**Лабораторная работа №8**

Выполнила: студент группы 4309

Курбанов Д.И.

Семантический анализатор: он использует синтаксическое дерево и таблицу символов, чтобы проверить, соответствует ли данная программа семантически определению языка. Он собирает информацию о типе и сохраняет ее либо в синтаксическом дереве, либо в таблице символов. Эта информация о типе впоследствии используется компилятором при генерации промежуточного кода.

Следующий шаг анализа текста программы – семантический, существенно отличается от синтаксического. И дело не столько в том, что фаза семантического анализа реализуется не формальными, а содержательными методами (т. е. на данный момент нет универсальных математических моделей и формальных средств описания «смысла» программы). Лексический и синтаксический анализ имеют дело со структурными, т. е. внешними, текстовыми конструкциями языка. Семантика же, ориентированная на содержательную интерпретацию, имеет дело с внутренним представлением «смысла» объектов, описанных в программе. Для любого, имеющего опыт практического программирования, ясно, что формальные конструкции языка дают описание свойств и действий над внутренними объектами, с которыми имеет дело программа. Для начала перечислим все, что их касается и лежит на поверхности:

· большинство объектов являются именованными. Имя объекта позволяет его идентифицировать, существуют различные области действия имен, соглашения об именах, различные умолчания и т. п.. Все это относится к семантике;

· виды, сложность и набор характеристик объектов различаются в разных языках программирования и сильно зависят от области приложения языка (в этом смысле семантика языков программирования более разнообразна, нежели синтаксис и лексика). Например, классический Си, ориентированный на максимальное приближение к архитектуре компьютера, работает с такими объектами, как типы данных, переменные, функции. Все они имеют различные свойства и характеристики. Например, переменная характеризуется именем, типом данных, размерностью, областью действия, временем жизни, текущим значением;

· объекты связаны между собой (ссылаются друг на друга). В том же Си переменная ссылается на описание того типа данных, к которому она относится, далее производный тип данных ссылается на базовый и т. п.. Можно сказать, что семантика программы во внутреннем представлении выглядит как система взаимосвязанных объектов;

· внутреннее представление семантики программы не совсем удачно называется семантическими таблицами. На самом деле структура данных, соответствующая представлению семантики, может быть любой. Термин «таблицы» говорит о том, что имеются множества объектов различных типов, для каждого из которых заведена отдельная таблица, но нельзя забывать, что элементы различных таблиц связаны между собой. Наиболее близкий термин для описания подобной системы – база данных.

Семантика программы – внутренняя модель (база данных) множества именованных объектов, с которыми работает программа, с описанием их свойств, характеристик и связей.

Семантика программы тоже не обладает структурной целостностью и представлена фрагментарно, но при этом связана с синтаксисом следующим образом:

· один и тот же семантический объект (например, переменная) может встречаться в различных, синтаксически несвязанных частях программы;

· синтаксические конструкции описаний, определений и объявлений являются источником семантики объектов программы, они «заявляют» о существовании объектов и задают их свойства;

· синтаксические конструкции, связанные с действиями, выполняемыми над объектами, являются потребителями семантики, их интерпретация, корректность, «смысл» зависят от семантических свойств объекта. Забегая вперед, можно заметить, что заключительная фаза трансляции (генерация кода, интерпретация) может рассматриваться как особые семантические действия, производимые над объектами;

· первичным источником семантики является лексический анализ. Некоторые из лексем (например, идентификаторы и константы) наряду с классом лексемы (обозначение выходной единицы лексического анализа), т. е. символом (обозначение той же единицы на входе синтаксического анализатора) имеют значение. Значением лексемы является сама распознанная цепочка литер, она и представляет семантическую составляющую лексемы, которая попадает в семантические таблицы;

· лексемы, или то же самое, что терминальные символы входной строки (в терминах синтаксического анализа), ссылаются в семантические таблицах на свою семантику. В дальнейшем каждый промежуточный нетерминал также ссылается на собственную семантику. При этом любое правило преобразует семантику терминалов и нетерминалов правой части в семантику нетерминала левой части при помощи назначенной правилу семантической процедуры. Таким образом, формирование семантической составляющей связано с движением снизу вверх по синтаксическому дереву, от вершин – потомков к предкам.

В процессе семантического анализа проверяется наличие семантических ошибок в исходной программе и накапливается информация о типах для следующей стадии – генерации кода. При семантическом анализе используются иерархические структуры, полученные во время синтаксического анализа для идентификации операторов и операндов выражений и инструкций.

Важным аспектом семантического анализа является проверка типов, когда компилятор проверяет, что каждый оператор имеет операнды допустимого спецификациями языка типа. Например, определение многих языков программирования требует, чтобы при использовании действительного числа в качестве индекса массива генерировалось сообщение об ошибке. В то же время спецификация языка может позволить определенное преобразование типов, например, когда бинарный арифметический оператор применяется к операндам целого и действительного типов. В этом случае компилятору может потребоваться преобразование целого числа в действительное.

