Лабораторная работа №3

Выполнила: студент группы 4309

Канькина О.А.

Синтаксический анализатор – это программное обеспечение, которое анализирует структуру текста и определяет его соответствие грамматике, что позволяет автоматически проверять и разбирать программный код, упрощая процесс разработки и отладки.

Синтаксический анализатор, также известный как парсер, является одной из ключевых компонент программного обеспечения, используемого в компиляторах и интерпретаторах. Он выполняет анализ входного текста, проверяет его на соответствие определенной грамматике и создает структуру данных, называемую синтаксическим деревом или абстрактным синтаксическим деревом (AST).

Синтаксический анализатор играет важную роль в процессе компиляции или интерпретации программного кода. Он принимает входной текст, который может быть написан на определенном языке программирования, и проверяет его на наличие синтаксических ошибок. Если текст соответствует грамматике языка, синтаксический анализатор создает структуру данных, которая представляет собой иерархическое представление программы.

Основная цель синтаксического анализатора – разбор входного текста и создание структуры данных, которая будет использоваться в дальнейшем для выполнения семантического анализа и генерации исполняемого кода. Он помогает программистам и разработчикам понять структуру программы и обнаружить возможные ошибки в синтаксисе.

Левая рекурсия устраняется путем преобразования грамматики в праворекурсивную грамматику.

Если имеются правила грамматики следующего вида

A -> A α | β,

то это леворекурсивная грамматика. Цепочка β не начинается с нетерминала A.

Левая рекурсия устраняется с помощью введения дополнительного нетерминала А':

А -> β А'

А' -> α А'

| ϵ,

где ϵ – пустая цепочка. Это праворекурсивная грамматика, которая порождает тот же язык,

что и исходная грамматика.

**Вариант 12**

**Текст задания**:

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

**Грамматика:**

S−−> <2> A <2> B

S−−> <1> A <1> B

A−−> <2> | <2> <1>

B−−> <2> B

B−−> <2>

Левая рекурсия отсутствует

S−−> <2> A <2> B−−> **<2> <2> <2> <2> <2> <2>**

S−−> <1> A <1> B−−>**<1> <2> <1> <2>**

Код программы:

UsyntAnalyzer

using System;

using System.Windows.Forms;

using nsLex;

namespace nsSynt

{

class Analyzer

{

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 CLex Lex = new CLex();

public void S()

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

A();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

B();

}

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

}

else if (Lex.enumPToken == TToken.lxmNumber) // добавила еще одно условие

{

Lex.NextToken();

A();

if (Lex.enumPToken == TToken.lxmNumber)

{

Lex.NextToken();

B();

}

}

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

}

public void A()

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

Lex.NextToken();

B();

}

}

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

B();

}

}

else throw new Exception("Текст верный");

}

public void B()

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

A();

}

}

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

}

}

}

ULex

using System;

namespace nsLex

{

public enum TState { Start, Continue, Finish };

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

public enum TToken { lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmText, lxmtz, lxmdt, lxmr, lxmrs, lxmls, lxmplus, lxmmul };

public class CLex

{

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 CLex()

{

}

public void GetSymbol()

{

intFSourceColSelection++;

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];

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 == '/' || chrFSelection == '\*' || chrFSelection == '+') enumFSelectionCharType = TCharType.ReservedSymbol;

else if (chrFSelection == '<' || chrFSelection == '>' || chrFSelection == '[' || chrFSelection == ']') enumFSelectionCharType = TCharType.ReservedSymbol;

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

enumFState = TState.Continue;

}

}

public void GetSymbol(bool check)

{

intFSourceColSelection++;

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];

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

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

else if (chrFSelection >= 'а' && chrFSelection <= 'Я') enumFSelectionCharType = TCharType.Letter;

else if (chrFSelection == '0' || chrFSelection == '9') enumFSelectionCharType = TCharType.Digit;

else if (chrFSelection == '/' || chrFSelection == '\*' || chrFSelection == '+') enumFSelectionCharType = TCharType.ReservedSymbol;

else if (chrFSelection == '<' || chrFSelection == '>' || chrFSelection == '[' || chrFSelection == ']') enumFSelectionCharType = TCharType.ReservedSymbol;

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 12

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

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

// AFin |BFin | | | |

// BFin |CFin |CFin|CFin|CFin|

A:

{

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

{

TakeSymbol();

goto CFin;

}

else if (chrFSelection == 'a')

{

TakeSymbol();

goto B;

}

else throw new Exception("Ошибка в идентификаторе");

}

B:

{

if (chrFSelection == 'a')

{

TakeSymbol();

goto BFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

BFin:

{

if (chrFSelection == 'a')

{

TakeSymbol();

goto BFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

CFin:

{

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

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

case TCharType.Digit:

{

A:

if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else throw new Exception("Ожидалось 1");

B:

if (chrFSelection == '0')

{

TakeSymbol();

goto C;

}

else throw new Exception("Ожидалось 1");

C:

if (chrFSelection == '1')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидалась цифра 1");

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto F;

}

else throw new Exception("Ожидалось 0");

F:

if (chrFSelection == '0')

{

TakeSymbol();

goto GFin;

}

else throw new Exception("Ожидалось 0");

GFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto H;

}

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

else throw new Exception("Ожидалось 0");

H:

if (chrFSelection == '0')

{

TakeSymbol();

goto J;

}

else throw new Exception("Ожидалось 0");

J:

if (chrFSelection == '1')

{

TakeSymbol();

goto GFin;

}

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

else throw new Exception("Ожидалось 0");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '[')

{

enumFToken = TToken.lxmls;

TakeSymbol();

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmrs;

TakeSymbol();

}

if (chrFSelection == '+')

{

enumFToken = TToken.lxmplus;

TakeSymbol();

}

if (chrFSelection == '\*')

{

enumFToken = TToken.lxmmul;

TakeSymbol();

}

if (chrFSelection == '/') //если указатель на сам символ равен /

{

GetSymbol(true);

if (chrFSelection == '/') //если указатель на сам символ равен /

{

while (enumFSelectionCharType != TCharType.EndRow)

GetSymbol(true);

}

GetSymbol(true);

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

uMainForm:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using nsSynt;

using nsLex;

namespace nsLexMainForm

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

int n = tbFSource.Lines.Length;

tbFSource.Text = "100a100a";

}

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

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

}

}

Результат:





