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

Построить для указанных грамматик отношения предшествование между символами.

Написать программу, реализующую восходящий анализ на основе отношений предшествования на базе МП-автомата:

Грамматика 1

S → aSbA | bc

A → aSb | d

Грамматика 2

S → TR | T

R → +T | -T | +TR | -TR

T → EF | E

F → \*E | /E | \*EF | /EF

E → (S) | a | b

Программа должна принимать на вход тестовый файл со строками, а выходе формировать

файл с построением обратного вывода. Пример:

b+a ← E+a ← T+a ← T+E ← T+T ← TR ← S

a+ab ← E+ab ← T+ab ← ошибка

# Краткая теория

Грамматики предшествования

Пусть G = <N, Т, Р, S> — КС-грамматика и , и пусть в ходе порождения предложений языка L(G) на некотором шаге вывода получена сен­тенциальная форма вида

 где 

Символы  и  стоят рядом в сентенциальной форме. При этом их сосед­ство можно охарактеризовать с помощью отношений специального вида, кото­рые отражают одну из трех возможных ситуаций, возникающих в ходе вывода. Эти отношения называют *отношениями предшествования.*

1. Между символами  и  существует отношение , если в грамма­тике G есть правило вида

, где 

2. Между символами  и  существует отношение , если в грамма­тике G есть правила вида

 и вывод , где .

3. Между символами  и  существует отношение , если в грамма­тике G есть правила вида



и выводы

 и , где 

или правило вида  и вывод .

Другими словами, существование отношения предшествования между сим­волами  и  говорит о том, что символы  и могут стоять рядом в промежу­точных цепочках вывода. А тип этого отношения показывает, что если:

* *,* то символы  и  принадлежат одной основе;
* , то символ  – самый левый символ некоторой основы;
* , то символ  – самый правый символ некоторой основы.

КС-грамматика G называется *грамматикой простого пред­шествования*, если она не содержит λ-правил и для любой пары ее символов выполняется не более одного отношения предшествования.

Для грамматик простого предшествования существует простой и эффектив­ный детерминированный анализатор предшествования. Его можно представить в виде устройства, имеющего входную ленту и магазин, и использующего в работе отношения предшествования и правила вывода грамматики.

Такт работы анализатора начинается с определения отношения предше­ствования, которое выполняется для пары, образованной из верхнего символа магазина и левого символа входной цепочки. При этом возможны следующие варианты:

1) Ни одно из отношений не определено, т.е. символы не могут стоять рядом в цепочке. Тогда входная цепочка *отвергается.*

2) Для пары определено отношение  или . Тогда выполняется действие *перенос:* анализатор записывает в магазин символ , если это отношение вы­полняется для данной пары, переносит после этого в магазин левый символ входной цепочки и сдвигает входную цепочку на одну ячейку влево.

3) Для пары определено отношение . Тогда анализатор пытается произве­сти *свертку:*

Шаг 1. Просматривает справа налево символы в магазине, пока не будет обнаружен символ  и выделяет подцепочку, состоящую из просмотренных символов (без символа *).* Переходит к шагу 2.

Шаг 2. Ищет правило вывода грамматики, правая часть Которого совпа­дает с выделенной на первом шаге подцепочкой. Если такое правило найдено, то переходит к шагу 3, в противном случае – к шагу 5.

Шаг 3. Исключает из магазина найденные на первом шаге подцепочку и символ . Образует пару, состоящую из символа, оказавшегося в магазине пра­вым, и символа из левой части найденного на втором шаге правила граммати­ки. Переходит к шагу 4.

Шаг 4. Ищет отношение предшествования для образованной на третьем шаге пары (это могут быть лишь отношения  или ). Если ни одно из отно­шений не выполняется, то входная цепочка отвергается. В противном случае в магазин записывается символ  (если выполняется отношение ) и нетерми­нальный символ из левой части правила. После этого *свертка* заканчивается.

