Министерство образования Российской Федерации

Федеральное агентство по образованию

КАЗАНСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. А.Н. Туполева - КАИ

Кафедра АСОИУ

Лабораторная работа №3

по дисциплине

«ТЕОРИЯ фОРМАЛЬНЫХ ГРАММАТИК И АВТОМАТОВ»

Выполнила:

Студентка группы 4309

Газимзянова А.Д.

Проверила: Бикмуллина И.И.

Казань 2023

**Лабораторная работа № 3. Разработка контекстно-свободного (КС) синтаксического анализатора**

**Теория**

*Синтаксический анализатор* — это программа или часть программы, которая анализирует входной поток данных (обычно текст), проверяя его на соответствие определенной грамматике.

*Левая рекурсия* — это ситуация, когда правило в формальной грамматике ссылается на само себя в первом символе правой части. Такая рекурсия вызывает проблемы при анализе кода, так как может привести к бесконечному циклу.

*Чтобы избавиться от левой рекурсии*, ее можно заменить на правую с помощью специального алгоритма. Один из наиболее распространенных — это алгоритм, предложенный Ахо и Ульманом.

Предположим, что у вас есть грамматика, которая содержит левую рекурсию:

A -> Aα | β

где A — это некоторый нетерминальный символ, α и β — последовательности терминальных и нетерминальных символов, и α не может быть пустой.

Алгоритм Ахо и Ульмана удаляет левую рекурсию, заменяя эту грамматику следующим образом:

A -> βA'

A' -> αA' | ε

где ε обозначает пустую строку.

Этот алгоритм применим для прямой левой рекурсии. Однако, если грамматика содержит косвенную левую рекурсию, алгоритм должен быть модифицирован или применен в сочетании с другими подходами.

**Текст задания:**

Для предложенного преподавателем варианта КС-грамматики разработать методом рекурсивного спуска синтаксический анализатор.

Примечание: здесь и далее через <1> и <2> обозначены слова из лабораторной работы №1.

Указания:

1. Лексический анализатор из лабораторной работы №1 должен быть расширен обработкой появившихся в КС-грамматике новых слов и включен в виде подпрограммы, поля класса или метода класса в синтаксический анализатор.

2. Оформить синтаксический анализатор в виде процедуры, функции или класса, которые при обращении обрабатывают весь исходный текст.

3. Если грамматика леворекурсивная, то устранить левую рекурсию.

4. При обнаружении лексической ошибки целесообразно возбуждать исключительную ситуацию, которая будет обрабатываться в главной форме программы.

**Грамматика (вариант 4):**

S → A :− B .

A → <2> (<1>)

B → B , A

B → A

B → B, A — левая рекурсия

**Избавление от левой рекурсии:**

S → A :− B .

A → <2> (<1>)

B → A | AC

C → , A | , AC

1. S → A :− B

**B** → <2> (<1>) :− A → **<2> (<1>) :− <2> (<1>)**

2. S → A :− B

**A** → A :− B , A → **<2> (<1>) :− A , <2> (<1>)** →

→ **<2> (<1>) :− <2> (<1>) , <2> (<1>)**

**Код программы:**

**Form1.cs**

using System;

using System.Windows.Forms;

using Lex;

using Synt;

namespace WindowsFormsApp2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

int n = textBox1.Lines.Length;

textBox1.Text = "acd(000010100):-acd(000010100)";

}

private void button1\_Click(object sender, EventArgs e)

{

textBox2.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = textBox1.Lines;

Synt.Lex.strPMessage = textBox2.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

textBox2.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Synt.Lex.intPSourceRowSelection; i++) n += textBox1.Lines[i].Length + 2;

n += Synt.Lex.intPSourceColSelection;

textBox1.SelectionLength = n;

}

}

}

}

**uLex.cs**

using System;

namespace Lex

{

public enum TState { Start, Continue, Finish };

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol };

public enum TToken { lxmIdentifier, lxmtz, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmrs, lxmDot, lxmComma, lxmdt, lxmMinus };

public class uLex

{

private String[] strFSource; // указатель на массив строк

private String[] strFMessage; // указатель на массив строк

public TCharType enumFSelectionCharType;

private char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

public String[] strPSource { set { strFSource = value; } get { return strFSource; } }

public String[] strPMessage { set { strFMessage = value; } get { return strFMessage; } }

public TState enumPState { set { enumFState = value; } get { return enumFState; } }

public String strPLexicalUnit { set { strFLexicalUnit = value; } get { return strFLexicalUnit; } }

public TToken enumPToken { set { enumFToken = value; } get { return enumFToken; } }

public int intPSourceRowSelection { get { return intFSourceRowSelection; } }

public int intPSourceColSelection { get { return intFSourceColSelection; } }

public uLex(){}

public void GetSymbol(bool check = false) //метод класса лексический анализатор

{

intFSourceColSelection++; // продвигаем номер колонки

if (intFSourceColSelection > strFSource[intFSourceRowSelection].Length - 1)

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

chrFSelection = strFSource[intFSourceRowSelection][intFSourceColSelection]; //классификация прочитанной литеры

if (chrFSelection == ' ') enumFSelectionCharType = TCharType.Space;

else if (chrFSelection >= 'a' && chrFSelection <= 'd') enumFSelectionCharType = TCharType.Letter;

else if (chrFSelection == '0' || chrFSelection == '1') enumFSelectionCharType = TCharType.Digit;

else if (chrFSelection == '-') enumFSelectionCharType = TCharType.ReservedSymbol;

else if (chrFSelection == ':') enumFSelectionCharType = TCharType.ReservedSymbol;

else if (chrFSelection == '(' || chrFSelection == ')' || chrFSelection == ',' || chrFSelection == '.' || chrFSelection == '/') enumFSelectionCharType = TCharType.ReservedSymbol;

else throw new System.Exception("Cимвол вне алфавита");

enumFState = TState.Continue;

}

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

while (enumFSelectionCharType == TCharType.Space || enumFSelectionCharType == TCharType.EndRow)

{

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol(true);

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol(true);

}

GetSymbol(true);

}

// Вариант 4

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | CFin |CFin|CFin| CFin |

// B | CFin | |CFin| CFin |

// CFin | CFin |CFin|CFin| CFin |

A:

{

if (chrFSelection == 'a' || chrFSelection == 'b' || chrFSelection == 'c' || chrFSelection == 'd')

{

TakeSymbol();

goto B;

}

else throw new Exception("Слово не должно начинаться с ab");

}

B:

{

if (chrFSelection == 'a' || chrFSelection == 'c' || chrFSelection == 'd')

{

TakeSymbol();

goto CFin;

}

else throw new Exception("Слово не должно начинаться с ab");

}

CFin:

{

if (chrFSelection == 'a' || chrFSelection == 'b' || chrFSelection == 'c' || chrFSelection == 'd')

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

case TCharType.Digit:

{

// 0 1

// A | B | |

// B | C | D |

// C | A | |

// D |FFin | |

// E |FFin | |

// G | E | |

// FFin | | G |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else throw new Exception("Ожидался 0");

B:

if (chrFSelection == '0')

{

TakeSymbol();

goto C;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидался 0");

C:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

FFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

else if (enumFSelectionCharType != TCharType.Digit) { enumFToken = TToken.lxmNumber; return; }

else throw new Exception("Ожидалась 1");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

else throw new Exception("Ожидался 0");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else if (enumFSelectionCharType != TCharType.Digit) { enumFToken = TToken.lxmNumber; return; }

else throw new Exception("Ожидался 0");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

{

while (enumFSelectionCharType != TCharType.EndRow)

GetSymbol();

}

GetSymbol();

}

if (chrFSelection == '(')

{

enumFToken = TToken.lxmLeftParenth;

GetSymbol();

return;

}

if (chrFSelection == ')')

{

enumFToken = TToken.lxmRightParenth;

GetSymbol();

return;

}

if (chrFSelection == '[')

{

enumFToken = TToken.lxmIs;

GetSymbol();

return;

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmrs;

GetSymbol();

return;

}

if (chrFSelection == ',')

{

enumFToken = TToken.lxmComma;

GetSymbol();

return;

}

if (chrFSelection == ':')

{

enumFToken = TToken.lxmdt;

GetSymbol();

return;

}

if (chrFSelection == '-')

{

enumFToken = TToken.lxmMinus;

GetSymbol();

return;

}

if (chrFSelection == '.')

{

enumFToken = TToken.lxmDot;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

**uSyntAnalyzer.cs**

using System;

using Lex;

namespace Synt

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

public String[] strPSource { set { strFSource = value; } get { return strFSource; } }

public String[] strPMessage { set { strFMessage = value; } get { return strFMessage; } }

public uLex Lex = new uLex();

public void S()

{

A();

if (Lex.enumPToken == TToken.lxmdt)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmMinus)

{

Lex.NextToken();

B();

if (Lex.enumPToken == TToken.lxmDot)

{

Lex.NextToken();

}

}

else throw new Exception("Ожидался минус");

}

else throw new Exception("Ожидалось двоеточие");

}

public void A()

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

Lex.NextToken();

}

else throw new Exception("Ожидалась )");

}

else throw new Exception("Ожидался числовой идентификатор");

}

else throw new Exception("Ожидалась (");

}

else throw new Exception("Ожидался буквенный идентификатор");

}

public void B()

{

A();

if (Lex.enumPToken == TToken.lxmComma)

C();

}

public void C()

{

if (Lex.enumPToken == TToken.lxmComma)

{

Lex.NextToken();

A();

if (Lex.enumPToken == TToken.lxmComma)

C();

}

}

}

}

**Результаты работы программы:**