Все типы известны во время компиляции, и это относится к типам выражений, идентификаторам и литералам. При этом неважно, насколько сложным является выражение: его тип может определяться во время компиляции за определенное количество шагов исходя из типов его составляющих. Фактически, это позволяет производить контроль типов во время компиляции и находить в процессе компиляции многие программные ошибки.

В процессе компиляции анализатору требуются таблицы:

– таблица символов;

– таблица типов;

– таблица функций;

– таблица меток.

Основная задача таблицы символов – установление соответствия между переменной и ее типом. Операции, связанные с таблицей символов:

– операция, соответствующая определяющему вхождению переменной. В таблицу символов помещаются имя переменной и ее тип;

– операция, соответствующая применимому вхождению переменной. Исследуется таблица символов для нахождения типа переменной.

Как известно, один и тот же синтаксис может быть представлен различными грамматиками. Они эквивалентны между собой, поскольку распознают одно и то же множество цепочек (Как известно, проблема эквивалентности формальных грамматик алгоритмически неразрешима, т. е. невозможно разработать алгоритм, устанавливающий эквивалентность двух произвольных грамматик). Мы уже видели, что требования к грамматикам восходящего и нисходящего разбора также различны. Различны и технологически подходы к реализации одних и тех же элементов синтаксиса (повторений, необязательных элементов синтаксиса и т. п.). Поэтому получается, что в различных грамматиках при разборе одного и того же предложении будут строиться различные деревья, но результат трансляции (например, последовательность выполняемых операций в выражении) должен быть если не одинаковым, то эквивалентным(разобрать методы и описать свой, задание написать).

Ошибки, распознанные семантическим анализатором, заключаются в следующем:

1. Несоответствие типов
2. Необъявленные переменные
3. Неправильное использование идентификатора

В данной лабораторной работе был использован третий пункт, я прописала условия для идентификатора, при неправильном использовании которых выдавало сообщение об ошибке

Функции семантического анализа:

1. Проверка типов – Гарантирует, что типы данных используются в соответствии их определением.
2. Проверка меток – Программа должна содержать ссылки на ярлыки.
3. Проверка управления потоком – Проверяет, что управляющие структуры используются надлежащим образом.

**Задание**. Создать семантический анализатор.

Код программы:

uSemantAnalyzer.cs:

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

class uSemantAnalyzer

{

public int i = 0;

public string strIndentifier;

public string strDigital;

private TreeView tree;

public uSemantAnalyzer()

{

}

public uSemantAnalyzer(TreeView treeView)

{

tree = treeView;

TreeController(tree);

}

public void TreeController(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

TreeController(node);

}

}

public void TreeController(TreeNode node)

{

if (node.Text == "A")

{

if (node.Nodes.Count > 1)

{

strIndentifier = node.Nodes[1].Nodes[0].Text.ToString();

strDigital = node.Nodes[2].Nodes[0].Text.ToString();

Check(strIndentifier, strDigital, node);

}

}

foreach (TreeNode childNode in node.Nodes)

{

TreeController(childNode);

}

}

private void Check(string ident, string digit, TreeNode node)

{

int identLen = ident.Length;

if (ident.Count(x => x == 'a') < digit.Count(x => x == '0'))

{

tree.SelectedNode = node;

tree.SelectedNode.BackColor = Color.Red;

throw new Exception("Условие не выполнено");

}

}

}

}

Form1.cs:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

Dictionary<int, List<string>> hashTableIdentifier = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableDigital = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableRezerv = new Dictionary<int, List<string>>();

public MyHashFunction hashFunction = new MyHashFunction();

public Form1()

{

InitializeComponent();

//textBox2.AppendText("010000001 010000001 caaaa");

textBox2.AppendText("010000001 ca caa 000 caaa");

int n = textBox2.Lines.Length;

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void Form1\_Load\_1(object sender, EventArgs e)

{

}

private void label5\_Click(object sender, EventArgs e)

{

}

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

{

if (SyntTree.Nodes.Count > 0)

{

SyntTree.Nodes.Clear();

SyntTree2.Nodes.Clear();

}

textBox4.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer(SyntTree);

// uSyntAnalyzer Synt2 = new uSyntAnalyzer(SyntTree2);

Synt.Lex.strPSource = textBox2.Lines;

Synt.Lex.strPMessage = textBox4.Lines;

Synt.Lex.enumPState = TState.Start;

Synt.Lex.hashTableRezerv = hashTableRezerv;

try

{

TreeNode parent = new TreeNode("TreeView");

// TreeNode parent2 = new TreeNode("TreeView2");