Шаг 5. Анализирует содержимое магазина и входной ленты. Если в мага­зине находится цепочка #S, где S — начальный символ (аксиома) граммати­ки, а на входной ленте остался один лишь маркер конца цепочки ⊥*,* то входная цепочка *допускается.* В противном случае она *отвергается.*

В ходе своей работы анализатор предшествования использует информацию о правилах вывода грамматики и об отношениях предшествования, заданных на множестве всех символов грамматики. Последние обычно представляются в виде квадратной *матрицы предшествования,* порядок которой равен n+1, где n — число терминальных и нетерминальных символов грамматики.

# Решение

## Отношения предшествования

### Грамматика 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| . | S | A | a | b | c | d | ┴ |
| S | . | . | . | = | . | . | = |
| A | . | . | . | > | . | . | > |
| a | = | . | < | < | . | . | . |
| b | . | = | < | > | = | < | > |
| c | . | . | . | > | . | . | > |
| d | . | . | . | > | . | . | > |
| ┴ | = | . | < | < | . | . | . |

### Грамматика 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| . | + | - | \* | / | ( | ) | a | b | S | R | T | F | E | ┴ |
| + | . | . | . | . | < | . | < | < | . | . | = | . | < | . |
| - | . | . | . | . | < | . | < | < | . | . | = | . | < | . |
| \* | . | . | . | . | < | . | < | < | . | . | . | . | = | . |
| / | . | . | . | . | < | . | < | < | . | . | . | . | = | . |
| ( | . | . | . | . | < | . | < | < | = | . | < | . | < | . |
| ) | > | > | > | > | . | > | . | . | . | > | . | > | . | > |
| a | > | > | > | > | . | > | . | . | . | > | . | > | . | > |
| b | > | > | > | > | . | > | . | . | . | > | . | > | . | > |
| S | . | . | . | . | . | = | . | . | . | . | . | . | . | . |
| R | . | . | . | . | . | > | . | . | . | . | . | . | . | > |
| T | < | < | . | . | . | > | . | . | . | = | . | . | . | > |
| F | > | > | . | . | . | > | . | . | . | > | . | . | . | > |
| E | > | > | < | < | . | > | . | . | . | > | . | = | . | > |
| ┴ | . | . | . | . | < | . | < | < | . | . | < | . | < | . |

## Листинг программы

*#!/usr/bin/python*

*# -\*- coding: utf-8 -\*-*

**class** **PPDA**:

    """Pushdown automaton for precedence grammar"""

**def** **\_\_init\_\_**(self, grammar, precedence\_matrix):

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

        self**.**T**.**add('|')

        rows **=** map(str**.**split, precedence\_matrix**.**split('\n')[1:**-**1])

        precedences **=** dict()

**for** row **in** rows[1:]:

            precedences[row[0]] **=** dict()

**for** relation, symbol **in** zip(row[1:], rows[0][1:]):

**if** relation **!=** '.':

                    precedences[row[0]][symbol] **=** relation

        rules **=** dict()

**for** rule **in** self**.**P:

            L, D **=** rule**.**replace(' ', '')**.**split('->')

            rules[L] **=** set(D**.**split('|'))

        self**.**rules **=** rules

        self**.**precedences **=** precedences

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

        result **=** chain

        chain **+=** '|'

        stack **=** '|'

**while** True:

**if** stack **+** chain **==** '|' **+** self**.**S **+** '|':

**return** result **+** ' SUCCESS!'

**try**:

**if** self**.**precedences[stack[**-**1]][chain[0]] **==** '>': *#reduce*

                    i **=** **-**1

**while** self**.**precedences[stack[i **-** 1]][stack[i]] **==** '=':

                        i **-=** 1

                    chain\_to\_reduce **=** stack[i:]

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

**if** chain\_to\_reduce **in** rules:

                            stack **=** stack[:i] **+** nonterm

                            result **+=** ' -> ' **+** (stack **+** chain)[1:**-**1]

**break**

**else**: *#shift*

                   stack **+=** chain[0]

                   chain **=** chain[1:]

**except** (**KeyError**, **IndexError**):

**return** result **+** ' -> FAIL!'

*# Grammar 1*

G1 **=** {

    'T': {'a', 'b', 'c', 'd'},

    'N': {'S', 'A'},

    'P': {

        'S -> aSbA | bc',

        'A -> aSb | d'},

    'S': 'S'}

MATRIX1 **=** """

. S A a b c d |

S . . . = . . =

A . . . > . . >

a = . < < . . .

b . = < > = < >

c . . . > . . >

d . . . > . . >

| = . < < . . .

"""

*# Grammar 2*

G2 **=** {

    'T': {'+', '-', '\*', '/', '(', ')', 'a', 'b'},

    'N': {'S', 'R', 'T', 'F', 'E'},

    'P': {

        'S -> TR | T',

        'R -> +T | -T | +TR | -TR',

        'T -> EF | E',

        'F -> \*E | /E | \*EF | /EF',

        'E -> (S) | a | b'

        },

    'S': 'S'}

MATRIX2 **=** """

. + - \* / ( ) a b S R T F E |

+ . . . . < . < < . . = . < .

- . . . . < . < < . . = . < .

\* . . . . < . < < . . . . = .

/ . . . . < . < < . . . . = .

( . . . . < . < < = . < . < .

) > > > > . > . . . > . > . >

a > > > > . > . . . > . > . >

b > > > > . > . . . > . > . >

S . . . . . = . . . . . . . .

R . . . . . > . . . . . . . >

T < < . . . > . . . = . . . >

F > > . . . > . . . > . . . >

E > > < < . > . . . > . = . >

| . . . . < . < < . . < . < .

"""

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

    automaton1 **=** PPDA(G1, MATRIX1)

    automaton2 **=** PPDA(G2, MATRIX2)

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

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

    result **=** '\n'**.**join(["First grammar:"] **+** map(automaton1**.**check\_chain, chains)) **+** '\n'

    result **+=** '\n'**.**join(["Second grammar:"] **+** map(automaton2**.**check\_chain, chains))

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

        f**.**write(result)

# Результат работы программы

## Входной файл

a\*b-b+a/b

b-a/b

abcbaabcbdb

abcbd

a\*b-(a\*b-b+a/b)+a/b

bacabaca

## Результат

# First grammar:

# a\*b-b+a/b -> FAIL!

# b-a/b -> FAIL!

# abcbaabcbdb -> aSbaabcbdb -> aSbaaSbdb -> aSbaaSbAb -> aSbaSb -> aSbA -> S SUCCESS!

# abcbd -> aSbd -> aSbA -> S SUCCESS!

# a\*b-(a\*b-b+a/b)+a/b -> FAIL!

# bacabaca -> FAIL!

# Second grammar:

# a\*b-b+a/b -> E\*b-b+a/b -> E\*E-b+a/b -> EF-b+a/b -> T-b+a/b -> T-E+a/b -> T-T+a/b -> T-T+E/b -> T-T+E/E -> T-T+EF -> T-T+T -> T-TR -> TR -> S SUCCESS!

# b-a/b -> E-a/b -> T-a/b -> T-E/b -> T-E/E -> T-EF -> T-T -> TR -> S SUCCESS!

# abcbaabcbdb -> FAIL!

# abcbd -> FAIL!

# a\*b-(a\*b-b+a/b)+a/b -> E\*b-(a\*b-b+a/b)+a/b -> E\*E-(a\*b-b+a/b)+a/b -> EF-(a\*b-b+a/b)+a/b -> T-(a\*b-b+a/b)+a/b -> T-(E\*b-b+a/b)+a/b -> T-(E\*E-b+a/b)+a/b -> T-(EF-b+a/b)+a/b -> T-(T-b+a/b)+a/b -> T-(T-E+a/b)+a/b -> T-(T-T+a/b)+a/b -> T-(T-T+E/b)+a/b -> T-(T-T+E/E)+a/b -> T-(T-T+EF)+a/b -> T-(T-T+T)+a/b -> T-(T-TR)+a/b -> T-(TR)+a/b -> T-(S)+a/b -> T-E+a/b -> T-T+a/b -> T-T+E/b -> T-T+E/E -> T-T+EF -> T-T+T -> T-TR -> TR -> S SUCCESS!

# bacabaca -> FAIL!

# Вывод

Я научился, реализовывать восходящий анализ на основе грамматики простого предшествования с заданными правилами. На входе программа принимает строки, а на выходе выдаёт файл с обратными выводами этих строк.