Министерство образования и науки российской федерации

(минобрнауки россии)

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет» (ФГБОУ ВПО «СПбГПУ»)

**Институт менеджмента и информационных технологий**

(филиал)федерального государственного бюджетного образовательного учреждения высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет» в г. Череповце (ИМИТ «СПбГПУ»)

Кафедра ПО ВТ и АС

ЛАБОРАТОРНАЯ РАБОТА №4

Дисциплина: «Теория языков программирования и методы трансляции»

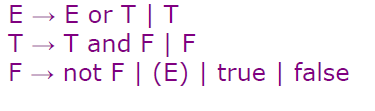
Тема: «МП-автоматы для нисходящего синтаксического анализа»

|  |  |  |
| --- | --- | --- |
| Выполнил студент группы о.291: | Шанин Игнат Леонидович | |
|  | № зачетной книжки: о2080127 | |
| Проверил: | Михайлов Андрей Евгеньевич | |
|  | «\_\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_ г. | |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | отметка о зачете | подпись преподавателя |

# Задание

На основе порождающей КС-грамматики построить МП-автомат для нисходящего синтаксического анализа без возвратов и реализовать его на любом ЯП. Алгоритм построения должен основываться на функциях FIRST() и FOLLOW(). Программа должна принимать на вход файл, содержащий строки символов. На выходе формируется сообщение о том, принадлежат или нет строки языку, порождаемому грамматикой.

Грамматика 1:



Грамматика 2:

R R^F | RT | R\* | (R) | F

F T | b

T +a | -a

# Теоретические сведения

#### Основная работа по построению анализатора – это создание множества , которое удобно представить в виде таблицы.

Нисходящий анализ, как правило, включает в себя только одно состояние, и поэтому требуется одна таблица. Такая таблица содержит столбцы с входными символами и строки с символами магазина. В ячейки таблицы помещаются правила вида выполнение которых приводит к удалению из магазина символа Ф и добавления в него цепочки , начиная с последнего символа. Для построения таблицы необходимы функции FIRST и FOLLOW.

## Функции FIRST и FOLLOW

Пусть - это КС-грамматика. Тогда для произвольной цепочки  можно определить функцию FIRST() как множество терминальных символов, с которых начинаются строки, выводимые из . Если из  можно вывести пустую строку , то она также принадлежит этому множеству.

Функция FOLLOW() определяет для некоторого нетерминала  множество терминальных символов, которые могут появиться непосредственно после  в некой сентенциальной форме грамматики, то есть это множество символов таких, что существует вывод вида  Если  самым правым некой цепочки вывода, то к множеству FOLLOW() добавляется символ окончания строки.

Для вычисления функции FIRST() вводится функция FIRST(), где .

## Построение таблицы предикативного анализатора

Для построения таблицы предиктивного анализатора используется алгоритм, основанный на следующем принципе. Предположим, что правило выводаграмматики и есть некий терминальный символ . Тогда анализатор заменяется  на  если входной символ в данный момент является .

# Решение

## Преобразование грамматик

### Грамматика 1

* E → E or T | T
* T → T and F | F
* F → not F | (E) | true | false

Устранение левой рекурсии:

* E → E or T | T
  + E → ТE’
  + E’→ or ТЕ’ | λ
* T → T and F | F
  + Т → FТ’
  + T’→ and FT’ | λ

Результат:

* E → ТE’
* E’→ or ТЕ’ | λ
* Т → FТ’
* T’→ and FT’ | λ
* F → not F | (E) | true | false

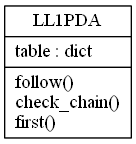
### Грамматика 2

* R → R^F | RT | R\* | (R) | F
* F → T | b
* T → +a | -a
* Устранение левой рекурсии:
* R → R^F | RT | R\* | (R) | F
  + R → FR’ | TR’ | (R)
  + R’→ ^FR’ | TR’ | \*R’| λ

Результат:

* R → FR’ | TR’ | (R)
* R’→ ^FR’ | TR’ | \*R’| λ
* F → T | b
* T → +a | -a

## Листинг программы

**class** **LL1PDA**:

    """Pushdown automaton for LL(1) grammar"""

**def** **\_\_init\_\_**(self, G):

        self**.**\_\_dict\_\_**.**update(G)

        table **=** dict([(A, dict()) **for** A **in** self**.**N])

**for** nonterm, rules **in** self**.**P**.**items():

**for** rule **in** rules:

**if** rule **==** '':

**for** term **in** self**.**follow(nonterm):

                        table[nonterm][term] **=** tuple()

**continue**

**for** term **in** filter(**lambda** e: e **in** self**.**T, self**.**first(rule)):

                    table[nonterm][term] **=** rule

**if** '' **in** self**.**first(rule):

**for** term **in** self**.**follow(nonterm):

                        table[nonterm][term] **=** rule

        self**.**table **=** table

**def** **first**(self, alpha):

        s **=** set()

*# if symbol*

**if** type(alpha) **is** str:

*# if terminal*

**if** alpha **in** self**.**T:

**return** set([alpha])

*# else checking rules*

**for** rule **in** self**.**P[alpha]:

**if** rule **==** '':

                    s**.**add('')

**else**:

                    s**.**update(self**.**first(rule))

*# if chain*

**else**:

**for** i, symbol **in** enumerate(alpha):

                symbols\_first **=** self**.**first(symbol)

                s**.**update(symbols\_first**.**difference(['']))

**if** '' **not** **in** symbols\_first:

**break**

**if** i **==** len(alpha) **-** 1 **and** '' **in** symbols\_first:

                    s**.**add('')

**return** s

**def** **follow**(self, X):

        s **=** set()

*# if X is axiom*

**if** X **==** self**.**S:

            s**.**add('$')

*# for each rule that have X in right part*

**for** nonterm, rules **in** self**.**P**.**items():

**for** rule **in** filter(**lambda** e: X **in** e, rules):

*# for each X in rule*

**for** i, x **in** filter(**lambda** e: e[1] **==** X, enumerate(rule)):

*# if it isn't last one*

**if** i **<** len(rule) **-** 1:

*# for rule like A -> aXb b is the following chain*

                        following\_chain **=** rule[i **+** 1:]

                        following\_first **=** self**.**first(following\_chain)

                        s**.**update(following\_first**.**difference(['']))

*# if folowing chain can become empty*

**if** '' **in** following\_first:

*# do the same thing as X was at the end*

                            s**.**update(self**.**follow(nonterm))

*# if X at the end, than it can have same follow as*

*# nondeterminate on the left side*

**else**:

**if** nonterm **!=** X:

                            s**.**update(self**.**follow(nonterm))

**return** s

**def** **check\_chain**(self, chain):

        chain **=** chain**.**split() **+** ['$']

        stack **=** [self**.**S, '$']

**while** True:

**if** stack[0] **in** self**.**N:

**try**:

                    rule **=** self**.**table[stack[0]][chain[0]]

                    stack**.**pop(0)

                    stack **=** list(rule) **+** stack

**except** **KeyError**:

**return** False

**elif** stack[0] **in** self**.**T:

**if** stack**.**pop(0) **!=** chain**.**pop(0):

**return** False

**elif** stack[0] **==** '$':

**return** stack[0] **==** chain[0]

*# Grammar 1*

G1 **=** {

    'T' : ('or', 'and', 'not', 'true', 'false', '(', ')'),

    'N' : ('E', 'E`', 'T', 'T`', 'F'),

    'P' : {

        'E' : [('T', 'E`')],

        'E`' : [('or', 'T', 'E`'), ''],

        'T' : [('F', 'T`')],

        'T`' : [('and', 'F', 'T`'), ''],

        'F' : [('not', 'F'), ('(', 'E', ')'), ('true', ), ('false', )]},

    'S' : 'E'}

*# Grammar 2*

G2 **=** {

    'T' : ('+', '-', 'a', 'b', '(', ')', '^', '\*'),

    'N' : ('R', 'R`', 'F', 'T'),

    'P' : {

        'R' : [('F', 'R`'), ('F', 'R`'), ('(', 'R', ')')],

        'R`' : [('^', 'F', 'R`'), ('T', 'R`'), ('\*', 'R`'), ''],

        'F' : [('T', ), ('b', )],

        'T' : [('+', 'a'), ('-', 'a')]},

    'S' : 'R'}

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    automaton1 **=** LL1PDA(G1)

    automaton2 **=** LL1PDA(G2)

**with** open("test.txt") **as** f:

        chains **=** f**.**read()**.**split('\n')

    chains1 **=** filter(automaton1**.**check\_chain, chains)

    chains2 **=** filter(automaton2**.**check\_chain, chains)

    other **=** filter(**lambda** x: x **not** **in** chains1 **and** x **not** **in** chains2, chains)

**with** open("1.txt", "w") **as** f:

        f**.**write('\n'**.**join(chains1))

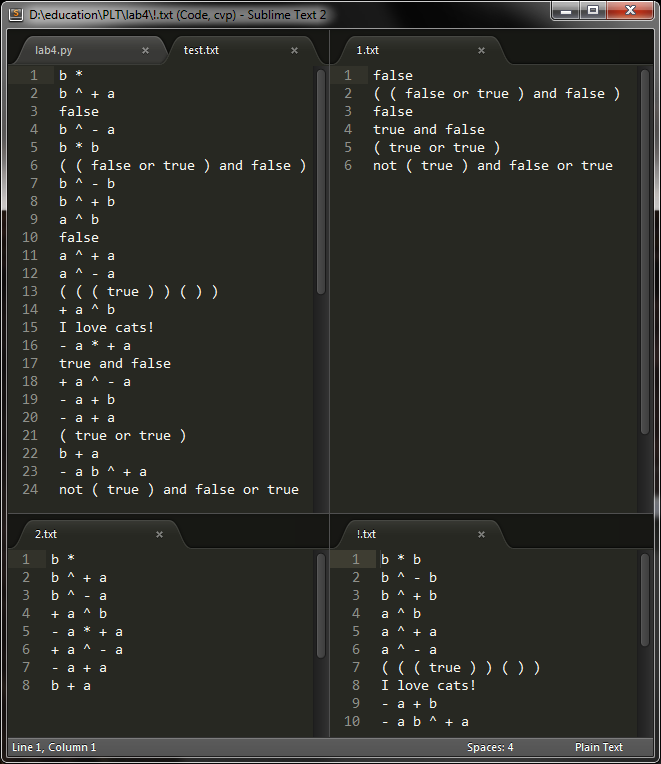
**with** open("2.txt", "w") **as** f:

        f**.**write('\n'**.**join(chains2))

**with** open("!.txt", "w") **as** f:

        f**.**write('\n'**.**join(other))

# Результат работы программы



# Вывод

Я научился использовать МП автомат для нисходящего синтаксического анализа, автоматически генерируя для него таблицу по заданной LL(1) грамматике.