SyntTree.Nodes.Add(parent);

//SyntTree2.Nodes.Add(parent2);

Synt.S(parent);

// Synt2.S(parent);

Synt.Lex.NextToken();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

textBox4.Text += exc.Message;

textBox2.Select();

textBox2.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Synt.Lex.intPSourceRowSelection; i++) n += textBox2.Lines[i].Length + 2;

n += Synt.Lex.intPSourceColSelection;

textBox2.SelectionLength = n;

}

Generator.CopyTree(SyntTree, SyntTree2);

Generator.RefactorTree(SyntTree2);

SyntTree.ExpandAll();

SyntTree2.ExpandAll();

}

private void button2\_Click(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = textBox2.Lines;

Lex.strPMessage = textBox4.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox2.TextLength;

int y = textBox2.Lines.Length;

textBox4.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

listBox4.Items.Add(Lex.strPLexicalUnit);

s1 = "id " + Lex.strPLexicalUnit;

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

listBox5.Items.Add(Lex.strPLexicalUnit);

s1 = "num " + Lex.strPLexicalUnit;

break;

}

case (TToken.lxmtz):

{

hashFunction.AddWord(hashTableRezerv, "\_");

listBox6.Items.Add(Lex.strPLexicalUnit);

s1 = "rez \_" + Lex.strPLexicalUnit;

break;

}

}

String m = "(" + s + "" + s1 + ")";

textBox4.Text += m;

}

}

catch (Exception exc)

{

textBox4.Text += exc.Message;

textBox2.Select();

textBox2.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += textBox2.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

textBox2.SelectionLength = n;

}

}

private void button3\_Click(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, listBox4.SelectedItem.ToString()) == 1)

{

button3.BackColor = Color.Green;

}

else

{

button3.BackColor = Color.Red;

}

}

private void button4\_Click(object sender, EventArgs e)

{

hashFunction.AddWord(hashTableIdentifier, textBox3.Text.ToString());

}

private void button5\_Click(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox4.SelectedItem.ToString()))

{

button5.BackColor = Color.Green;

}

else

{

button5.BackColor = Color.Red;

}

}

private void button6\_Click(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox4.SelectedItem.ToString()))

{

hashFunction.AddWord(hashTableIdentifier, textBox3.Text.ToString());

button6.BackColor = Color.Green;

}

else

{

button6.BackColor = Color.Red;

}

}

private void button7\_Click(object sender, EventArgs e)

{

listBox4.Items.Clear();

listBox5.Items.Clear();

listBox6.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

listBox4.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableDigital)

{

listBox5.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableRezerv)

{

listBox6.Items.Add(string.Join(", ", entry.Value));

}

}

private void label7\_Click(object sender, EventArgs e)

{

}

private void listBox6\_SelectedIndexChanged(object sender, EventArgs e)

{

}

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

{

{

TreeConverter treeConverter = new TreeConverter();

treeConverter.ConvertBinaryToDecimal(SyntTree);

}

}

private void SyntTree2\_AfterSelect(object sender, TreeViewEventArgs e)

{

}

}

}

uSyntAnalyzer.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

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 CLex Lex = new CLex();

TToken first;

bool init;

TreeView tree;

public uSyntAnalyzer(TreeView treeView)

{

tree = treeView;

}

public void S(TreeNode highParent)

{

TreeNode parent = new TreeNode("S");

highParent.Nodes.Add(parent);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

A(parent);

S(parent);

}

else if (Lex.enumPToken == TToken.lxmIdentifier)

{

if (IsIdentifierUnique(Lex.strPLexicalUnit))

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

}

else throw new Exception("идентификатор повторяется");

}

else throw new Exception("Ожидался число или идентификатор");

}

public void A(TreeNode highParent)

{

TreeNode parent = new TreeNode("A");

highParent.Nodes.Add(parent);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

if (IsIdentifierUnique(Lex.strPLexicalUnit))

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

S(parent);

A(parent);

uSemantAnalyzer uSemant = new uSemantAnalyzer(tree);

}

else throw new Exception("идентификатор повторяется");

}

else if (Lex.enumPToken == TToken.lxmNumber)

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

}

else throw new Exception("Ожидался число или идентификатор");

}

private bool IsIdentifierUnique(string identifier)

{

foreach (TreeNode node in tree.Nodes)

{

if (!IsIdentifierUniqueInNode(identifier, node))

{

return false;

}

}

return true;

}

private bool IsIdentifierUniqueInNode(string identifier, TreeNode node)

{

if (node.Text == identifier)

{

return false;

}

foreach (TreeNode child in node.Nodes)

{

if (!IsIdentifierUniqueInNode(identifier, child))

{

return false;

}

}

return true;

}

}

}

Результат работы программы:

