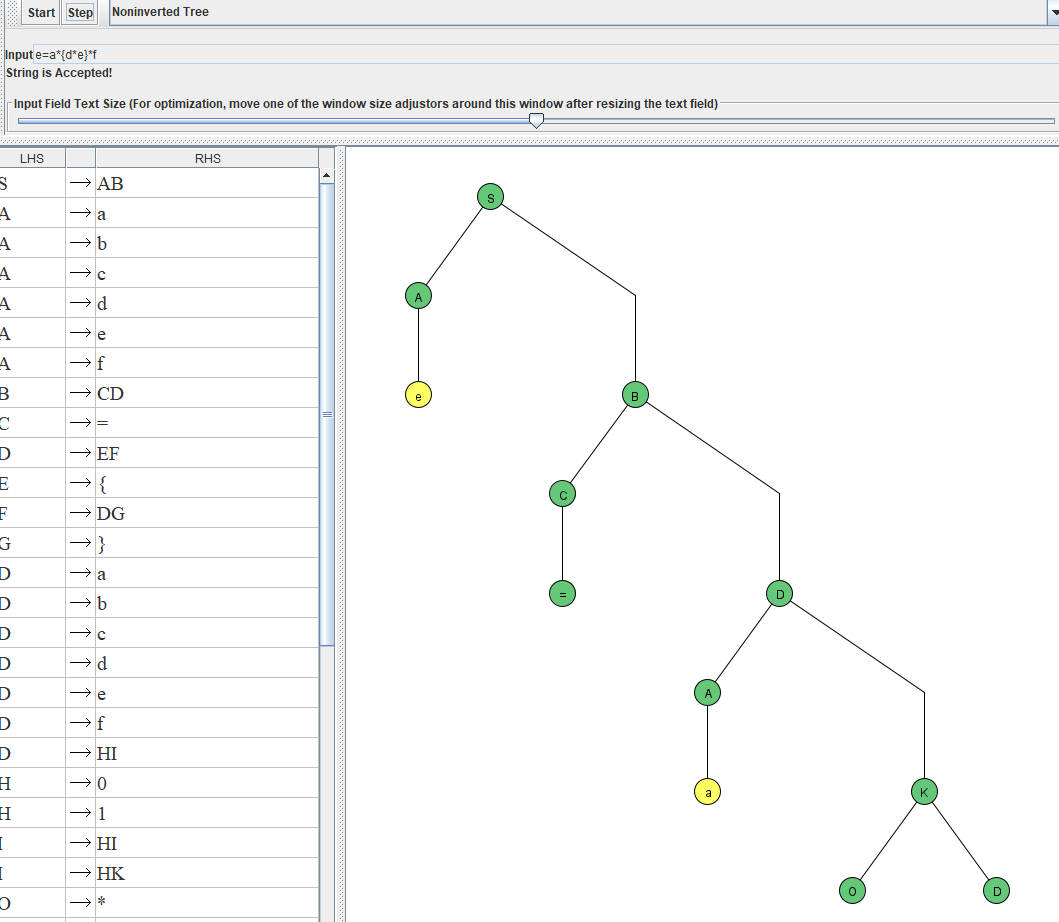


Рисунок 10 – Перехват экрана тестовых цепочек методом CYK

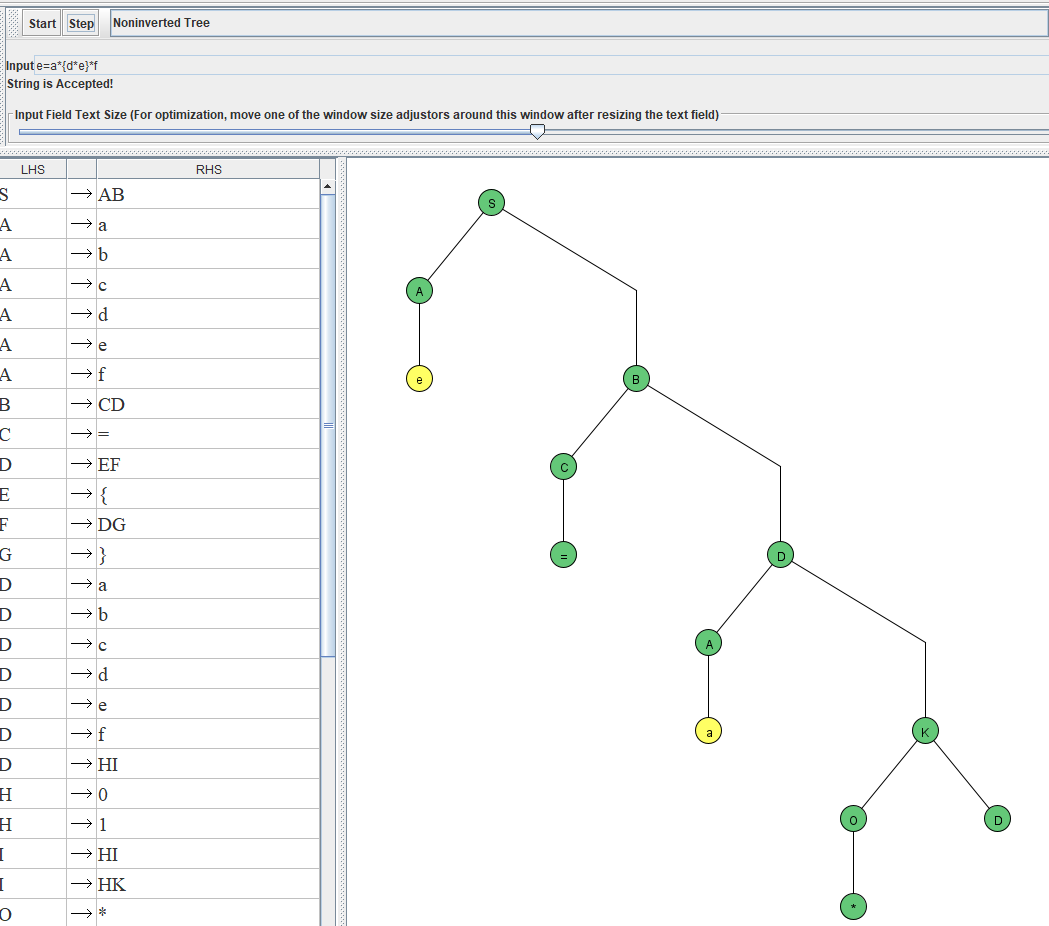


Рисунок 11 – Перехват экрана тестовых цепочек методом CYK

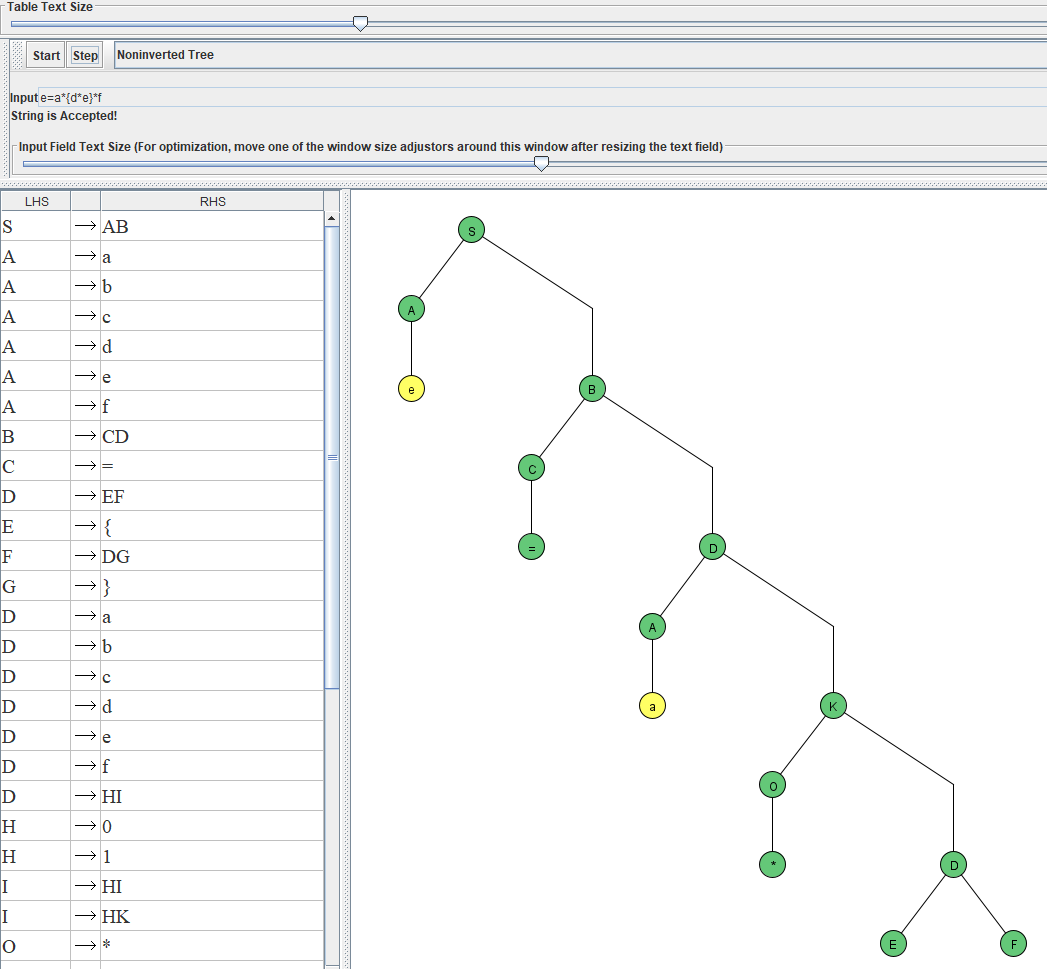


Рисунок 12 – Перехват экрана тестовых цепочек методом CYK

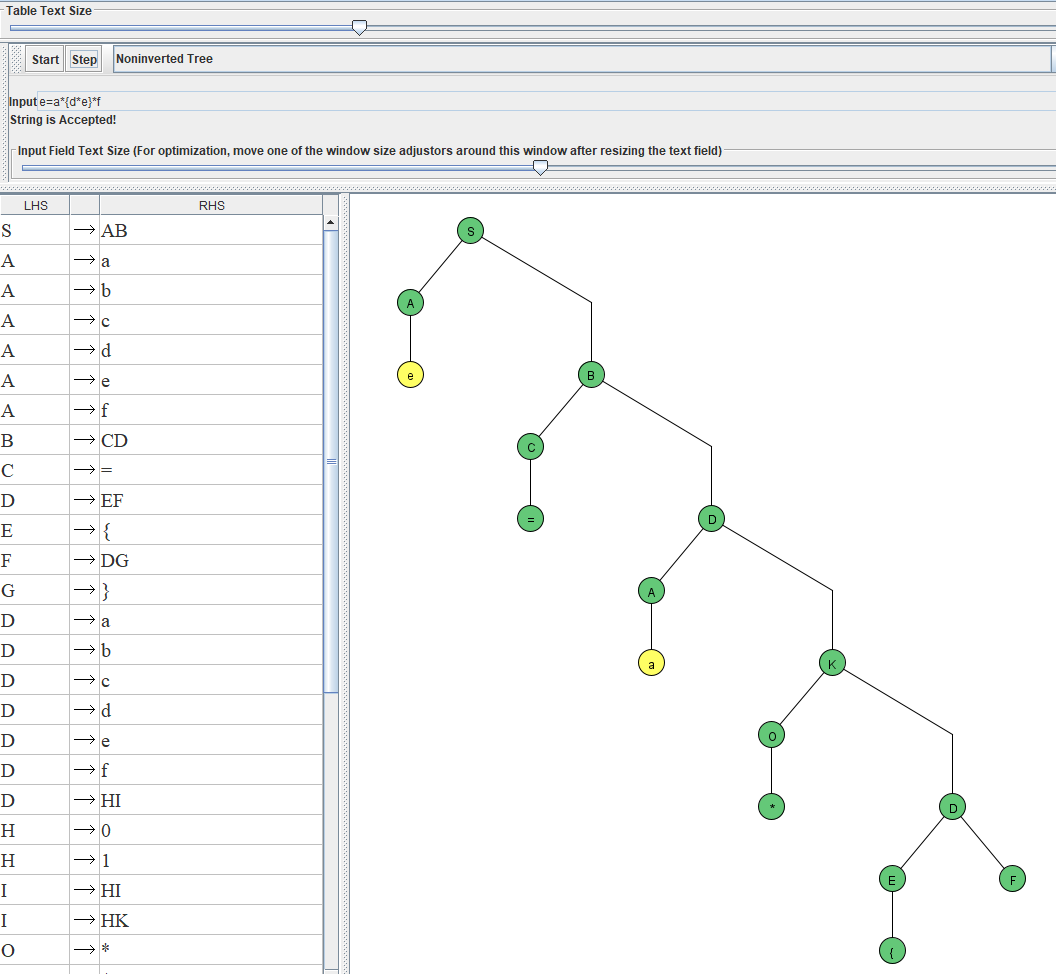


Рисунок 13 – Перехват экрана тестовых цепочек методом CYK

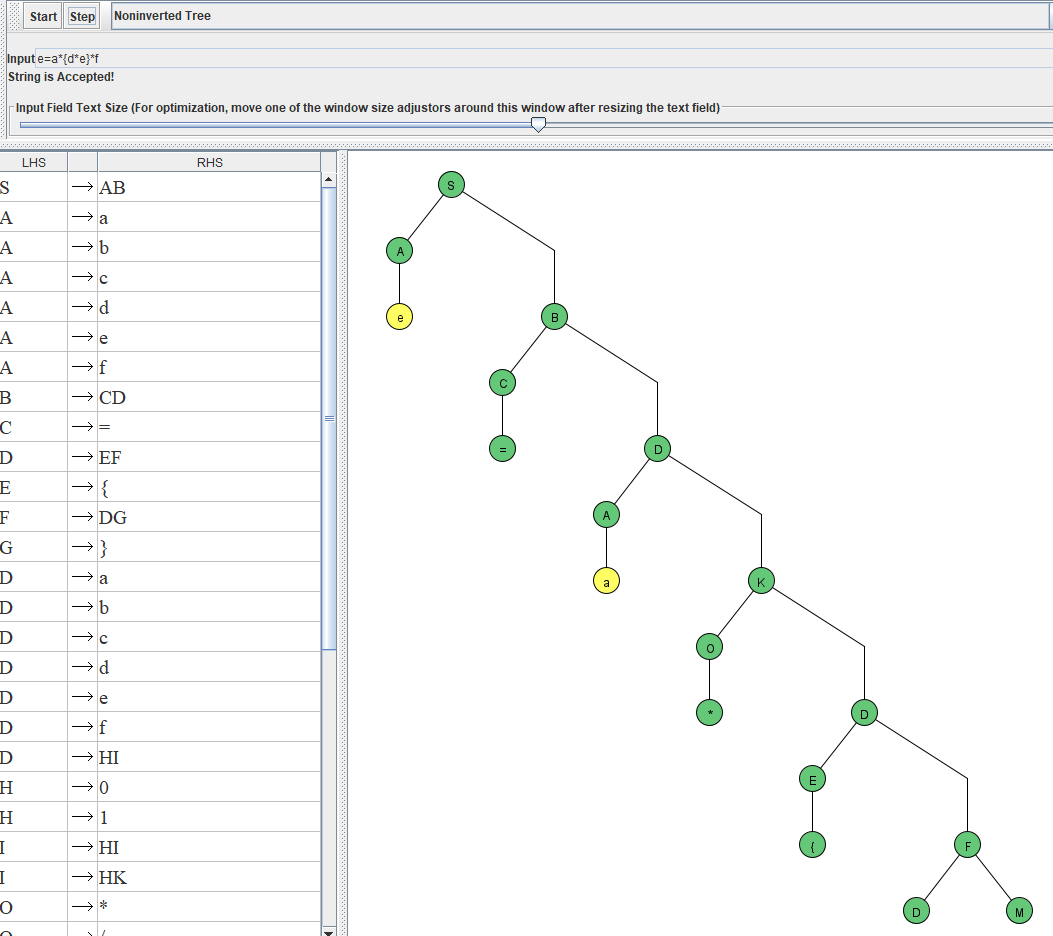


Рисунок 14 – Перехват экрана тестовых цепочек методом CYK

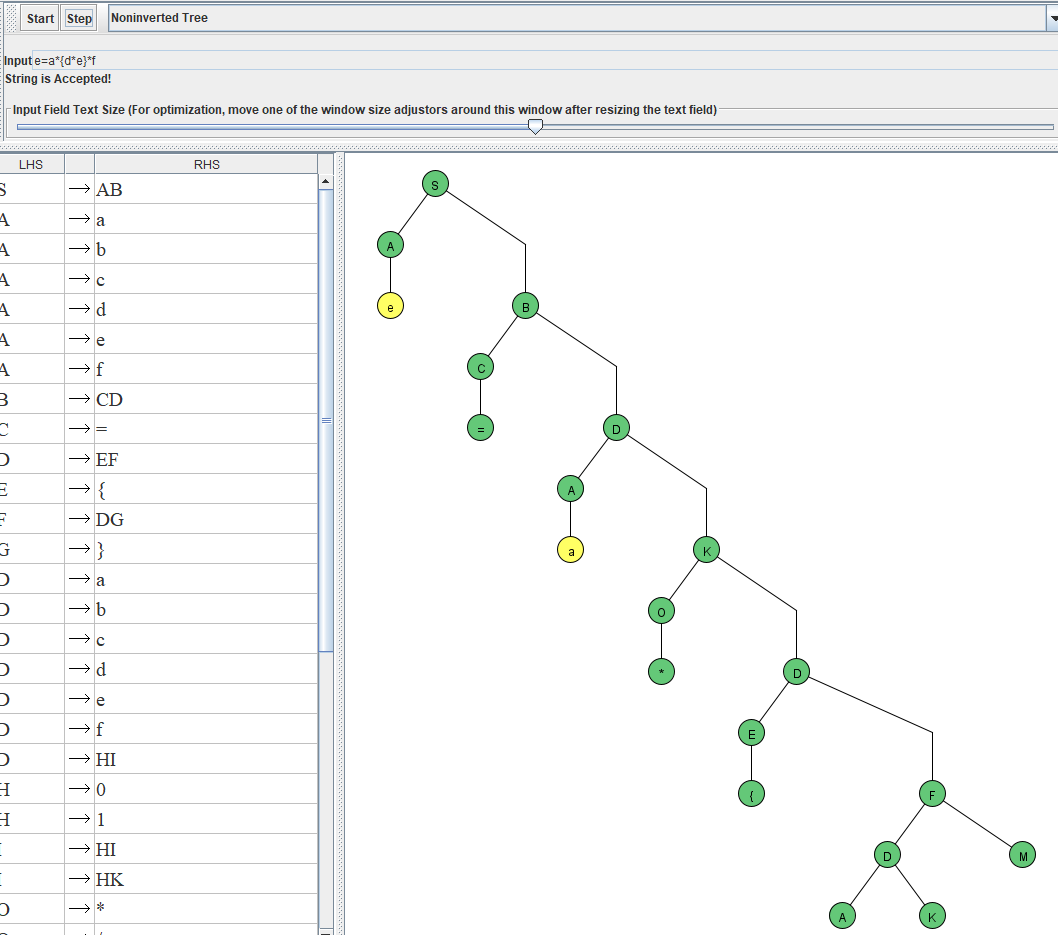


Рисунок 15 – Перехват экрана тестовых цепочек методом CYK

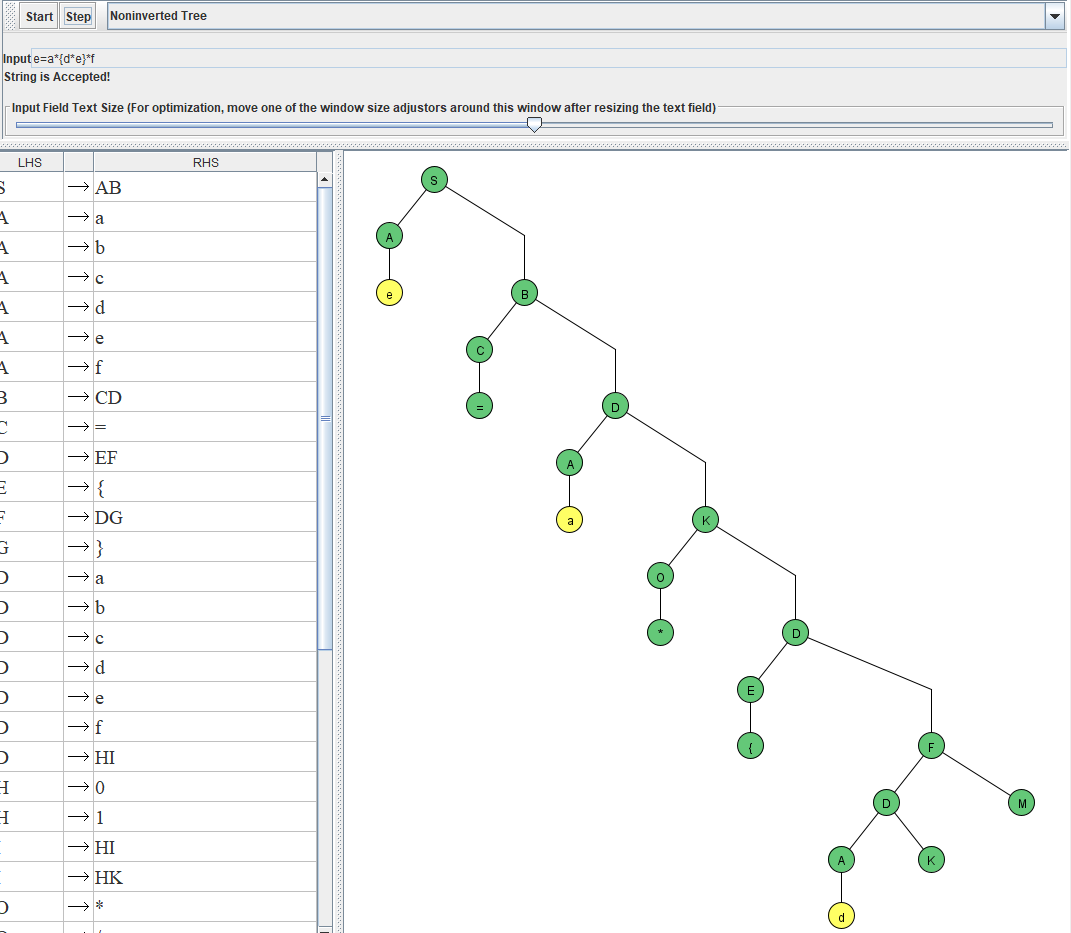


Рисунок 16 – Перехват экрана тестовых цепочек методом CYK

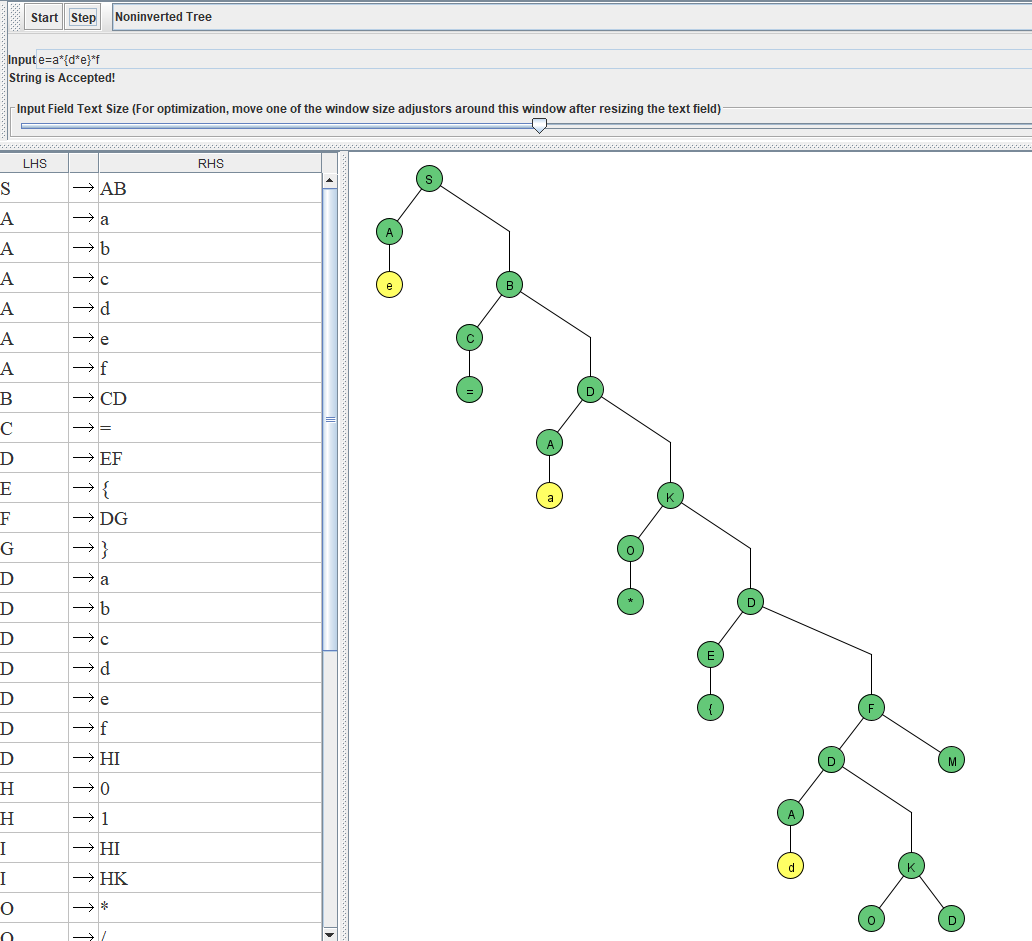


Рисунок 17 – Перехват экрана тестовых цепочек методом CYK

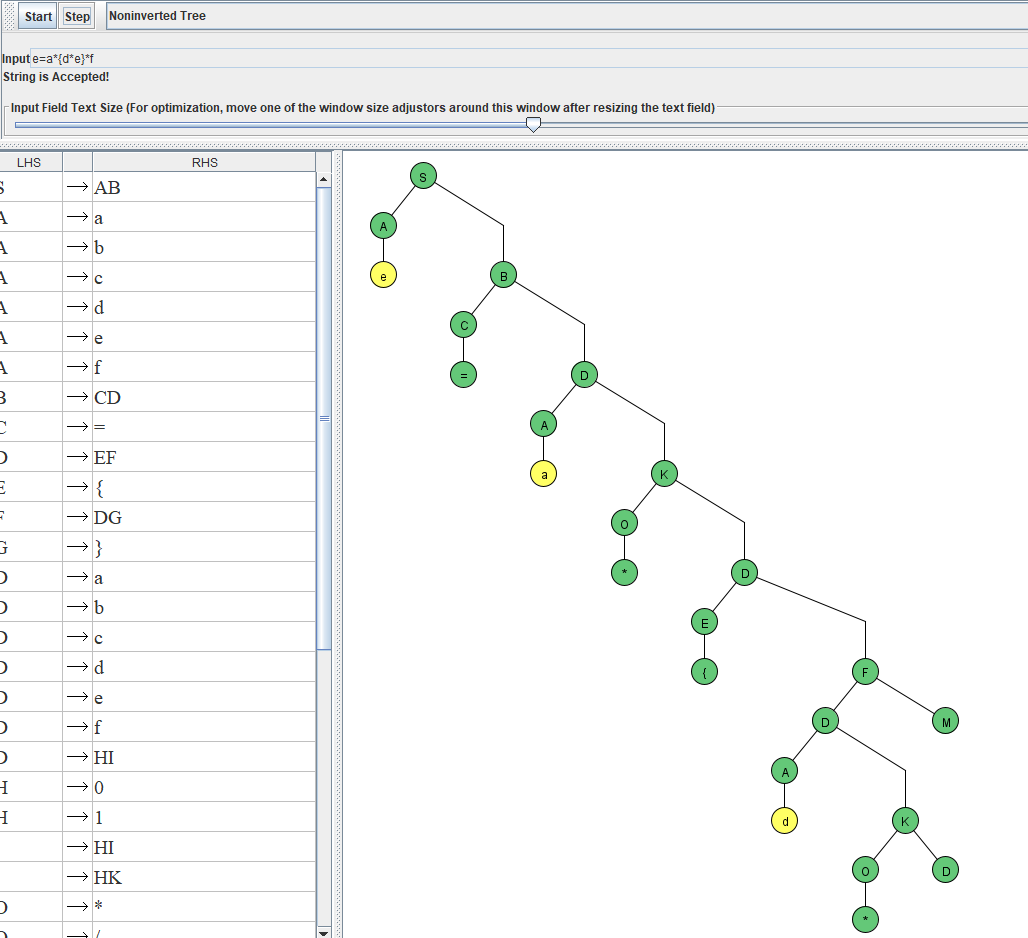


Рисунок 18 – Перехват экрана тестовых цепочек методом CYK

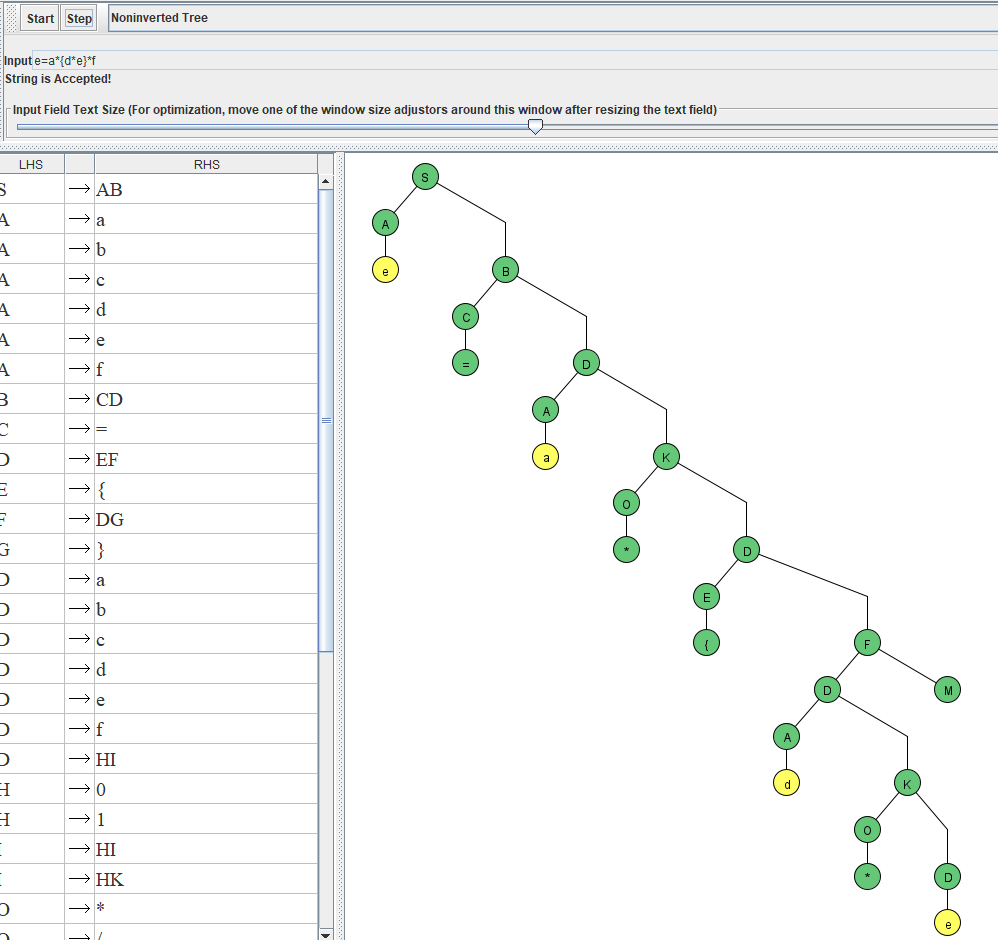


Рисунок 19 – Перехват экрана тестовых цепочек методом CYK

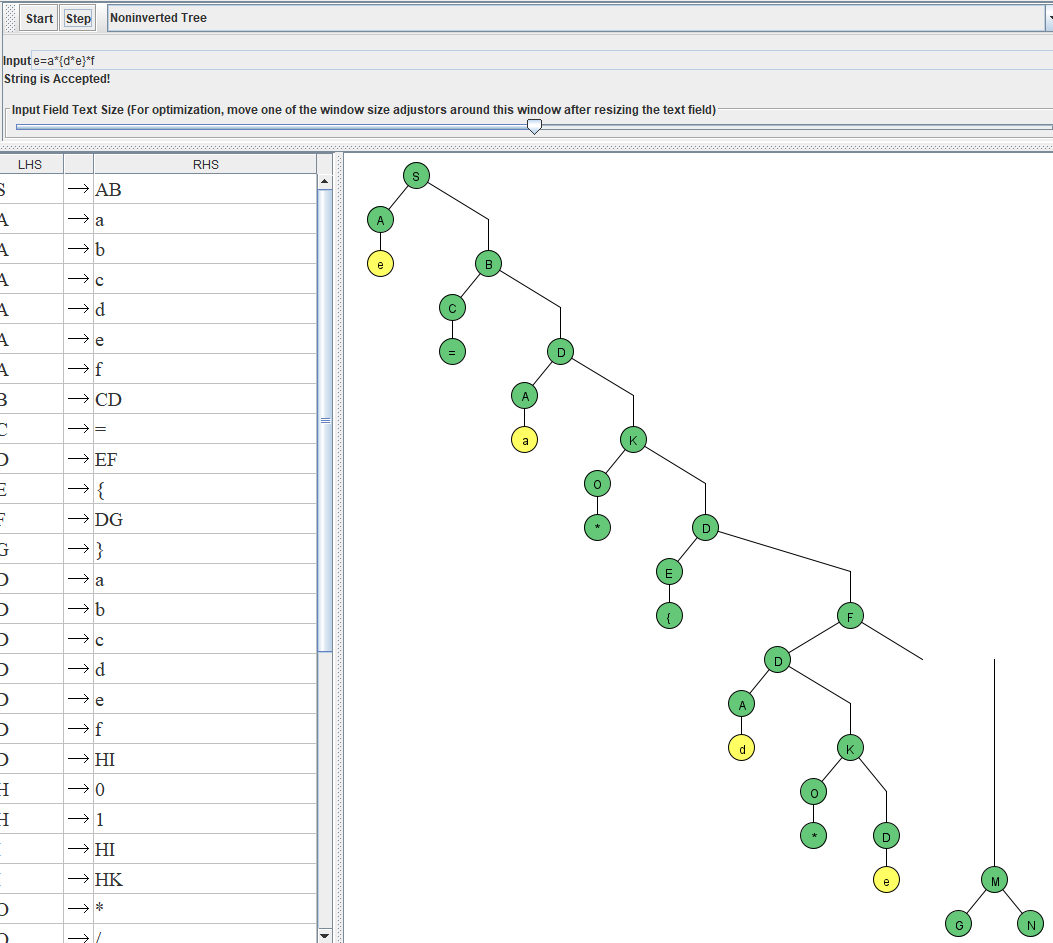


Рисунок 20 – Перехват экрана тестовых цепочек методом CYK

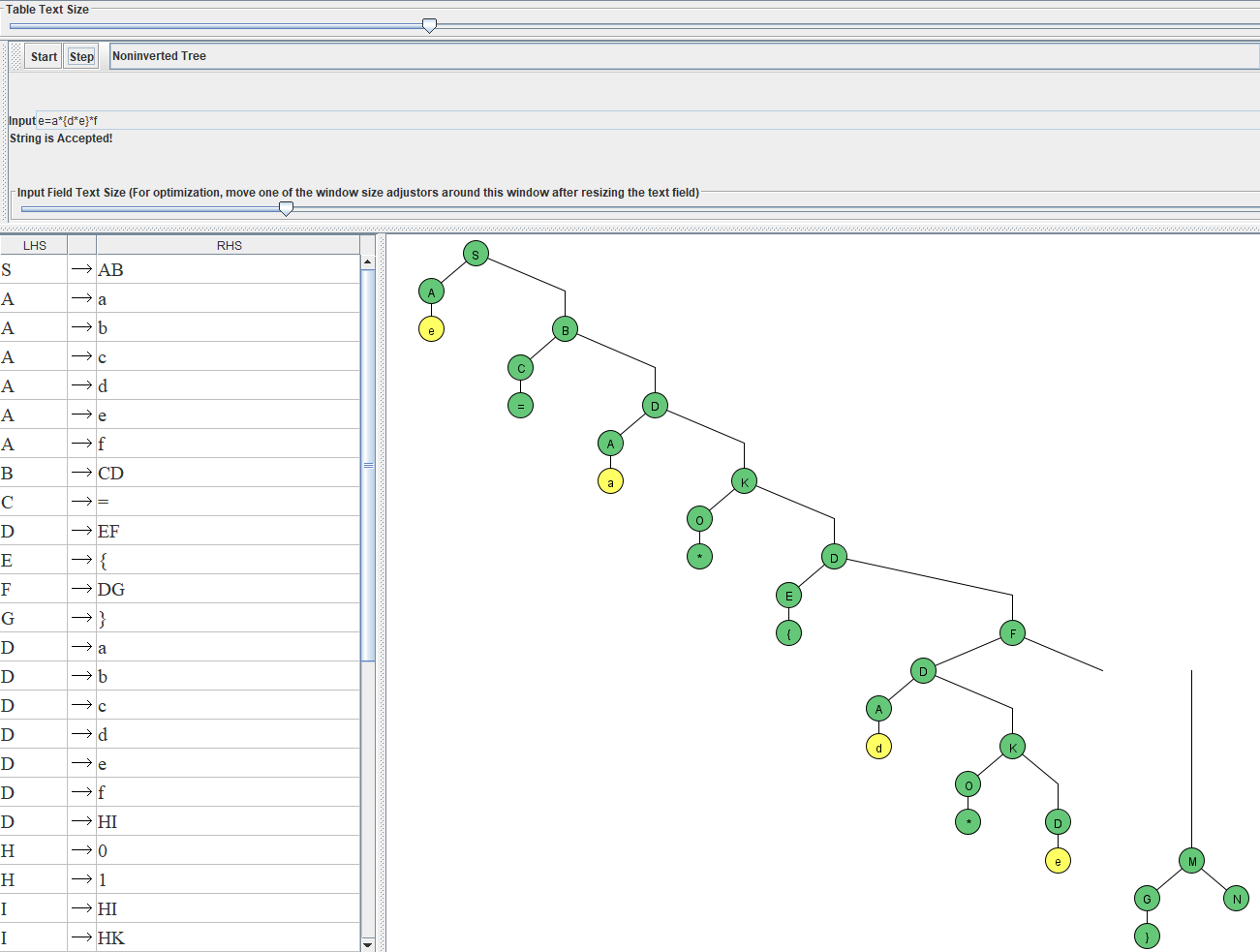


Рисунок 21 – Перехват экрана тестовых цепочек методом CYK

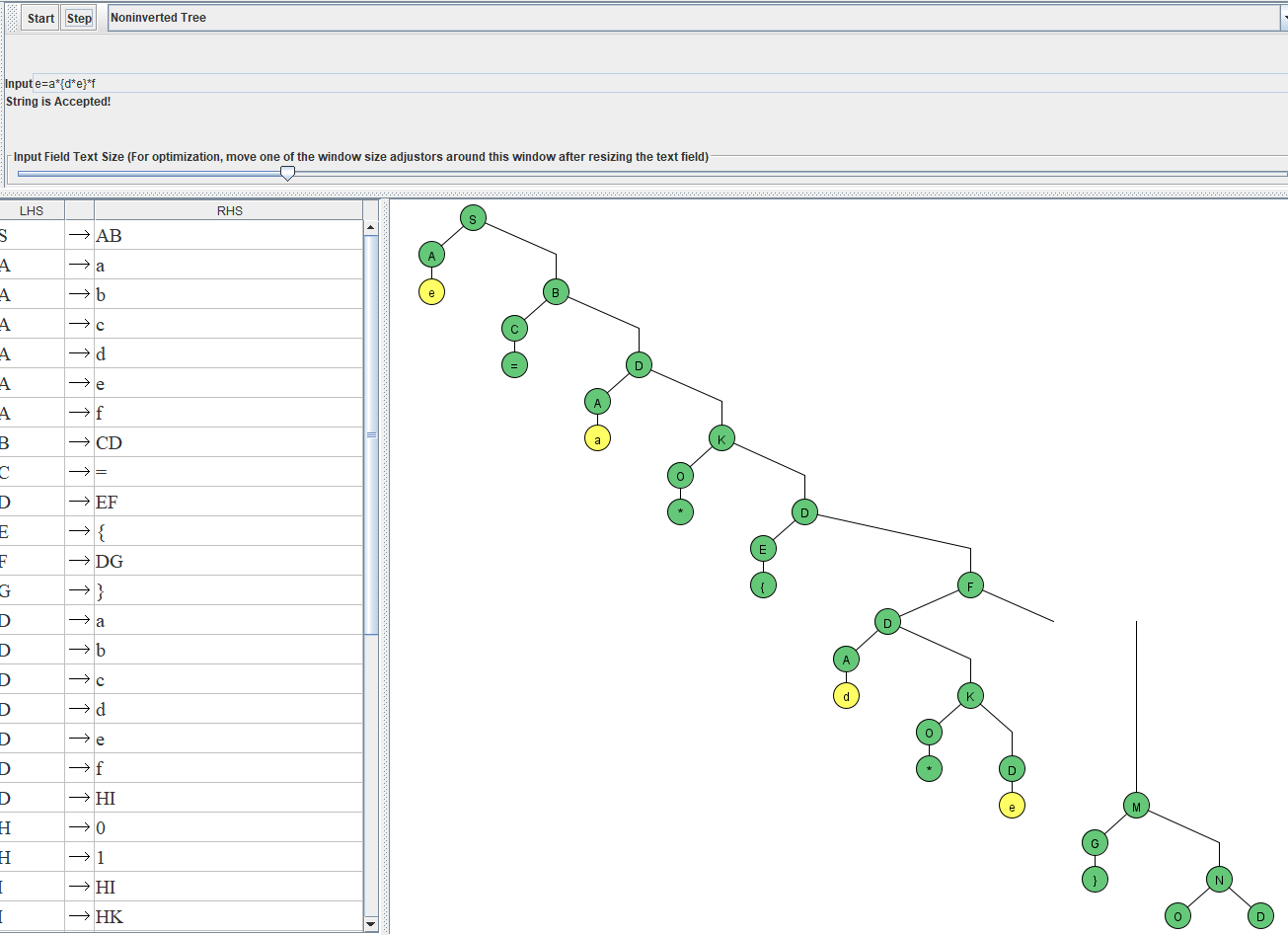


Рисунок 22 – Перехват экрана тестовых цепочек методом CYK

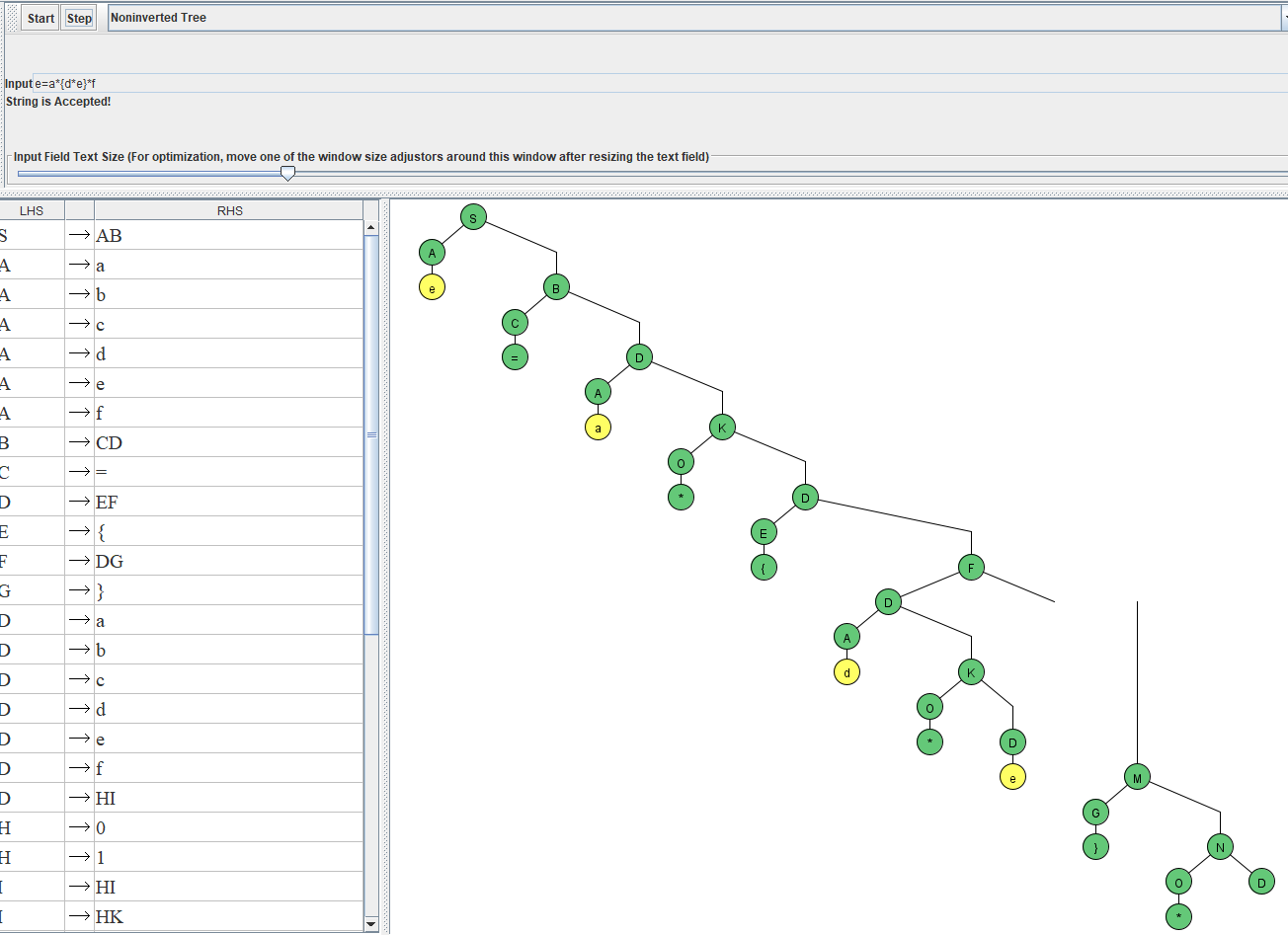


Рисунок 23 – Перехват экрана тестовых цепочек методом CYK

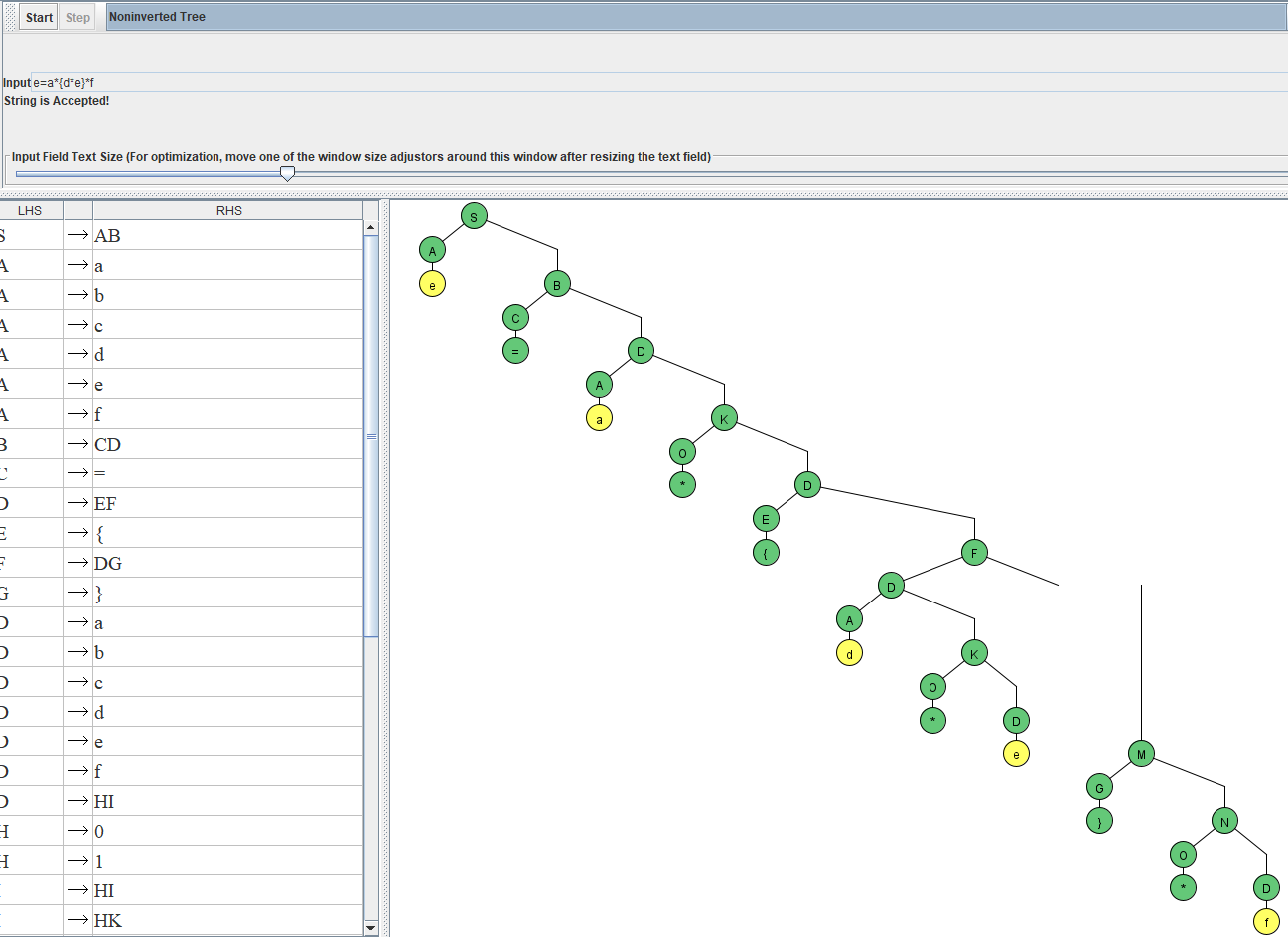


Рисунок 24 – Перехват экрана тестовых цепочек методом CYK

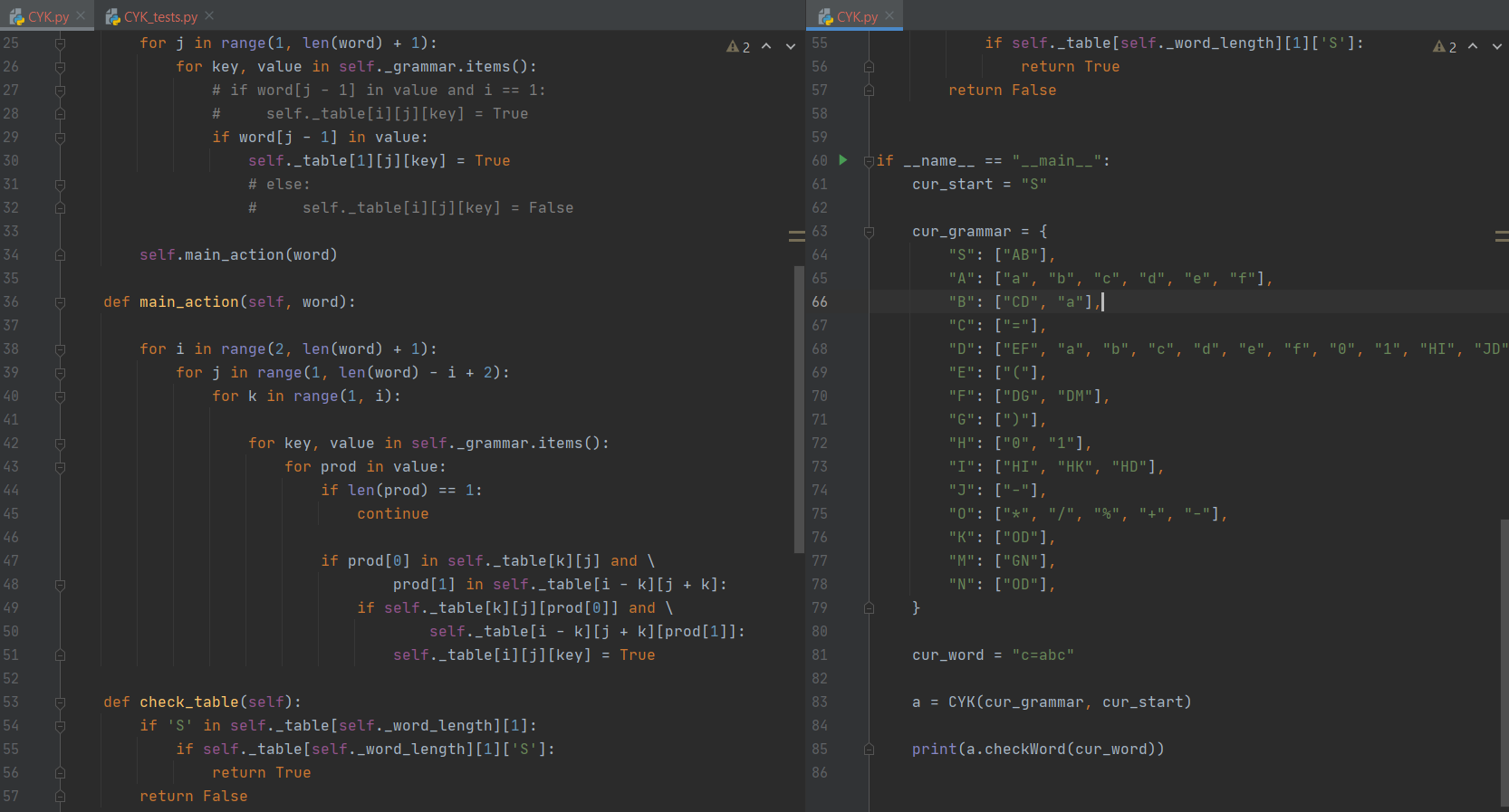


Рисунок 25 – Программная реализация алгоритма CYK (файл CYK.py)

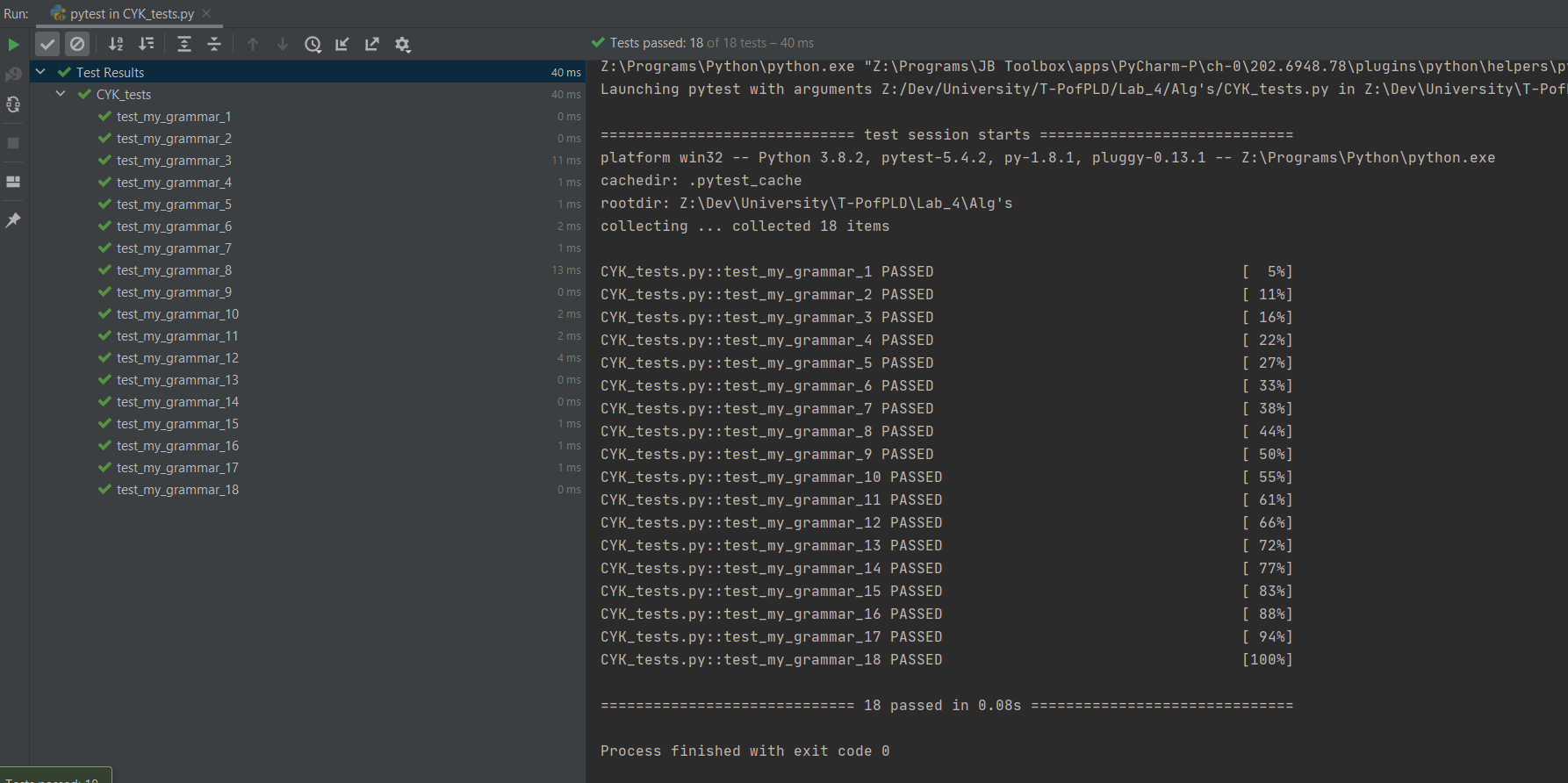


Рисунок 26 – Тестирование реализации (те же входные цепочки, что и для грамматики)

Для запуска тестов необходимо установить библиотеку *pytest* (pip install pytest)для python и выполнить *pytest CYK\_tests.py*

# Вывод

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

**ПРИЛОЖЕНИЕ А**

Листинг 1 – файл CYK.py

UNKNOWN\_SYMBOL\_ERR = 0  
NOT\_REACHED\_FINAL\_STATE = 1  
REACHED\_FINAL\_STATE = 2  
  
  
class CYK:  
 \_word\_length = 0  
  
 def \_\_init\_\_(self, grammar, start):  
 self.\_grammar = grammar  
 self.\_start = start  
 self.\_table = [[{}]] *# Формат хранения: [индекс][индекс][ключ(нетерминальный символ)-значение]* def checkWord(self, word):  
 self.\_word\_length = len(word)  
 self.init\_step(word)  
 return self.check\_table()  
  
 def init\_step(self, word):  
 self.\_table = [[{} *# r* for j in range(len(word) + 1)] *# n* for k in range(len(word) + 1)] *# n  
  
 # for i in range(1, len(word) + 1):* for j in range(1, len(word) + 1):  
 for key, value in self.\_grammar.items():  
 *# if word[j - 1] in value and i == 1:  
 # self.\_table[i][j][key] = True* if word[j - 1] in value:  
 self.\_table[1][j][key] = True  
 *# else:  
 # self.\_table[i][j][key] = False* self.main\_action(word)  
  
 def main\_action(self, word):  
  
 for i in range(2, len(word) + 1):  
 for j in range(1, len(word) - i + 2):  
 for k in range(1, i):  
  
 for key, value in self.\_grammar.items():  
 for prod in value:  
 if len(prod) == 1:  
 continue  
  
 if prod[0] in self.\_table[k][j] and \  
 prod[1] in self.\_table[i - k][j + k]:  
 if self.\_table[k][j][prod[0]] and \  
 self.\_table[i - k][j + k][prod[1]]:  
 self.\_table[i][j][key] = True  
  
 def check\_table(self):  
 if **'S'** in self.\_table[self.\_word\_length][1]:  
 if self.\_table[self.\_word\_length][1][**'S'**]:  
 return True  
 return False  
  
  
if \_\_name\_\_ == **"\_\_main\_\_"**:  
 cur\_start = **"S"** cur\_grammar = {  
 **"S"**: [**"AB"**],  
 **"A"**: [**"a"**, **"b"**, **"c"**, **"d"**, **"e"**, **"f"**],  
 **"B"**: [**"CD"**, **"a"**],  
 **"C"**: [**"="**],  
 **"D"**: [**"EF"**, **"a"**, **"b"**, **"c"**, **"d"**, **"e"**, **"f"**, **"0"**, **"1"**, **"HI"**, **"JD"**, **"AK"**, **"HK"**],  
 **"E"**: [**"("**],  
 **"F"**: [**"DG"**, **"DM"**],  
 **"G"**: [**")"**],  
 **"H"**: [**"0"**, **"1"**],  
 **"I"**: [**"HI"**, **"HK"**, **"HD"**],  
 **"J"**: [**"-"**],  
 **"O"**: [**"\*"**, **"/"**, **"%"**, **"+"**, **"-"**],  
 **"K"**: [**"OD"**],  
 **"M"**: [**"GN"**],  
 **"N"**: [**"OD"**],  
 }  
  
 cur\_word = **"c=abc"** a = CYK(cur\_grammar, cur\_start)  
  
 print(a.checkWord(cur\_word))

Листинг 2 – файл CYK\_tests.py

from CYK import CYK  
  
cur\_start = **"S"**cur\_grammar = {  
 cur\_start: [**"AB"**],  
 **"A"**: [**"a"**, **"b"**, **"c"**, **"d"**, **"e"**, **"f"**],  
 **"B"**: [**"CD"**, **"a"**],  
 **"C"**: [**"="**],  
 **"D"**: [**"EF"**, **"a"**, **"b"**, **"c"**, **"d"**, **"e"**, **"f"**, **"0"**, **"1"**, **"HI"**, **"JD"**, **"AK"**, **"HK"**],  
 **"E"**: [**"("**],  
 **"F"**: [**"DG"**, **"DM"**],  
 **"G"**: [**")"**],  
 **"H"**: [**"0"**, **"1"**],  
 **"I"**: [**"HI"**, **"HK"**, **"HD"**],  
 **"J"**: [**"-"**],  
 **"O"**: [**"\*"**, **"/"**, **"%"**, **"+"**, **"-"**],  
 **"K"**: [**"OD"**],  
 **"M"**: [**"GN"**],  
 **"N"**: [**"OD"**],  
}  
  
cyk = CYK(cur\_grammar, cur\_start)  
  
  
def test\_my\_grammar\_1():  
  
 cur\_word = **"a=a\*b"** assert cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_2():  
  
 cur\_word = **"b=(a\*1010)"** assert cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_3():  
  
 cur\_word = **"c=a\*111111111111111\*a"** assert cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_4():  
  
 cur\_word = **"d=(b\*1)\*c"** assert cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_5():  
  
 cur\_word = **"e=a\*(d\*e)\*f"** assert cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_6():  
  
 cur\_word = **"f=a\*(a)\*a"** assert cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_7():  
  
 cur\_word = **"a=(a\*-1010)"** assert cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_8():  
  
 cur\_word = **"b=a\*-111111111111111\*a"** assert cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_9():  
  
 cur\_word = **"c=(b\*1)\*c"** assert cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_10():  
  
 cur\_word = **"d=1+(a-(b\*c)/f)"** assert cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_11():  
  
 cur\_word = **"e=a\*-(d\*e)\*f"** assert cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_12():  
  
 cur\_word = **"f=a\*((-a))\*a"** assert cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_13():  
  
 cur\_word = **"a=a--b"** assert cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_14():  
  
 cur\_word = **"b=a\*\*b"** assert not cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_15():  
  
 cur\_word = **"c=abc"** assert not cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_16():  
  
 cur\_word = **"d=a\*-((de)\*f"** assert not cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_17():  
  
 cur\_word = **"e=a\*(-a))\*a"** assert not cyk.checkWord(cur\_word)  
  
  
def test\_my\_grammar\_18():  
  
 cur\_word = **"a+b"** assert not cyk.checkWord(cur\_word)